Technical Countermeasure Report

sigesweb

Report generated by:	Maria Jose Rotter
Unique ID:	sigesweb
Workflow State:	

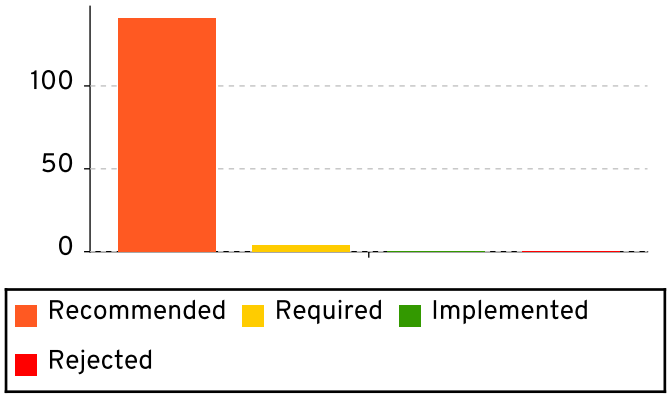
Index

Summary

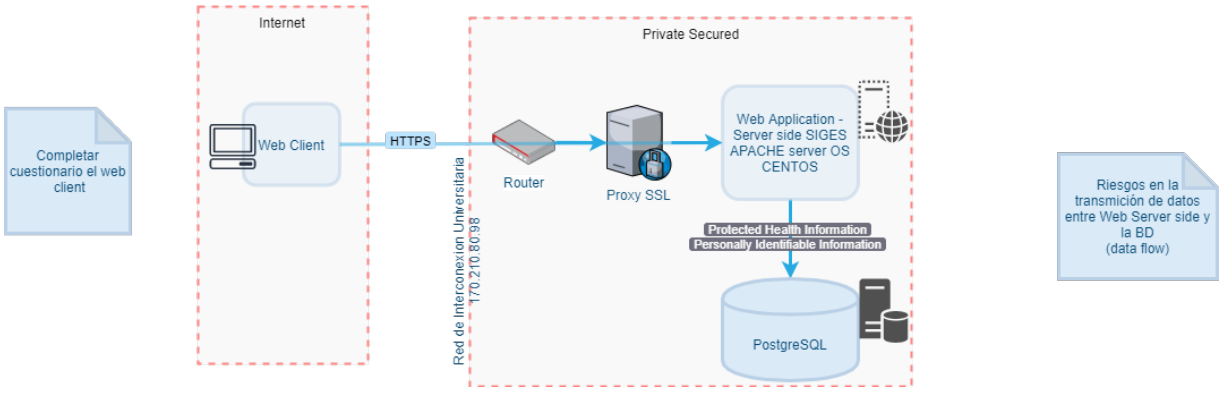
Shown below is a brief description of the product and summary analysis of the risks.

Product name:	sigesweb
Unique ID:	sigesweb
Product description:	
Gestiona cuentas de usuarios de la obra social.	
Business unit:	bu_user_mariajoserotter@gmail.com
Owner:	Maria Jose Rotter

Countermeasures Summary



Architectural Diagrams



Filename: irius-risk-diagram-architecture-image.png
Username: mariajoserotter@gmail.com **Date Uploaded:** Jun 24, 2021 8:20:49 PM

Required Countermeasures

Component: PostgreSQL

Countermeasure name	Test result	Related threats
Require authentication before presenting restricted data	No probado	• An attacker gains access to an application, service, or device, with the privileges of an authorized or highly privileged user by evading or circumventing an authentication mechanism
Encrypt data between the client and server/service	No probado	• Authentication credentials compromised through network sniffing

Component: Web Application - Server side SIGES APACHE server OS CENTOS

Countermeasure name	Test result	Related threats
Require authentication before presenting restricted data	No probado	• An attacker gains access to an application, service, or device, with the privileges of an authorized or highly privileged user by evading or circumventing an authentication mechanism
Encrypt data between the client and server/service	No probado	• Attackers gain control of the connection through a Man In The Middle attack • Authentication credentials compromised through network sniffing • Sensitive data is compromised through network sniffing attacks • An attacker can obtain a valid session identifier to impersonate legitimate users

Countermeasure Test Results

Appendix A: Countermeasure Details

This appendix shows all of the countermeasures mitigating the threats found in the project.

Component: PostgreSQL

Id	Name	Description	State	Result
CDS-USER-TRACK	Log details of user actions within the system	To maintain proper accountability, logs should be maintained with sufficient information to track user actions within the system. These logs should be forensically sound, non-repudiable, and contain comprehensive details about activity. While the exact data for an event may vary, the following should be captured at a minimum: • Timestamps against a proven external source (e.g. an NTP server) • Origin, with this field we mark if the logs are provided by a trusted or untrusted source. • Event, status, and/or error codes (with sensitive data masked as appropriate or not introduced in logs) • Service, command, application or function name and details • User or system account associated with an event • Devices used (e.g. source and destination IPs, terminal session ID, web browser, etc) Source: https://security.berkeley.edu/security	Recommended	No probado

CWE-311-AT-REST	Encrypt data stored on the host (data at rest)	Data stored on the server or the client must be protected by encryption (data *at rest*). • Cryptographically strong symmetric or asymmetric (public-key) encryption should be used to protect the data. • Encryption should be performed before the data is written to disk or other persistent storage. • The key for encrypting and decrypting the data should *not* be accessible from the same host. • The encryption and decryption operation should be performed on a different host. • A recognized, proven, and tested implementation/library should be used (in preference to a bespoke implementation).	Recommended	No probado
CWE-662	Use a synchronised time source	In order to correlate logs and data from different internal and external systems, and to preserve forensic quality of the logs, it is important a unified and trusted synchronized time source is used throughout the environment. • Servers should synchronize to an internal or external NTP server • The centralized source should in turn use (or be) a trusted central time source. This control is critical in identifying application events (including attacks) through logging, and in conducting post-event analysis, and in particular to track the entire user (or attacker) journey through the system should it be compromised. It is good practice to use the concept of Indicators of Compromise (IoC) which should be leveraged to detect possible situations in which the system has been compromised and give an appropriate response. IoCs are often tracked through logs, and accurate time is essential.	Recommended	No probado

CWE-89-PREPARED	Use prepared statements for all database queries	Database injection attacks, such as SQLi (SQL Injection) rely on sending tainted client-side data which is used in dynamic SQL queries on the server-side, in an unsafe manner. Creating queries by concatenating strings using untrusted data may result in vulnerable code; for example, an attacker may append an 'OR' statement to the customerName parameter in order to bypass checks and return additional data from the database: • String query = "SELECT user FROM users WHERE name = '" + request.getParameter("customerName")+"'"; Using prepared statements with carefully controlled and validated input conditions mitigates against SQLi and related attacks. • Database queries should always be executed using prepared statements or parameterized queries. • Queries through an Object-Relational mapper should also be treated as tainted input, and again executed using prepared statements	Recommended	No probado
EU-GDPR-BACKUP	Implement a Backup and Recovery process	Ensure backup policy is active and tested.The policy should describe the required recovery time objective (RTO) and recovery point objective (RPO) so that the availability of personal data can be restored in a timely manner (based upon the requirements specified by the DPO/CISO). Ensure an SLA has been defined for data availability. How 'timely manner' will be interpreted depends on your SLA.	Recommended	No probado

EU-GDPR-ENCRYPT-PERSONAL-DATA	Encrypt personal data	Implement encryption at rest (see guidance below) or give risk-based explanation why encryption was not implemented. Use well known encryption libraries, taking into account the data use, and do not invent your own. • personal data must be encrypted • data that is not used by the application (e.g. passwords, ...) should be hashed so they cannot be recovered easily	Recommended	No probado
PATCH-SERVICE	Apply required security patches to the service	Vendors and other maintainers of software release patches in response to security flaws and other bugs in their products. The longer a system is exposed with a known security vulnerability, the easier to compromise it. As the exploit enters the public domain, they get included in automated exploitation suites like Metasploit and a wider less skilled miscreant is able to leverage them. • Apply patches and other software updates in a timely manner to prevent unexpected failures or exploitation. • Clearly define an approach for testing and applying patches, in particular security patches, with expected timescales. There is often a small window between the release of a patch, and potentially malicious actors reverse-engineering the patch to identify and exploit the flaw. • Use a threat intelligence, vulnerability scanning, or other alerting services to ensure the project team is promptly aware of issues within the project or its components.	Recommended	No probado

PROPER-REVOCATION-CERTIFICATE	Configure and enable appropriate certification revocation	Configure and enable appropriate certification revocation for each certificate created, such as Online Certificate Status Protocol (OCSP) Stapling. OCSP is a protocol to check if an SSL certificate has been revoked. Instead of the client downloading a large list of revoked certificates, they can simply submit a request to a CA server, which returns a signed response with the certificate current status.	Recommended	No probado
RESTRICT-ACCESS-DATABASE	Access the data store from an account with the least privileges necessary	Use an account with only the minimum set of permissions required to access the data store. The account should not be able to perform operations that are not explicitly required by the component that performs these operations. For example, if a web application needs to read data from certain tables and insert and update data from others, then a database account with only those specific permissions should be used by the application server.	Recommended	No probado
RESTRICT-SERVICE	Restrict access to the service at the network layer to reduce exposure	Access to APIs should be restricted to expected sources, limiting exposure of the service and its attack surface, and therefore likelihood of a malicious party gaining access to the system. • Apply network layer security controls so that only the necessary and expected IP addresses are permitted access to connect to the service.	Recommended	No probado

TLS-STRONG-CIPHERS	Require cryptographically strong TLS cipher suites	Only cryptographically strong ciphers should be required. Best-practice dictates a subset of ' *known good*' ciphers and protocols be defined and enforced on the server. This may, however, have compatibility issues with older browsers, requiring a balance be sought between accessibility and security. • Define and enforce a list of acceptable ciphers and protocols • Explicitly disable known-bad ciphers and protocols, such as: • Null and export ciphers • DH, MD5 and other weak cryptography • Ciphers with keys less than 128 bits • CBC ciphers with TLSv1.0 or earlier	Recommended	No probado
TLS-STRONG-PROTOCOLS	Require cryptographically secure protocols (e.g. TLSv1.2 and above)	Only cryptographically strong ciphers should be required. Best-practice dictates a subset of 'known good' ciphers and protocols be defined and enforced on the server. This may, however, have compatibility issues with older browsers, requiring a balance be sought between accessibility and security. • Define and enforce a list of acceptable ciphers and protocols. Disable SSLv3 and earlier protocols on the service. • Ideally only TLSv1.2 and newer should be supported • If TLSv1.1 or 1.0 are required, known secure configurations and ciphers should be selected. • SSLv3.0 and earlier should not be used	Recommended	No probado

require-use-strong-passwords	Require the use of strong passwords	Passwords used either as sole verification credentials, or as part of a multi-factor authentication, are a key aspect of application security, and strong password selection should be encouraged and enforced. The application should allow flexibility in user password selection, and enforce minimum criteria for password quality. This should include: • Minimum password length requirements, to mitigate brute-force and dictionary attacks. • Encourage use of pass-phrases using multiple words, achieving longer passwords more resistant to attack. • Enforce use of mixed case, numeric and/or special characters to increase complexity. • Prevent or discourage use of dictionary words and common passwords through black-lists. For example, a set of commonly used passwords can be found on SecLists at https://github.com/danielmiessler/SecLists/tree/master/Passwords Password length: Password length considers the minimum and maximum length of characters comprising the password of your users. For ease of changing this length, its implementation can be configurable possibly using a properties file or xml configuration file. • Minimum length. • Memory secrets shall be at least 8 characters long. • Memory secrets generated automatically shall be at least 6 numeric characters. • Maximum length. People tend to forget their passwords easily. The longer the password, the more likely people are to enter it incorrectly in the system. However, long pass-phrases can be easily remembered, and should not be prevented through unnecessarily strict upper restrictions on length. Passwords with 64 characters or longer should be permitted. Password Complexity:	Recommended	No probado

• Passwords with consecutive multiple spaces should be coalesced and converted into only one space. After this modification, the password length should be at least 12 characters long.
• Unicode characters should be allowed in the password. A single Unicode code point is considered a character.
• Reject those passwords commonly used and that have been leaked in a previous compromise. You may choose to block the top 1000 or 10000 most common passwords which meet the above length requirements and are found in compromised password lists. The following link contains the most commonly found passwords: https://github.com/danielmiessler/SecLists/tree/master/Passwords
• Actual passwords must not be stored, to protect against brute forcing if the database is compromised.
Password Topologies:

• Ban commonly used password topologies.
• Force multiple users to use different password topologies.
• Require a minimum topology change between old and new passwords.
Additional Information:

• Make sure that every character the user types in is actually included in the password. We've seen systems that truncate the password at a length shorter than the user provided (e.g., truncated at 15 characters when they entered 20).
• As application's require more complex password policies, they need to be very clear about what these policies are. The required policy needs to be explicitly stated on the password change page
• If the new password doesn't comply with the complexity policy, the error message should describe EVERY complexity rule that the new password does not comply with, not just the 1st rule it doesn't comply with.

store-backups-securely	Encrypt Backups securely on the host (data at rest)	Sensitive data and Backups of sensitive data shall be stored securely by encryption (data at rest). • Cryptographically strong symmetric or asymmetric (public-key) encryption should be used to protect the data. • Encryption should be performed before the data is written to disk or other persistent storage. • The key for encrypting and decrypting the data should not be accessible from the same host. • The encryption and decryption operation should be performed on a different host. • A recognized, proven, and tested implementation/library should be used (in preference to a bespoke implementation).	Recommended	No probado
CWE-306-SERVICE	Require authentication before presenting restricted data	The application should ensure users have undergone an Identification and Verification (ID&V) process before allowing access to secret, sensitive or otherwise restricted data. For less sensitive but still restricted data, simple verification of the location of the user may suffice (e.g. IP restrictions). • For non-sensitive but non-public data, access could be restricted by IP address, limiting access to internal networks, workstations, or gateways • For more sensitive data, TLS client-side certificates may be appropriate • Where secret or other sensitive data is handled, a full authentication process to identify and validate users with single or multi-factor authentication may be required	Required	No probado

CWE-319-TRANSPORT	Encrypt data between the client and server/service	Data passed between the client and server should be protected by encryption in transit. • Implement cryptographically strong TLS end-to-end encryption between the client and server, terminating within a secure environment on the server-side. • Consider use of client certificates to prevent interception of (or man-in-the-middle attacks on) the encrypted connection. • Alternatively, asymmetric (public-key) encryption could be utilized and a recognized, proven, and tested implementation/library should be used	Required	No probado

Component: Web Application - Server side SIGES APACHE server OS CENTOS

Id	Name	Description	State	Result
ACCESS-CONTROL-CENTRALIZED	Ensure that access control mechanisms are centralized	Access control, authentication and authorization procedures should be centralized, and all roles, permissions, groups and users shall be controlled - or gain access - through this mechanism. To ensure roles, groups and permissions are assigned properly to users, we need to have centralized access control, to grant or deny access easily and securely.	Recommended	No probado
ASSIGN-WHITE	Use a white-list approach to assign values to variables	All data received from the client-side should be considered tainted and a potential risk, regardless of the source or transport method. Using mass-assignment or auto-binding when accepting values from an untrusted source and mapping them to trusted server side values increases the risk of tainted data being used unsafely. • Bind individual values to specific fields, instead of binding whole objects. • Use available features of the language or framework that allow specification of white lists of attributes or fields that are allowed to be modified. • For example, applications written with Ruby on Rails can use the attr_accessible (white list) or attr_protected (black list) macros in each class that may be used in mass assignment.	Recommended	No probado
ASVS-11.5	Do not share system information in HTTP headers and responses	The application should not share detailed version information of system components or other information that would assist attackers in the HTTP header or HTTP responses.	Recommended	No probado

ASVS-16.6	Do not store user files under the web root	Storage of user-controlled content in folders accessible through the web-server increases the risk of exposure, and also facilitates attacks relying on the upload and execution, or distribution, of malicious content. In the case of proprietary user data, it may also expose the restricted data to compromise through direct browsing or direct object reference manipulation if access controls are not securely implemented • Store uploaded files and other content in segregated folders outside of the web root directory tree to prevent direct access. • File content that is intended to be accessed by users from the client-side should be streamed with appropriate MIME types. • Ensure appropriate permissions are applied to the files and the folders; they should only be readable by the relevant service accounts, and should not be executable. • Ideally, the folder containing the files should not be accessible through the web-application even in the event of a directory traversal or server-side file include vulnerability.	Recommended	No probado
ASVS-18.8	Reject requests containing unexpected or missing content type headers	Reject requests containing unexpected or missing content type headers with HTTP response status "406 Unacceptable" or "415 Unsupported Media Type".	Recommended	No probado

ASVS-2.17	Do not transmit password in email or other unencrypted channels	Data passed between the user and application without appropriate encryption, in particular in emails, may be intercepted or monitored by malicious parties. • Do not send plaintext passwords in emails or other unencrypted transport (e.g. a web page server over HTTP). • Password recovery processes should not send either the old or a new password to users (the former is particular bad due to the penchant for people to reuse the same password across multiple systems or applications). • If the recovery process is built on sending a temporary random password to users, the risk can be partly mitigated (but not eliminated) by: • Ensuring the random password is single use (one time password), and forcing the user to change the password on first login. • Making the password valid only for a short-time (time limited). • Requiring a further validation or identification step when the time limited one time password is used.	Recommended	No probado
ASVS-2014-3.12-DOMAIN	Session cookie domain attributes should be restricted to prevent exposure	The session token value issued after users have successfully identified and authenticated themselves is of equivalent value to the secrets the user presents to login, and must be protected accordingly. If the cookie domain attribute is too liberal, the cookie may be accessible to other hosts within the parent domain. This is a particular issue in multi-tenanted hosting, or where applications are delivered as subdomains of a third party parent domain. • The domain attribute for the session cookie should be restricted to the fully qualified hostname on which the application is running. • For example, the cookie domain should be set to myapp.example.com rather than .example.com • e.g. Set-Cookie: session=token; Path=/app; **Domain=myapp.example.com**; secure; HTTPOnly	Recommended	No probado

ASVS-2014-3.12-PATH	Session cookie path attributes should be restricted to prevent exposure	The session token value issued after users have successfully identified and authenticated themselves is of equivalent value to the secrets the user presents to login, and must be protected accordingly. If the cookie path attribute is not appropriately restricted, the session token may be accessible from other applications or content on the server. • The path attribute for the session cookie should be restricted to the application that requires it. • For example, if the application is located at: "myapp.example.com/theapp". Then the path should e set to: "/theapp/".	Recommended	No probado

ASVS-2014-3.6	Session ID's should be transmitted securely	The session token value issued after users have successfully identified and authenticated themselves is of equivalent value to the secrets the user presents to login, and must be protected accordingly. Data passed in the URL (address bar) between client and server is likely to be exposed in logs on the server, or intermediate devices (such as proxies or other network devices), or in local browser logs/history. A common mistake is, for example, to pass a SessionID value as a parameter in the URL of a get request between hosts. To mitigate the risk of exposure of sensitive data, it should only be sent in the body of an HTTP message (for example a POST request), or in the HTTP headers (either standard headers such as the Cookie values, or a custom X- header). • Ensure sensitive, private, or otherwise restricted data is not sent in a URL value 'GET' parameter. • Specifically, never send the session ID token in the URL or over unencrypted transport. • Pass data only in appropriately protected fields in the body or headers, for example: • A session Cookie (e.g. JSESSIONID, ASP.NET_SessionID) • A POST parameter (e.g. a token in the HTTP request body, or JSON parameter) • A standard or bespoke HTTP header (Authorization: X-Session header)	Recommended	No probado
ASVS-8.1	Ensure sensitive data is not revealed through error output messages.	Ensure the application does not output error messages or stack traces containing sensitive data that could assist an attacker, including session id, software/framework versions and personal information.	Recommended	No probado

ASVS-8.2	Deny access by default if a security control fails with an exception.	If a security control fails, the application should not allow the user access to the application. Therefore, if in the security control an exception is launched the system should stay in a secure state, meaning access is denied to the user. Conversely, access will only be granted if there is no security failure / exception.	Recommended	No probado
ASVS-8.8	Escape meta-characters from un-trusted data	If untrusted data, including any data received from the client side of a connection is directly written to a log file, then this data could contain newline or other meta-characters that may allow an attacker to forge log entries. Such meta-characters should first be escaped or removed before the data is written to the logging system.	Recommended	No probado

AV-DoS-BH	Use blackholing to mitigate L4 DDoS attacks	DDoS attack blocking, commonly referred to as blackholing, is a method typically used by ISPs to stop a DDoS attack on one of its customers. This approach to block DDoS attacks makes the site in question completely inaccessible to all traffic, both malicious attack traffic and legitimate user traffic. Blackholing is typically deployed by the ISP to protect other customers on its network from the adverse effects of DDoS attacks such as slow network performance and disrupted service. Common attacks are SYN flood and Smurf attack. For Smurf attack a simple solution is to disable IP broadcasting addresses at each network router and firewall. Older routers are likely to enable broadcasting by default, while newer routers will likely already have it disabled. Remediation: Disable broadcasting by default on old routers. To implement RTBH (Remote Triggered Black Hole) Network Operator sets up a null route for the 192.0.2.1 address on all the backbone routers which participate in BGP: ip route 192.0.2.1 255.255.255.255 null 0 254 192.0.2.1 is part of 192.0.2.0/24, the TEST-NET, one of the reserved IPv4 address blocks. Create a route-map to catch routes which need to be blackholed • Static routes can be tagged in Cisco IOS - we will tag routes to be blackholed with the value of 66 • Set origin to be iBGP • Set local-preference to be 150 • higher than any other local-preference set in the backbone • Set community to be no-export and internal marker community (ASN:666) • Don't want prefix to leak outside the AS • Set next-hop to 192.0.2.1 (IPv4) or 100::1 (IPv6) Then introduce the route-map into the BGP configuration. To implement the trigger, simply null	Recommended	No probado

route whatever address or address block needs to be blackholed:
ip route 50.62.124.1 255.255.255.255 null0 tag 66
And this ensures that (for example) 50.62.124.1/32 is announced to the entire backbone with next-hop 192.0.2.1 set.

AV-DoS-ICMP-Flooding	Rate-limit ICMP traffic and prevent the attack from impacting bandwidth and firewall performance	Common attacks are ICMP Flooding, Ping flood and Ping of death - Layer 3 infrastructure DDoS attack methods that use ICMP messages to overload the targeted network's bandwidth. These kind of attacks can be prevented from taking place by rate-limiting ICMP traffic. Remediation: For example, iptables can be used on Linux systems as follows: To protect against ping flood attacks the 'limit' module of iptables can be used: -A INPUT -p icmp --icmp-type echo-request -m limit --limit 60/minute --limit-burst 120 -j ACCEPT -A INPUT -p icmp --icmp-type echo-request -m limit --limit 1/minute --limit-burst 2 -j LOG -A INPUT -p icmp --icmp-type echo-request -j DROP Note that the above lines will protect only against ping request attacks. To protect against generic ICMP flood attacks we can remove the constraint '--icmp-type echo-request'. This is only good enough to protect against PC to PC attacks. If flood is done using multiple sources or using hardware then this configuration may not be enough. The configuration will also cause denial of service to other users when under attack even from single source, as we are limiting based on protocol without considering the source address of the packet. Iptables can also be used to prevent abuse of network resources by rate limiting them: -A OUTPUT -p tcp -m tcp --dport 80 -m limit --limit 4/second --limit-burst 12 -j ACCEPT -A OUTPUT -p tcp -m tcp --dport 80 -m limit --limit 1/minute --limit-burst 1 -j LOG -A OUTPUT -p tcp -m tcp --dport 80 -j DROP In the above example we are limiting outgoing connections to port 80. This is useful when we can't completely block outgoing connections but we do not want it to become channel from where people start browsing net or	Recommended	No probado

downloading. The limit module can also be used in rate limiting incoming traffic for performance reasons. Disabling a ping flood is most easily accomplished by disabling the ICMP functionality of the targeted router, computer or other device. A network administrator can access the administrative interface of the device and disable its ability to send and receive any requests using the ICMP, effectively eliminating both the processing of the request and the Echo Reply. The consequence of this is that all network activities that involve ICMP are disabled, making the device unresponsive to ping requests, traceroute requests, and other network activities.

AV-DoS-L7	Identity which functions of the application are resource consuming and protect and monitor them (optionally set up a WAF)	DoS attacks against the application layer also aim to use up the memory and process time that the application requires to function properly while minimizing the resources employed by the attacker by using vulnerabilities or flaws in the application. The main characteristics of this type of attack are: • The requirement for less bandwidth compared to other methods such as SYN Flood attacks, and as such, the infrastructure necessary to launch a successful attack is usually minor. • Greater difficulty in distinguishing between malicious and legitimate traffic. Many of the most common protection measures or systems focus on the network or transport layer, and as such, they would not be efficient when the target of the attack is the application layer. It is even difficult for the defense systems that monitor this layer to distinguish between malicious and legitimate traffic. As we have indicated, this type of attack is very specific, and generally require fewer resources from the attacker to achieve devastating effects in the systems attacked. However, this type of attack requires more time to analyze the targeted system in order to discover vulnerabilities or bottlenecks that can be exploited during the attack. Remediation • Analyze weaknesses or bottlenecks in the design, the implementation, or even the functioning of the application to identify "heavy" requests that are resource consuming. For example, if the web application has a search engine, an attack could be to carry out searches that require a high level of resources to generate a bottleneck and eventually disable the service. • Implement a challenge to the device making the network request in order to test whether or not it is a bot. This is done through a test much like the CAPTCHA test commonly found when creating an account online. By	Recommended	No probado

		giving a requirement such as a JavaScript computational challenge, many attacks can be mitigated. • Use of a web application firewall, managing and filtering traffic through an IP reputation database, and through on-the-fly network analysis by engineers.		
AV-DoS-TLS	Inspect the TLS/SSL traffic on a capable device before sending the connection down on your infrastructure	TLS adds another vector for computational attacks since a client can easily (with little computational effort) force the server to expend relatively large computational work. If a web application is delivered over TLS, an attacker can also choose to attack the TLS negotiation process. TLS is computationally expensive so an attacker can reduce a server's availability by sending unintelligible data. In a variation of this attack, an attacker completes the TLS handshake but perpetually renegotiates the encryption method. Or an attacker can attempt to exhaust server resources by opening and closing many TLS sessions. Remediation To mitigate consider options like offloading the SSL from the origin infrastructure and inspecting the application traffic for signs of attack traffic or violations of policy at an application delivery platform (ADP). A good ADP will also ensure that your traffic is then re-encrypted and forwarded back to the origin infrastructure with unencrypted content only ever residing in protected memory on a secure bastion host.	Recommended	No probado

CDS-USER-TRACK	Log details of user actions within the system	To maintain proper accountability, logs should be maintained with sufficient information to track user actions within the system. These logs should be forensically sound, non-repudiable, and contain comprehensive details about activity. While the exact data for an event may vary, the following should be captured at a minimum: • Timestamps against a proven external source (e.g. an NTP server) • Origin, with this field we mark if the logs are provided by a trusted or untrusted source. • Event, status, and/or error codes (with sensitive data masked as appropriate or not introduced in logs) • Service, command, application or function name and details • User or system account associated with an event • Devices used (e.g. source and destination IPs, terminal session ID, web browser, etc) Source: https://security.berkeley.edu/security	Recommended	No probado
CSD-ADM-TRUST	Restrict access to administrative interfaces	Restrict access to administrative interfaces to trusted actors from trusted locations to reduce the application attack surface and likelihood of compromise. Restrict administrative access to specific networks or hosts. Use strong authentication for privileged access, for example a 2FA.	Recommended	No probado
CSD-PWD-CHG	Give the user an option to terminate all active sessions when a password change event is triggered	A password change could be the event of a possible password compromise. The user must have the option of invalidating all the active sessions when performing the password change.	Recommended	No probado
CSD-SESS-FED	Provide a section with a list of all active sessions and information about them	The application should provide a section with a list of all the active sessions. That list should contain as much information as possible and should allow the user to invalidate all sessions or any of them selectively.	Recommended	No probado

CSD-SESS-REAUTH	Require additional authentication for sensitive operations / high value transactions	Having gained access to an account, for example through session hijacking or cross-site scripting, an attack may compromise data and functionality accessible to the victim. This may include elevating privileges or accessing sensitive data and functionality, such as changing passwords, creating accounts, or transferring funds. These critical functions and sensitive data should be further protected from attack using a risk-based authentication model requiring re-authentication via multi-factor authentication, or use of a token to sign transactions or operations. • Implement re-authentication on key functions (such as requiring the old password before setting a new one). • Use secondary authentication such as a transaction PIN to allow financial transfers. • Critical administrative functionality should require multi-factor authentication (such as a hard or soft token). • Consider use of signing technologies or tokens that enable the server-side to verify the authenticity of a request and mitigate Man in the Middle (MiTM) or Man in the Browser (MiTB) attacks.	Recommended	No probado

CSD-VAL-LOG	Log and reject all data validation failures	Data validation failures, together with access control violations, are symptomatic of malicious activity where an attacker is attempting to subvert the protections in place. It is therefore likely that unexpected input detected by the application relates to an attack. Rejecting and logging such activity, and ideally terminating the session, increases the likelihood of detecting and inhibiting structured attacks against the application. • Log all validation failures when rejecting requests. • Ensure logged data is appropriately sanitized and encoded to prevent attacks against the logs and subsequent access to them. • Terminate the offending user session to inhibit further attack. • Ensure errors returned to the client-side are generic to prevent an attacker enumerating the defenses in place or gaining knowledge about the back-end.	Recommended	No probado

CWE-147	Validate input parameters to prevent HTTP Parameter Pollution	All data received from external sources should be considered tainted and a potential risk, regardless of the source or transport method. HTTP Parameter Pollution attacks result when parameters are added or inserted into requests, and are inappropriately or unsafely handled by the server. For example, injection of parameters into strings or cookies sent to the application can result in the variables being interpreted out of sequence, exploiting flaws in the business logic and flow. For example, there was a bug in the Blogger platform which illustrates this vulnerability. The bug allowed malicious users to take ownership of the victim's blog by using the following HTTP request: POST /add-authors.do HTTP/1.1 security_token=attackertoken&**blogID=attackerblogidvalue**&**blogID=victim blogidvalue**&authorsList=goldshlager19test%40gmail.com(attacker email)&ok=Invite The flaw resided in the authentication mechanism used by the web application, as the security check was performed on the first blogID parameter, whereas the actual operation used the second occurrence.	Recommended	No probado

CWE-178-AUTH	The login function should distinguish between upper and lower case passwords	Passwords used either as sole verification credentials, or as part of a multi-factor authentication, are a key aspect of application security, and use of strong passwords is critical. In addition to password quality criteria, it is important that the verification of passwords by the application does not reduce the character space, and therefore quality. • The login function should distinguish between upper and lowercase passwords; for example PASSWORD should be considered different to password, Password, PASSword etc. • Reducing the character set by considering an uppercase or a lowercase as equivalent also reduces the Computational Complexity. • Reduction in character space reduces computational cost of brute-force or dictionary attacks against the credentials.	Recommended	No probado

CWE-204-USERNAME-LOGIN	Ensure application errors do not reveal account status	Error messages can reveal information of use to an attacker, particularly in the cases relating to login, registration and recovery processes. Explicit errors stating accounts are valid or not, or whether they are locked out, are useful to an attacker in enumerating usernames for subsequent attacks. Even small variations in errors returned to the user can be used to infer how the application operates, or enforces security controls. To prevent a malicious user using error conditions to determine how the application functions in order to subvert it: • Trap and log all known error conditions on the server-side • Create a generic trap for unexpected errors • E.g. Error messages during the login process should not disclose which authentication token was incorrect, instead a generic message simply stating that authentication failed should be displayed. • Ensure the error message returned to the user is generic, or contains only an obfuscated reference that can be correlated with the logged errors on the server-side. In the case of applications delivered over HTTP, the HTTP header should also be generic to prevent enumeration (e.g. all error pages should produce a consistent HTTP 500 error)	Recommended	No probado

CWE-226	Overwrite data in memory before release	Memory chunks released by an application are not actively overwritten, they are simply de-referenced with the data left until the memory is reallocated and used by another process. As such, sensitive data that is stored in memory may be exposed to an attacker with the ability to inspect that memory; for example through use of an uninitialized variable or other process. Overwrite memory with zeros or random data before release. Pay particular attention to sensitive data, such as passwords or other credentials, or sensitive personal information.	Recommended	No probado
CWE-255	Remove default credentials and role-based accounts from the application	Security is often compromised through default or predictable account credentials, such as 'admin/admin'. Best-practice dictates that accounts are only enabled when required, do not have common account names, and force users to choose unique strong passwords rather than using vendor defaults. • Ensure all default application and software accounts are disabled or removed if not required • Strong passwords should be set on accounts that are required, default credentials must be changed. • Build application accounts from a least-privilege perspective. • Accounts should only be enabled if required. • Users should have individual accounts rather than role-based ones (e.g. dave-admin, sue-admin rather than a shared 'admin' user)	Recommended	No probado

CWE-285	Apply authorization checks to segregate and control access to user data	Applications protecting sensitive or otherwise restricted resources must ensure only appropriate and authorized users may access the application data. It is important that an application prevents unauthorized users gaining inappropriate access to each other's data; although user A and user B may both be trusted to access data within the application, they may be authorized to only access different subsets of the protected resources. E.g. user A should not be able to access user B's personal data by manipulating a request (typical examples are manipulation of an ID value or other object reference sent in the URL or body of an HTTP request). It is not sufficient to rely on obscurity, for example obfuscated or secret URLs or filenames. The application must validate each request for protected data against the proven identity of the user. Before providing access to restricted resources the application must: • Ensure the user has undergone appropriate authentication (identification and verification, or ID&V). E.g. they must have provided their identity and confirmed this with a password, token, or other verification. Typically this will be through checking the validity of the session token issued after login. • Confirm the user is entitled to access the data or resource they are requesting. E.g. their confirmed identify checked against a server-side access control matrix to determine whether they may access the requested resource. • Access controls should be granular, and allow for access to individual resources to be issued to individual users or roles. URL and asset based access control is provided by most web-frameworks, and it is preferable to use an established and proven framework.	Recommended	No probado

CWE-306-AUTH	Ensure the application uses a single vetted authentication mechanism that is known to be secure	• Ensure that the application uses a single vetted authentication mechanism that is known to be secure, can be extended to include strong authentication, and has sufficient logging and monitoring to detect account abuse or breaches. • Ensure that all authentication pathways and identity management APIs implement consistent authentication security control strength, such that there are no weaker alternatives per the risk of the application.	Recommended	No probado
CWE-311-AT-REST	Encrypt data stored on the host (data at rest)	Data stored on the server or the client must be protected by encryption (data *at rest*). • Cryptographically strong symmetric or asymmetric (public-key) encryption should be used to protect the data. • Encryption should be performed before the data is written to disk or other persistent storage. • The key for encrypting and decrypting the data should *not* be accessible from the same host. • The encryption and decryption operation should be performed on a different host. • A recognized, proven, and tested implementation/library should be used (in preference to a bespoke implementation).	Recommended	No probado

CWE-367-TOCTOU	Ensure that all high-value business logic flows, including authentication, session management and access control are thread safe and resistant to time-of-check and time-of-use race conditions.	• Do not perform a check before the use. This does not resolve the underlying issue of the execution of a function on a resource whose state and identity cannot be assured, but simply gives a false sense of security given by the check. • When the file being altered is owned by the current user and group, set the effective gid and uid to that of the current user and group when executing this statement. • Limit the interleaving of operations on files from multiple processes. • If you cannot perform operations atomically and you must share access to the resource between multiple processes or threads, then try to limit the amount of time (CPU cycles) between the check and use of the resource. This will not fix the problem, but it could make it more difficult for an attack to succeed. • Recheck the resource after the use call to verify that the action was taken appropriately. • Ensure that some environmental locking mechanism can be used to protect resources effectively. Ensure that locking occurs before the check, as opposed to afterwards, such that the resource, as checked, is the same as it is when in use.	Recommended	No probado

CWE-384	Actively reject Session IDs that have not been generated by the application	The application should recognize previously issued Session IDs and reject every session ID that has not been generated by the application. Some types of attacks rely on session fixation. This involves an attacker making up a Session ID not generated by the application and then deceiving the user into clicking a link and authenticating using this Session ID. As the attacker has knowledge of this Session ID he/she will have access if that Session ID is accepted and not invalidated and changed after authentication.	Recommended	No probado

CWE-485-PREPROD	Ensure personal and other sensitive data is not exposed in pre-production environments	Pre-production environments should not expose personally identifiable information (PII) or other sensitive information. Often, such environments are populated with production data for testing or other purposes. Security weaknesses, either in the software or controls around access to pre-production systems, can unnecessarily expose sensitive data. This is often a breach of legal and commercial requirements, including various governmental data protection standards and privacy acts, and industry regulations such as those required by the Payments Card Industry (PCI). Where test data is derived from production data, the following must be observed: 1. Data masking or obfuscation techniques must be leveraged to anonymize the data used in pre-production. For example, all but the start and end of a credit card number must be removed/replaced 2. No real personally identifiable information (PII) or other sensitive data should be present in pre-production environments. For example, names, dates of birth, and other personal information must be removed, randomized, or replaced. 3. Only data that has been processed and anonymized should be transferred into pre-production; ideally representative test data should be generated rather than relying on obfuscated production data.	Recommended	No probado

CWE-524	Clear caches of sensitive data	Applications storing or caching data locally are at risk of attack and compromise of that data. Local temporary storage, thumbnail images, and configuration files often contain excerpts of sensitive information that may be left on the device during or after the application is used. • Ensure all temporary files, caches, and storage are purged after use and when the application is closed. • Do not allow the application to cache sensitive information outside of the OS provided secure stores. • Use OS controls to restrict thumbnails of the application in cases which sensitive information is displayed	Recommended	No probado
CWE-532	Do not write secrets to the log files	The logs may be accessed by attackers and in order to protect sensitive data, no such sensitive data should be included in the logs	Recommended	No probado
CWE-541	Prevent unauthorised access to source code through the service	Access to the source-code for the application can aid an attacker in determining bugs or vulnerabilities in the code or logic. For closed-source projects it is therefore important to control and restrict access to the source. Application services may unexpectedly expose code, for example a service providing files to a user could be manipulated to access source code if implemented insecurely. • Ensure source code is not inadvertently disclosed through misconfiguration or vulnerabilities in the service. • Check configuration files are not downloadable directly from the service, and cannot be read and viewed through the service itself. • Ensure backups, operating systems, and version control artifacts do not expose code.	Recommended	No probado

CWE-598	Ensure no sensitive data is sent in the URL	Data passed in the URL (address bar) between client and server is likely to be exposed in logs on the server, or intermediate devices (such as proxies or other network devices), or in local browser logs/history. A common mistake is, for example, to pass a SessionID value as a parameter in the URL of a get request between hosts. To mitigate the risk of exposure of sensitive data, this sensitive data should only be sent in the body of an HTTP message (for example a POST request), or in the HTTP headers (i.e. standard headers such as the Cookie values, or a custom X- header). • Ensure sensitive, private, or otherwise restricted data is not sent in a URL value. • Pass sensitive data only in appropriately protected fields in the body or headers.	Recommended	No probado
CWE-601	Whitelist which URLs the application may redirect to	The application should maintain a list of URLs to which it may redirect users. This whitelisting can also be achieved by applying regular expressions. The application should only allow redirects to trusted URLS.	Recommended	No probado

CWE-603	Enforce authentication on the server-side	All data on the client-side must be considered tainted. As such, decisions on authentication (or Identification and Verification - ID&V) must be made or validated on the server-side to prevent their subversion. • Ensure credentials are passed securely from client-to server. • The server should compare the credentials with those stored on the server-side (e.g. the hashed password compared against those stored for the claimed identity). • Implement controls to mitigate brute-force attacks; for example through rate-limiting, account lockouts, or escalating timeouts. • Once authenticated, a non-predictable and cryptographically secure token should be passed securely to the client-side to validate further interaction with the server. • Messages returned to the user during authentication, in particular when the process fails, should not reveal to the user whether the username was valid. Returning generic errors prevents an attacker enumerating valid account IDs for subsequent attacks.	Recommended	No probado

CWE-639	Avoid using direct references to files	All data received from the client-side should be considered tainted and a potential risk, regardless of the source or transport method. Many flaws in applications result from unsafe handling of filenames or path data, for example directory traversal and arbitrary file disclosure. Use of filenames or other direct references to objects such as files on the server-side increases the risk of compromise of the system or unauthorized access to other files, content, or functionality. • Avoid using direct references to files or validate file name. • For example, instead of using: www.example.com/open.jsp?file=details.txt use: www.example.com/open.jsp?file=[GUID] and then use a lookup table on the server to associate the [GUID] value with file details.txt. • Files should be stored outside the web-root to prevent an attacker browsing directly to them. • Do not use obscurity to prevent access; for example do not rely on renaming a file to include a random string to prevent direct download or disclosure. If direct references are used to access files from within the web-root deviating from recommended security practice, the following must be implemented: • String input validation must be applied. Identify and filter for directory traversal meta-characters such as "../" or "..\" or "/", together with encoded variations and derivations. • Use of a strict white-list of files that may be downloaded.	Recommended	No probado
CWE-646-CSP	Ensure that user-uploaded files are served by either octet stream downloads, or from an unrelated domain	Ensure that user-uploaded files - if required to be displayed or downloaded from the application - are served by either octet stream downloads, or from an unrelated domain, such as a cloud file storage bucket. Implement a suitable content security policy to reduce the risk from XSS vectors or other attacks from the uploaded file.	Recommended	No probado

CWE-654	Authenticate users (multi - or two-factor - authentication [2FA])	The application should ensure users have undergone an Identification and Verification (ID&V) process before allowing access to secret, sensitive or otherwise restricted data. For more sensitive applications hosting data or functionality requiring greater assurance over the identification of users, a multi - or two-factor - authentication process is recommended. Together with asserting their identity and a password/phrase/PIN (something you know), such systems require additional factors, such as biometric validation (something you are), user-initiated action such as a button press on a FIDO hardware key or a token (something you have). • Implement multi-factor authentication for applications providing access to sensitive data or functionality, and for highly privileged user access (e.g. administrators). • This should leverage something such as a soft or hard-token verification in addition to the secret password/pass-phrase for example.	Recommended	No probado

CWE-662	Use a synchronised time source	In order to correlate logs and data from different internal and external systems, and to preserve forensic quality of the logs, it is important a unified and trusted synchronized time source is used throughout the environment. • Servers should synchronize to an internal or external NTP server • The centralized source should in turn use (or be) a trusted central time source. This control is critical in identifying application events (including attacks) through logging, and in conducting post-event analysis, and in particular to track the entire user (or attacker) journey through the system should it be compromised. It is good practice to use the concept of Indicators of Compromise (IoC) which should be leveraged to detect possible situations in which the system has been compromised and give an appropriate response. IoCs are often tracked through logs, and accurate time is essential.	Recommended	No probado
CWE-693-STRICT-TRANSPORT-SECURITY	Set the HTTP security header 'Strict-Transport-Security' (HSTS)	Modern browsers support the Strict-Transport-Security HTTP header. This header instructs the browser to only communicate over HTTPS for all subsequent requests to the domain, mitigating a number of attacks designed to force users to switch to unencrypted transport in order to expose data. • Set the Strict-Transport-Security HTTP header in all server-responses.	Recommended	No probado
CWE-923-SEGREG	Ensure segregation of components of differing trust levels	Ensure segregation of components of differing trust levels through well-defined security controls, firewall rules, API gateways, reverse proxies, cloud-based security groups, or similar mechanisms.	Recommended	No probado

DATA-VAL	Validate all data received from the client side	All data received from the client-side should be considered tainted and a potential risk, regardless of the source or transport method. For example, while hidden form fields, cookies, or other headers may be obfuscated from a user, as well as parameters passed in ViewStates or other encapsulated forms, these could be modified by the user on the client-side in memory, or in transit on the network. Similarly, data passed from binary or compiled components could also be modified in situ or in transit. *Furthermore, encryption only secures the data in transit between the two ends of the encrypted tunnel (one end of which is typically controlled by the client); data passing through the link may still be malicious.* As such, all data from the client-side must be subjected to strict validation, sanitization, and encoding against expected syntactic and semantic criteria. • Define a specification for the data that is expected at each input; both the syntax (e.g. alphanumeric only) and semantics (e.g. a word of between 1 and 25 characters, or a specific list). As an example of a business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." • Implement a 'known good' or white-list approach, where only inputs that meet the strict criteria for each input are accepted, and reject, transform, or encapsulate any non-compliant data. • While useful for identifying malicious content, do not rely on looking for specific malformed or attack payloads (blacklists). It is almost impossible to maintain a comprehensive and accurate blacklist due to the complexity and evolving nature of attacks, opportunities to obfuscate payloads, and changes to	Recommended	No probado

		the code's execution environment. Having said that, blacklists may be useful for detecting and logging potential attacks, or determining which inputs are so malformed that they should be rejected outright. • Validate all data received from the client, including values such as HTTP headers and cookie values if these are used as input on the server side, X-headers, and other platform specific data objects passed between the client and server.		
DIRECTORY_LISTING	Disable web server directory listing and ensure file metadata (e.g. .git) and backup files are not present within web roots	Indexable directories allow an attacker to easily discover the existence of content on your web server that should remain private. Remediation: 1. Disable directory listings in the web - or application - server configuration by default. 2. Restrict access to unnecessary directories and files. 3. Create an index (default) file for each directory.	Recommended	No probado

ENV-USE	Ensure there are no environmental security weaknesses	Application security can be undermined by misconfiguration of the underlying server or development environment. Ensure that vendor and best-practice guidelines have been applied, in particular affecting areas such as: • Cryptographic strength of elements such as session tokens, GUIDs, and protection of data in transit and at rest. • Removal of debugging and compiler options and artifacts used during development. • Configuration of error trapping and use of generic error messages and pages (e.g. stack traces should not be returned to users). • Appropriate access controls over software components, variables (e.g. EJBs and methods). • Insecure or inappropriate storage of data in configuration files or other data structures (e.g. passwords in configuration files). • Use of compilation/build features or security enhancements provided by the IDE	Recommended	No probado
EU-GDPR-BACKUP	Implement a Backup and Recovery process	Ensure backup policy is active and tested.The policy should describe the required recovery time objective (RTO) and recovery point objective (RPO) so that the availability of personal data can be restored in a timely manner (based upon the requirements specified by the DPO/CISO). Ensure an SLA has been defined for data availability. How 'timely manner' will be interpreted depends on your SLA.	Recommended	No probado

EU-GDPR-IMPLEMENT-FORGET	Implement forget user functionality	Make sure that your deletion functionality also facilitates the manual deletion of one data subject (person of whom personal data is processed) at any given point in time. All data of the data subject should be deleted from the operational databases as well as back-ups. **Remediation:** You should develop a script to ensure the re-deletion of data subjects (persons of whom personal data is processed) who exercised their right to be forgotten and run it after each database restore. The data subjects should be deleted from the script whenever the retention period of the data has expired.	Recommended	No probado
EU-GDPR-LOGGING	Logging the data access and the modification of personal data	Make sure that every access and modification of personal data is logged (The log should be precise enough to know who changed what and when). This includes data access from the application as well as data access from privileged accounts, such as database admins. In practice, this means logging both on application, middleware and operating system (OS) level.	Recommended	No probado
EU-GDPR-PERMANTLY-DELETE-UNCESSESSARY-DATA	Permanently delete unnecessary data	Ensure that personal data is automatically deleted, within operational and back-up environments, whenever the defined retention periods have expired. This requires automatic identification of latest activities within the database (timestamping logging) and a data deletion functionality.	Recommended	No probado
FEAT-ACC-CTRL	Ensure that attribute or feature-based access control is used	Ensure that attribute or feature-based access control is used whereby the code checks the user's authorization for a feature/data item rather than just their role. Permissions should still be allocated using roles.	Recommended	No probado

FILE-DATA-CHECK	Validate the file format before processing	Attackers may attempt to bypass controls on uploaded content by renaming, embedding, or otherwise manipulating benign files with malicious content. As with client-side data, the application should screen uploaded content before processing it. • File contents and file-system path data must to be checked for sanity. • Consider directory traversal, local file inclusion, and OS injection attacks when validating filenames. • Verify MIME type is correct and appropriate, and matches content. • Files with an structured format should be checked against the expected format/definition (e.g. XML against a XSD or DTD).	Recommended	No probado
FILE-SCAN	Perform content scanning on untrusted files when uploaded	The application should perform validation and sanitization on all tainted content from the client-side, including uploaded files. • Content/malware scanning performed against the file (e.g. next generation anti-virus) to detect suspicious, dangerous, or unexpected content. • Content analysis should also be used to reject unexpected or dangerous file formats that may harbor malicious code, for example encrypted word, pdf or other office documents or archives, or other files with hidden or executable payloads.	Recommended	No probado
LEAST-PRIV-ENF	Ensure enforcement of the principle of least privilege	Ensure enforcement of the principle of least privilege in functions, data files, URLs, controllers, services, and other resources. This implies protection against spoofing and elevation of privilege.	Recommended	No probado
LIMIT-ACTIVE-SESSIONS	Limit the number of active concurrent sessions	The number of active sessions shall be limited to avoid several active sessions at the same time. If there are several active sessions, one of them may be used by the real user and another session by attackers who are able to implement changes without the real user being aware.	Recommended	No probado

LOG-RETENTION	Develop a log retention policy	Develop a log retention policy to identify storage requirements for device logs and implement procedures to ensure that the audit logs are available for a security response in the case of incident or investigation. The audit logs must be collected for the last 30 days in easily accessible storage media. Older logs should be archived in a protected storage and should be accessible in the future as required for incidents or investigations.	Recommended	No probado
LOG-TLS-FAILURES	Log the backend TLS connection failures	Implement functionality to record backend TLS connection failures and include these in the logs.	Recommended	No probado
LOGS-INTEGRITY	Ensure the integrity of the logging system	Ensure Log integrity for the application generated logs, such as storing logs on write-once media, forwarding a copy of the logs to a centralized SIEM or generating message digests for each log file.	Recommended	No probado
OTG-BUSLOGIC	Detect and notify the usage of automated tools or unusual behavior	Don't allow users to manipulate a system or guess its behavior based on input or output timing and detect the usage of automated tools or unusual behavior, such as actions not performed in reasonable "human time" or other abnormal time patterns. When the usage of automated tools is detected, the application should respond by denying access and notifying the security group.	Recommended	No probado
OTG-BUSLOGIC-006	Restrict actions of users that follow unusual patterns.	Restrict actions that users can do outside of the approved/required business process flow. This is important because without this safeguard in place attackers may be able to bypass or circumvent work-flows and checks allowing them to prematurely enter or skip required sections of the application potentially allowing actions/transactions to be completed without successfully completing the entire business process, leaving the system with incomplete back-end tracking information.	Recommended	No probado

PROPER-REVOCATION-CERTIFICATE	Configure and enable appropriate certification revocation	Configure and enable appropriate certification revocation for each certificate created, such as Online Certificate Status Protocol (OCSP) Stapling. OCSP is a protocol to check if an SSL certificate has been revoked. Instead of the client downloading a large list of revoked certificates, they can simply submit a request to a CA server, which returns a signed response with the certificate current status.	Recommended	No probado
RENEW-SESSION	Implement mechanisms to renew the session and session id	Implement a mechanism to generate a new session and new session id, when a successful authentication or re-authentication occurs. After this creation, the old session and session ID shall be removed securely.	Recommended	No probado

RESTRICT-HTTP-METHODS	Ensure that the application accepts only a defined set of required HTTP request methods	HTTP offers a number of methods that can be used to perform actions on the web server. Many of theses methods are designed to aid developers in deploying and testing HTTP applications. These HTTP methods can be used for nefarious purposes if the web server is misconfigured. For example, Cross Site Tracing (XST), a form of cross site scripting using the server's HTTP TRACE method. While GET and POST are by far the most common methods that are used to access information provided by a web server, the Hypertext Transfer Protocol (HTTP) allows several other (and somewhat less known) methods. RFC 2616 (which describes HTTP version 1.1 which is the standard today) defines the following eight methods: • HEAD • GET • POST • PUT • DELETE • TRACE • OPTIONS • CONNECT Some of these methods can potentially pose a security risk for a web application, as they allow an attacker to modify the files stored on the web server and, in some scenarios, steal the credentials of legitimate users. More specifically, the methods that should be disabled are the following: • PUT: This method allows a client to upload new files onto the web server. An attacker may exploit this by uploading malicious files (e.g.: an asp file that executes commands by invoking cmd.exe), or by simply using the victim's server as a file repository. • DELETE: This method allows a client to delete a file on the web server. An attacker may exploit it as a very simple and direct way to deface a web site or to mount a DoS attack. • CONNECT: This method could allow a client to use the web server as	Recommended	No probado

		a proxy. • TRACE: This method simply echoes back to the client whatever string has been sent to the server, and is used mainly for debugging purposes. This method, originally assumed harmless, can be used to mount an attack known as Cross Site Tracing, which was discovered by Jeremiah Grossman (see links at the bottom of the page). If an application needs one or more of these methods, such as REST Web Services (which may require PUT or DELETE), it is important to check their usage is properly limited to trusted users and safe conditions. Remediation: Ensure the application accepts only the HTTP requests GET and POST. The HTTP requests TRACE, PUT and DELETE are blocked.		
RESTRICT-NUMBER-ACCOUNT-TO-LOGS	Limit the number of accounts with privileges allowing modification and/or deletion of audit logs files	Limit the number of accounts with privileges to modify and/or delete audit logs files.	Recommended	No probado
RFC6819-4.2.1-C1	Provide mechanisms to confirm server authenticity	Authorization servers should attempt to educate users about the risks posed by phishing attacks and should provide mechanisms that make it easy for users to confirm the authenticity of their sites.	Recommended	No probado
RFC6819-4.4.3.1-C1	Avoid using "resource owner password credentials" and "implicit" flows	Considerations to migrate to OAuth2.1: Use other flows that do not rely on the client's cooperation for resource owner interaction. Moreover, in OAuth2.1 the implicit and resource owner password flows have been removed and they can't be used anymore.	Recommended	No probado

RFC6819-5.1.5.2	Determine expiration time	Tokens should generally expire after a reasonable duration. This complements and strengthens other security measures (such as signatures) and reduces the impact of all kinds of token leaks. Depending on the risk associated with token leakage, tokens may expire after a few minutes (e.g., for payment transactions) or stay valid for hours (e.g., read access to contacts). The expiration time is determined by several factors, including: • risk associated with token leakage, • duration of the underlying access grant, • duration until the modification of an access grant should take effect, and • time required for an attacker to guess or produce a valid token.	Recommended	No probado
RFC6819-5.1.5.3	Use short expiration time	A short expiration time for tokens is a means of protection against the following threats: • replay • token leak (a short expiration time will reduce impact) • online guessing (a short expiration time will reduce the likelihood of success) Note: Short token duration requires more precise clock synchronization between the authorization server and resource server. Furthermore, shorter duration may require more token refreshes (access token) or repeated end-user authorization processes (authorization "code" and refresh token).	Recommended	No probado
RFC6819-5.1.5.9	Sign self-contained tokens	Self-contained tokens should be signed in order to detect any attempt to modify or produce faked tokens (e.g., Hash-based Message Authentication Code or digital signatures).	Recommended	No probado

SAME-ENCODING-STYLE	Ensure that the client-side and the server-side are using the same encoding style	Ensure that the client-side and the server-side are using the same encoding style.	Recommended	No probado
SEC-DEPLOY	Ensure the build pipeline contains a build step to automatically build and verify the secure deployment of the application	Ensure that the build pipeline contains a build step to automatically build and verify the secure deployment of the application, particularly if the application infrastructure is software defined, such as cloud environment build scripts.	Recommended	No probado
SEC-FEATURES	Verify the application correctly implements the security features needed according to its security level	The security controls in place will in part be defined by the nature of the application. A low security unauthenticated application will have different requirements to an application providing transactional capabilities in a finance system, for example. • Implement features around security like authentication, access control, confidentiality, cryptography, and privilege management based on the principle of 'fit for purpose' (appropriate to the application).	Recommended	No probado
SER-UNTRUST	Ensure that serialization is not used when communicating with untrusted clients	Ensure that serialization is not used when communicating with untrusted clients. If this is not possible, ensure that adequate integrity controls (and possibly encryption if sensitive data is sent) are enforced to prevent deserialization attacks including object injection.	Recommended	No probado

SESS-SNIFF-COOKIE	Set the 'secure' flag (or directive) on sensitive cookies	Cookies and their data are passed in the HTTP headers of client-web server communications, and are subject to the same transport security. Sensitive data such as the Session ID sent over unencrypted HTTP transport can be intercepted or monitored in transit, exposing the information or the user's session. The secure directive instructs compliant browsers to only pass the cookie over secure transport, which in most instances implies it will only be sent over an encrypted HTTPS (TLS) connection, thereby protecting users from network sniffing and spoofing attacks. • Set the secure directive on all cookies containing sensitive data (in particular the session ID token). • Where an application uses a mix of encrypted and unencrypted content, if session tracking is required a secondary session token which does not permit access to sensitive content could be used, although this deviates from security best-practice.	Recommended	No probado
SRV-JSON-WEB-SIGN	Implement JSON Web Signing for sensitive or authentication data	Data received from the client-side may have been modified at source or in transit. JSON Web Signing functionality is available to prevent (or rather detect) such modification. • Sensitive data, in particular any relied on for making access control or other security decisions, should be signed. This provides assurance that: • The data comes from the trusted source and not from an untrusted actor. • The data received is that produced by the trusted party and has not been modified in transit.	Recommended	No probado

STRONG-ALGORITHMS-PROTOCOLS	Ensure that only strong algorithms, cyphers and protocols are used	Ensure that the application is using strong algorithms, cyphers and protocols. The general practice and required minimum key lengths depend on the scenarios listed below. • Key exchange: Diffie-Hellman key exchange with minimum 2048 bits • Message Integrity: HMAC-SHA2 • Message Hash: SHA2 256 bits • Asymmetric encryption: RSA 2048 bits • Symmetric-key algorithm: AES 128 bits • Password Hashing: Argon2, PBKDF2, Scrypt, Bcrypt.	Recommended	No probado
TLS-STRONG-CIPHERS	Require cryptographically strong TLS cipher suites	Only cryptographically strong ciphers should be required. Best-practice dictates a subset of ' *known good*' ciphers and protocols be defined and enforced on the server. This may, however, have compatibility issues with older browsers, requiring a balance be sought between accessibility and security. • Define and enforce a list of acceptable ciphers and protocols • Explicitly disable known-bad ciphers and protocols, such as: • Null and export ciphers • DH, MD5 and other weak cryptography • Ciphers with keys less than 128 bits • CBC ciphers with TLSv1.0 or earlier	Recommended	No probado

TLS-STRONG-PROTOCOLS	Require cryptographically secure protocols (e.g. TLSv1.2 and above)	Only cryptographically strong ciphers should be required. Best-practice dictates a subset of 'known good' ciphers and protocols be defined and enforced on the server. This may, however, have compatibility issues with older browsers, requiring a balance be sought between accessibility and security. • Define and enforce a list of acceptable ciphers and protocols. Disable SSLv3 and earlier protocols on the service. • Ideally only TLSv1.2 and newer should be supported • If TLSv1.1 or 1.0 are required, known secure configurations and ciphers should be selected. • SSLv3.0 and earlier should not be used	Recommended	No probado
WEB-EXEC-DATA	Do not execute files received from untrusted sources	Untrusted file data sources could contain malicious software, therefore, data sources should not be executed even if an anti-malware scanner has been run against the file. This includes use of pre-processing that may result in execution (e.g. eval of tainted content).	Recommended	No probado
WEB-FILE-DATA	Do not use untrusted client-side data in server-side file operations	The application may be required to perform server-side file operations based on client-side actions. This can expose the application to attack through remote or local file-inclusion, or remote code execution attacks if unsafe data is used in file inclusion, class loader, or reflection capabilities. • Tainted, untrusted, client-side data must not be used directly in server-side file load operations. • Where server-side operations are dependent on client-side input, this should be mapped to pre-defined operations on the server-side rather than using tainted client-side input directly.	Recommended	No probado

WEB-SRV-ADM-AUTH	Restrict access to administrative functionality	If inadequate controls are in place, lower privileged users may be able to access higher privilege or administrative functionality to subvert security within the application. • Restrict administration functions to designated administrators only through robust access controls. • Ensure this restriction is applied at the server-side; do not rely on 'secret' areas of the application, menu hiding, or other client-side techniques to protect the functionality. • Measures to prevent cross-site request forgery must be present on administrative functions.	Recommended	No probado
WEB-SRV-XML	Define and enforce secure validation through an XSD or DSD schema on XML input data	The XML standards provides for formal validation criteria to be specified in a schema (XSD or DTD). Defining a schema with security in mind, and enforcing it on input data, will mitigate many attacks. • Define an XSD or DTD schema with secure input validation criteria. • Validate all XML input data against the schema before processing it. • The acceptance of unvalidated or non compliant data could have unexpected impact on the behavior of the application, and may facilitate attacks such as SQL injection.	Recommended	No probado
configuration-integrity	Ensure the integrity of all security-relevant configurations to detect tampering	Web server and application server configurations play a key role in the security of a web application. These servers are responsible for serving content and invoking applications that generate content. In addition, many application servers provide a number of services that web applications can use, including data storage, directory services, mail, messaging, and more. Failure to manage the proper configuration of your servers can lead to a wide variety of security problems. **Remediation:** Authorized administrators need to ensure the integrity of all security-relevant configurations to detect tampering.	Recommended	No probado

cwe-352-csrf	Protect cookies from CSRF	Cross-Site Request Forgery (CSRF) is a type of attack that occurs when a malicious web site, email, blog, instant message, or program causes a user's web browser to perform an unwanted action on a trusted site when the user is authenticated. A CSRF attack works because browser requests automatically include any credentials associated with the site, such as the user's session cookie, IP address, etc. Therefore, if the user is authenticated to the site, the site cannot distinguish between the forged or legitimate request sent by the victim. A token/identifier that is not accessible to the attacker is needed to mitigate this and would not be sent along (like cookies) with forged requests that the attacker initiates. Remediation: Requiring re-authentication of sensitive functions is an effective way to protect your application against a CSRF, but intrusive to the user experience. You can also protect cookies from CSRF using at least one or more of the following recommendations: 1. CSRF nonces. Use nonces to prevent unauthorized access by providing a secret 'key' (which is valid only once) that must be validated in any sensitive form submission. For maximum security, the nonce is also time sensitive and expires. 2. Triple or double submit cookie pattern. If maintaining the state for the CSRF token on the server-side is problematic, an alternative defense is to use the double-submit cookie technique. This technique is easy to implement and is stateless. In this technique, a random value is sent in both a cookie and as a request parameter, with the server verifying if the cookie value and request value match. When a user visits (even before authenticating to prevent login CSRF), the site should generate a (cryptographically strong) pseudo random value and set it as a cookie on the user's machine separate from the	Recommended	No probado

session identifier. The site then requires that every transaction request include this pseudo-random value as a hidden form value (or other request parameter/header). If both of them match on the server-side, the server accepts it as a legitimate request and if they don't, then rejects the request.

3. ORIGIN request header checks.There are two steps to this mitigation, both of which rely on examining an HTTP request header value. Firstly, determining the origin the request is coming from (source origin). This can be achieved via Origin and/or referer headers. Secondly, determining the origin the request is going to (target origin). On the server-side verify if both of these match. If they do, we accept the request as legitimate (meaning it's the same origin request) and if they don't, discard the request (meaning that the request originated from a cross-domain). Reliability of these headers comes from the fact they cannot be altered programmatically (using JavaScript in an XSS) as they fall under the forbidden headers list (i.e., only browsers can set them).

deny-default-enf	Assure that the principle of deny by default exists for new users/roles	Deny by default is the principle that if a request is not specifically allowed, it is denied. There are many ways this rule will manifest in application code. Some examples of these are: 1. Assure the principle of deny by default exists whereby new users/roles start with minimal or no permissions and users/roles do not receive access to new features until access is explicitly assigned. 2. Application code may throw an error or exception while processing access control requests. In these cases access control should always be denied. 3. When an administrator creates a new user or a user registers for a new account, that account should have minimal or no access by default until that access is configured. 4. When a new feature is added to an application all users should be denied use of that feature until it's properly configured.	Recommended	No probado
dyn-exec	Assure that the application avoids the use of eval() or other dynamic code execution features	Assure that the application avoids the use of eval() or other dynamic code execution features. Where there is no alternative, any user input being included must be sanitized or sandboxed before being executed.	Recommended	No probado

error-handling-centralized	Implement centralized mechanisms to manage errors and exceptions	Create functionality or a mechanism to manage exceptions in a centralized manner and avoid including duplicated try/catch blocks in the code. A well-planned error/exception handling strategy is important for three reasons: • Good error handling does not give an attacker any information which can be leveraged for attacking the application • A proper centralized error strategy is easier to maintain and reduces the chance of any uncaught errors "Bubbling up" to the front-end of an application. • Information leakage can lead to social engineering exploits Remediation: Building an infrastructure for consistent error reporting proves more difficult than error handling. All exceptions should be caught as events with the severity of the error. These events will be logged. The centralized system should reorganize all events and throw the corresponding error or system message in a consistent manner, depending on the severity of the error.	Recommended	No probado

follow-jwt-standard-generation-token	Implement token generation for stateless server following the recommendation of the JWT standard	JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed. A JWT token is created during authentication and is verified by the server (or servers) before any processing. However, JWT's are often not saved by the server after initial creation. JWT's are typically created and then handed to a client without being saved by the server in any way. The integrity of the token is maintained through the use of digital signatures so a server can later verify that the JWT is still valid and not been tampered with since its creation. This approach is both stateless and portable in the way that client and server technologies can be different yet still interact. Ensure implementation of the following requirements to protect the stateless token for digital signatures, encryption and other countermeasures following the requirements of JWT standard. Remediation: To create a JWT, the following steps are performed. The order of the steps is not significant in cases where there are no dependencies between the inputs and outputs of the steps. 1. Create a JWT Claims Set containing the desired claims. Note that whitespace is explicitly allowed in the representation and no canonicalization need be performed before encoding. 2. Let the Message be the octets of the UTF-8 representation of the JWT Claims Set. 3. Create a JOSE Header containing the desired set of Header Parameters. The JWT MUST conform to either the [JWS] or [JWE] specifications. Note that whitespace is explicitly allowed in the representation and no canonicalization need be performed before encoding. 4. Depending upon whether the JWT	Recommended	No probado

is a JWS or JWE, there are two cases:

- If the JWT is a JWS, create a JWS using the Message as the JWS Payload; all steps specified in [JWS] for creating a JWS MUST be followed.
- Else, if the JWT is a JWE, create a JWE using the Message as the plaintext for the JWE; all steps specified in [JWE] for creating a JWE MUST be followed.

5. If a nested signing or encryption operation will be performed, let the Message be the JWS or JWE, and return to Step 3, using a "cty" (content type) value of "JWT" in the new JOSE Header created in that step.
6. Otherwise, let the resulting JWT be the JWS or JWE.

harden-http-headers	Harden HTTP Headers	HTTP headers are a fundamental part of website security. • The Referrer-Policy HTTP header controls how much referrer information (sent via the Referrer header) should be included with requests. This security header was designed to prevent cross-domain Referrer leakage. • The Origin request header indicates where a fetch originates from. It doesn't include any path information, but only the server name. It is sent with CORS requests, as well as with POST requests. It is similar to the Referer header, but, unlike this header, it doesn't disclose the whole path. • The Content-Disposition response header tells the browser to download a file rather than displaying it in the browser window. • If a response states that it contains HTML content but does not specify a character set, then the browser may analyze the HTML and attempt to determine which character set it appears to be using. Even if the majority of the HTML actually employs a standard character set such as UTF-8, the presence of non-standard characters anywhere in the response may cause the browser to interpret the content using a different character set. This can have unexpected results, and can lead to cross-site scripting vulnerabilities in which non-standard encodings like UTF-7 can be used to bypass the application's defensive filters. Remediation: • Use a suitable "Referrer-Policy" header, such as "no-referrer" or "same-origin". • Don't use the Origin header for authentication or access control decisions, as the Origin header can easily be changed by an attacker. • Use the Content-Disposition response header for file downloading in all API responses with the following directives: Content-Disposition:	Recommended	No probado

		attachment; filename="api.json" (or other appropriate filename for the content type). • Use a content-type header specifying a safe character set (e.g., UTF-8, ISO 8859-1) in every HTTP response.		
identify-dns-domains	Identify the DNS domains periodically to update the information about them	The application should have a protection from sub-domain takeovers if the application relies upon DNS entries or DNS sub-domains, such as expired domain names, out of date DNS pointers or CNAMEs, expired projects at public source code repos, or transient cloud APIs, serverless functions, or storage buckets (autogen-bucket-id.cloud.example.com) or similar. Protections can include ensuring that DNS names used by applications are regularly checked for expiry or	Recommended	No probado
identify-sensitive-data	Identify the processed and stored sensitive data	Create a mechanism or procedures to identify all created and processed data by the application and classify it to determine which level of sensitivity each piece of data belongs to. Each data category can then be mapped to security rules necessary for each level of sensitivity. Therefore, we should create a policy explaining how sensitive data should be managed and processed. For example, public marketing information that is not sensitive may be categorized as public data which is fine to place on the public website. Credit card numbers may be classified as private user data which may need to be encrypted whilst stored or in transit.	Recommended	No probado

implement-monitoring-unusual-activities	Implement and monitor Business Logic Limits for actions and transactions	Implement in the application the appropriate limits for specific business actions and transactions, to enforce the correct usage of the application for each user ensuring they cannot bypass important steps in the flow. This business logic limits provide us protection against likely business risks or threats, identified using threat modeling or similar methodologies. All business actions or transactions that exceed the established limits, should be recorded in the logs and monitored as unusual events.	Recommended	No probado

look-up-secret-auth	Implement Look-Up Secret Authentication	Verifiers of look-up secrets should prompt the claimant for the next secret from their authenticator or for a specific secret. A given secret from an authenticator must be used successfully only once. If the look-up secret is derived from a grid card, each cell of the grid shall be used only once. Verifiers should store look-up secrets in a form that is resistant to offline attacks and have: • At least 112 bits entropy and hashed with an approved one-way function. • With fewer than 112 bits of entropy, it should be salted and hashed using a suitable one-way key derivation function, and salt value of at least 32 bits in length and arbitrarily chosen so as to minimize salt value collisions among stored hashes. • Less than 64 bits of entropy, the verifier must implement a rate limiting mechanisms that effectively limits the number of failed authentication attempts that can be made on the subscriber's account Look up secrets: pre-generated lists of secret codes, similar to Transaction Authorization Numbers (TAN), social media recovery codes, or a grid containing a set of random values. These are distributed securely to users. These lookup codes are used once, and once all used, the lookup secret list is discarded. This type of authenticator is considered "something you have". https://pages.nist.gov/800-63-3/sp800-63b.html Verifier: An entity that verifies the claimant's identity by verifying the claimant's possession and control of a token using an authentication protocol. To do this, the Verifier may also need to validate credentials that link the token and identity and check their status. https://csrc.nist.gov/glossary/term/verifier	Recommended	No probado

network-rate-limit	Implement application and network rate limiting	A number of attacks rely on brute-force techniques to send large volumes of requests to enumerate or attempt to exploit flaws in an application, for example, sending common passwords to multiple target accounts within an application. By profiling normal traffic volumes, and applying rate limiting, the application can be built to actively mitigate such attacks. • Connection rate-limiting based on the source IP address can be used to restrict attacks against the authentication or registration systems. Multiple failures (or attempts) from a single IP should result in temporarily blocking or dropping traffic from the source. Note however that some corporate and ISP environments may place multiple valid and discrete clients behind the same IP address, resulting in false-positives. • Attackers may use botnets and other IP masking techniques to deliver attacks to avoid IP based rate-limiting. To mitigate this class of attack, Indicators of Compromise should be monitored (for example a higher rate of login failures than usual), and appropriate actions taken. For example, when the application detects active brute-force attacks, a Web Application Firewall (WAF) or other intermediate devices could be used to block attacks sharing a signature from pattern matching or deep packet inspection (e.g. HTTP headers or common passwords across multiple accounts). Similarly, the application could respond by requiring a CAPTCHA, cookie, or Javascript challenge when an attack is detected. Remediation: Implement the mechanisms to lockout accounts: • When the application detects a set number of failure login attempts, the account shall be locked for a certain time period. This period shall be increased as per each new failed attempt up to an hour as maximum.	Recommended	No probado

		• When the application detects an account is locked more times than usual, this account should be disabled (no more than 100 failure attempts). A disabled account should only be restored by an administrator. • When the application detects active brute-force attacks, the application shall require a CAPTCHA, cookie, or JavaScript challenge before attempting authentication. • Only accept those authentication requests that come from a white list of IP addresses from which the user has been successfully authenticated before. • If the user successfully authenticates, the previous failed attempts shall be reset for that user from the same IP address.		
password-change-facility	Offer a password change facility	Users must retain control of their credentials in a single factor system, and to facilitate this they must have the ability to change their password. This is particularly important should the user determine their password has been compromised, potentially allowing a third party to access the system impersonating them. Furthermore, accountability is adversely affected where users cannot change their passwords, as they can plausibly claim that even though they knew their account credentials were compromised, there was no way for them to prevent unauthorized use of the account. • Users should be allowed to change their passwords, ideally within the application • Password change should require re-authentication to prevent abuse of a compromised or unattended session. I.e. the user should be required to enter their existing password in order to change it. • After a successful password change, the application should terminate all other active sessions.	Recommended	No probado

password-reset-email-best-practices	Implement Password reset email design best practices	The password/account recovery process must have equivalent security to the normal authentication process. It should be resistant to brute-force or network monitoring attacks. • Passwords should not be sent in plain text (for example in an email), as they may be intercepted or monitored. • The client-side and the server-side should communicate by a secured and independent channel. • The application should verify the user's identity, and leverage a time-limited (not longer than 10 minutes) one-time password token (TOTP) from the user. • Alternatively, the application could require the user enters a soft-token (e.g. Google authenticator), a mobile-push / SMS code, or previously established off-line recovery mechanism (for example a list of pre-defined recovery codes). • It should be noted that use of a random value in an e-mail or SMS has known weaknesses, and should be considered a less secure option. • When the user provides the token, the application should verify their identity (for example through the use of a 'secret question', or verification of recent activity or transaction information). • The application should then require they choose a new password or phrase in line with the chosen policy. • For change and reset password functionality it is recommended to keep a history of old passwords (salted hashes used). These can be leveraged to prevent password reuse, or an attacker alternating between a selection of passwords (Monday1, Tuesday2 etc.).	Recommended	No probado

pseudo-random-number-generator	Use a cryptographically secure pseudo-random number generator	Use a well-vetted algorithm that is currently considered to be strong and secure by experts in the field and ensure well-tested implementation with adequate length seeds. In general, if a pseudo-random number generator is not advertised as being cryptographically secure, then it is probably a statistical PRNG and should not be used in a security-sensitive context. Pseudo-random number generators can produce predictable numbers if the generator is known and the seed can be guessed. A 256-bit seed is a good starting point for producing a "random enough" number. Remediation: Ensure that all random numbers, especially those used for cryptographic parameters (keys, IV's, MAC tags), random file names, random GUIDs, and random strings are generated in a cryptographically strong fashion. Ensure that random algorithms are seeded with sufficient entropy, containing at least 112 bits of entropy (typically a six digit random number is sufficient). If less than 112 bits of entropy, ensure salting with a unique and random 32-bit salt and hashed with an approved one-way hash. Use a secure pseudo-random number generator ensuring it leverages an appropriate length seed. The recommendation by FIPS 140-2 is 256-bit seed. Ensure that the challenge nonce was generated with at least 64 bits and it is unique over the lifetime of the cryptographic device. Tools like NIST RNG Test tool (as used in PCI PTS Derived Test Requirements) can be used to comprehensively assess the quality of a Random Number Generator by reading e.g. 128MB of data from the RNG source and then assessing its randomness properties with the tool. The following libraries are considered weak random numbers generators and should not be used. C library: random(), rand() instead use getrandom(2) Java library: java.util.Random() instead	Recommended	No probado

use java.security.SecureRandom
For secure random number generation, refer to NIST SP 800-90A. CTR-DRBG, HASH-DRBG, HMAC-DRBG. Refer to NIST SP800-22 A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications, and the testing toolkit.

require-use-strong-passwords-with-ui	Require the use of strong passwords with UI	Passwords used either as sole verification credentials, or as part of a multi-factor authentication, are a key aspect of application security, and strong password selection should be encouraged and enforced. The application should allow flexibility in user password selection, and enforce minimum criteria for password complexity. This should include: • Minimum password length requirements, to mitigate brute-force and dictionary attacks. • Use of pass-phrases using multiple words, achieving longer memorable passwords more resistant to attack. • Use of mixed case, numeric and/or special characters to increase complexity. • Preventing or discouraging use of dictionary words and common passwords through black-lists. For example, a set of commonly used passwords can be found on SecLists: https://github.com/danielmiessler/SecLists/tree/master/Passwords Password length: Password length considers the minimum and maximum length of characters comprising the password. For ease of changing this length, its implementation could be configurable using a properties file or xml configuration file. • Minimum length. • Memory secrets should be at least 8 characters long. • Memory secrets generated automatically should be at least 6 numeric characters. • Maximum length. People tend to forget their passwords easily. The longer the password, the more likely people are to enter them incorrectly. However, long pass-phrases can be easily remembered, and should not be prevented through unnecessarily strict upper restrictions on length. Passwords with 64 characters or longer should be permitted. Password Complexity:	Recommended	No probado

• Passwords with consecutive multiple spaces should be coalesced and converted into only one space. After this modification, the password length should still be at least 12 characters long.
• Unicode characters will be permitted in the password. A single Unicode code point is considered a character.
• Reject those passwords commonly used and those leaked in a previous compromise. You may choose to block the top 1000 or 10000 most common passwords which meet the above length requirements and are found in compromised password lists. The following link contains the most commonly found passwords: https://github.com/danielmiessler/SecLists/tree/master/Passwords
• Plain text passwords must not be stored, to protect them against brute forcing if the database is compromised and screen reader support enabled.
UI features:

• Allow the "paste" functionality, browser password helpers, and external password managers.
• Mask the entire password as typed, or show briefly the last typed character of the password on platforms that do no have this as native functionality.
• Include a password strength meter in pages in which the password is created, to help users generate a stronger password.
Password Topologies:

• Ban commonly used password topologies.
• Force multiple users to use different password topologies.
• Require a minimum topology change between old and new passwords.
Additional Information:

• Make sure that every character the user types in is actually included in the password. We've seen systems truncate the password at a length shorter than the user provided (e.g.,

		truncated at 15 characters when they entered 20). • As application's require more complex password policies, they need to be very clear about what these policies are. The required policy needs to be explicitly stated on the password change page • If the new password doesn't comply with the complexity policy, the error message should describe EVERY complexity rule that the new password does not comply with, not just the 1st rule it doesn't comply with.		
revoke-compromised-authentication-tokens	Implement an effective process to revoke compromised authentication tokens	Theft or loss of an authentication token can undermine security in any multi-factor authentication process. Loss of a physical token or mobile device with soft-tokens, or compromise of a client-side certificate could mean an attacker has all of the factors required to authenticate to a system and bypass the protections provided by multi-factor authentication. • Define an organization process to identify or report lost or compromised authentication tokens/factors • Implement technical controls to suspend or revoke compromised tokens or accounts • These processes should be implementable in a timely manner to reduce the attack window • Implement mechanisms to renew authentication tokens/factors within a sufficient timeframe before they are revoked.	Recommended	No probado

same-encoding-parsers	Use the same encodings and parsers in all application components	Errors may occur when converting between differently coded character data. There are two general types of encoding errors. If the byte sequence is not valid for the specified charset then the input is considered malformed. If the byte sequence cannot be mapped to an equivalent character sequence then an unmappable character has been encountered. Special care should be taken when decoding untrusted byte data to ensure that malformed input or unmappable character errors do not result in defects and vulnerabilities. Encoding errors can also occur, for example, encoding a cryptographic key containing malformed input for transmission will result in an error. Encoding and decoding errors typically result in data corruption. **Remediation** Use the same encodings and parsers in all application components to avoid parsing attacks that exploit different URI or file parsing behavior that could be used in SSRF and RFI attacks.	Recommended	No probado

secure-attributes-of-cookies	Secure Cookies with their attributes	The session ID exchange mechanism based on cookies provides multiple security features in the form of cookie attributes that can be used to protect the exchange of the session ID: Secure and HttpOnly Attribute: a secure cookie is only sent to the server with an encrypted request over the HTTPS protocol. Even with Secure, sensitive information should never be stored in cookies, as they are inherently insecure and this flag doesn't offer robust protection. Starting with Chrome 52 and Firefox 52, insecure sites (http:) can't set cookies with the Secure directive. To mitigate cross-site scripting (XSS) attacks, HttpOnly cookies are inaccessible to JavaScript's Document.cookie API; they are only sent to the server. For example, cookies that persist server-side sessions don't need to be available to JavaScript, and so the HttpOnlyflag should be set. Set-Cookie: id=a3fWa; Expires=Wed, 21 Oct 2015 07:28:00 GMT; Secure; HttpOnly SameSite Attribute: The SameSite attribute should be configured to limit exposure against cross-site request forgery attacks. There are two possibilities: • Strict: the browser will only send cookies if the request originated from the website that set the cookie. If the request originated from a different URL than the URL of the current location, none of the cookies tagged with the Strict attribute will be included. • Lax: if the attribute is set to Lax, same-site cookies are withheld on cross-site subrequests, such as calls to load images or frames, but will be sent when a user navigates to the URL from an external site, for example, by following a link.	Recommended	No probado

By default SameSite is not configured, we need to configure it with one of the above types, as shown below:
Set-Cookie: key=value; SameSite=Strict

Domain and Path Attributes: The session token value issued after users have successfully identified and authenticated themselves is of equivalent value to the secrets the user presents to login, and must be protected accordingly.

- If the cookie domain attribute is too liberal, the cookie may be accessible to other hosts within the parent domain. This is a particular issue in multi-tenanted hosting, or where applications are delivered as subdomains of a third party parent domain.
- If the cookie path attribute is not appropriately restricted, the session token may be accessible from other applications or content on the server. The domain and the path attributes for the session cookie should be restricted to the fully qualified hostname on which the application is running.

For example:
Set-Cookie: session=token; Path=/theapp/; Domain=myapp.example.com; secure; HTTPOnly

__Host- prefix: ensure that all Cookies with a name starting with __Host- are:

- set with the secure flag,
- from a secure page (HTTPS),
- does not have a domain specified (and therefore aren't sent to subdomains)

secure-communication-ra-and-csp	Use secure communication between CSP and RA	In situations where the verifier and CSP are separate entities, communications between RA (Registration Authority) and CSP (Credential Service Provider), should be authenticated with a mutually-authenticated secure channel, such as a client-authenticated TLS connection, using approved cryptographic algorithms. NIST's digital identity model involves some entities: - CSP (Credential Service Provider): A credential service provider is a trusted entity that issues security tokens or electronic credentials to subscribers. A CSP forms part of an authentication system, most typically identified as a separate entity in a Federated authentication system. A CSP may be an independent third party, or may issue credentials for its own use. Credential Service Provider is typically also the Identity Provider (IDP). An example of a CSP would be an online site whose primary purpose may be, for example, internet banking - but whose users may be subsequently authenticated to other sites, applications or services without further action on their part. - Registration Authority (RA): A trusted entity that establishes and vouches for the identity of a subscriber to a CSP. The RA may be an integral part of a CSP, or it may be independent of a CSP, but it has a relationship to the CSP. - Verifier: This refers to an entity that verifies the claimant's identity by verifying the claimant's possession and control of one or two authenticators, using an authentication protocol. - RP (Relying party): This refers to an entity that relies on the subscriber's authenticator(s) and credentials or a verifier's assertion of a claimant's identity, typically to process a transaction or grant access to information or a system. - Applicant: This refers to a subject undergoing the processes of enrollment and identity proofing. - Claimant: This refers to a subject whose identity is to be verified using	Recommended	No probado

		one or more authentication protocols. - Subscriber: This refers to a party who has received a credential or an authenticator from a CSP.		
secure-file-storage	Ensure the application is implementing file storage protections for uploaded files	Uploaded files represent a significant risk to applications. In one scenario one of the risks could be related to how the application manages file storage for uploaded files. A cheap and easy way to perform a denial-of-service attack is to upload a very large file, in the hope the server runs out of space. It is necessary to limit (per-user) the amount of resources that are accessible to unprivileged users and allow the system administrator to configure these limits. For any security checks that are performed on the client side, ensure these same checks are duplicated on the server side. Remediation: The application shall: • Not allow or accept large files that could fill up storage or cause a denial of service attack. • Check compressed files to look for "zip bombs" - small input files that will decompress into huge files thus exhausting file storage limits. • Set a file size quota and maximum number of files per user ensuring a single user cannot fill up the storage with too many files, or excessively large files.	Recommended	No probado

secure-recovery-token-reset-account	Use a secure recovery token to reset an account password (such as a TOTP)	The password/account recovery process must have equivalent security to the normal authentication process. It should be resistant to brute-force or network monitoring attacks. • Having identified the user through secret questions/answers, the application should proceed with a secure reset process. • Passwords should not be sent in plain text (for example in an email), as they may be intercepted or monitored. • Instead, the application should verify the user's identity, and use a time-limited one-time password token (TOTP). • If a time-based multi factor OTP token is re-used during the validity period, it should be logged and rejected with secure notifications sent to the holder of the device. • Alternatively, the application could require the user enters a soft-token (e.g. Google authenticator), a mobile-push / SMS code, or previously established off-line recovery mechanism (for example a list of pre-defined recovery codes). It should be noted that use of a random value in an e-mail or SMS has known weaknesses, and should be considered a less secure option. • When the user provides the token, the application should verify their identity (for example through the use of a 'secret question', or verification of recent activity or transaction information). • If any token OTP has been sent in cases of theft or other loss, the validity of the token must be immediately and effectively revoked across the logged-in sessions. • The application should then require they choose a new password or phrase in line with the chosen policy.	Recommended	No probado

secure-session-generation-and-expiration	Implement a secure Session Generation and Expiration	The session ID exchange mechanism based on cookies provides multiple security features in the form of Session Generation and Expiration which is tracked in a session. This session is typically stored on the server for traditional web based session management. A session identifier is then given to the user so they can identify which server-side session contains the correct user data. The client only needs to maintain this session identifier, which also keeps sensitive server-side session data off of the client. Here are a few controls to consider when building or implementing session management solutions: • Ensure that the session id is long, unique and random. The session ID length must be at least 128 bits (16 bytes) • The session ID must be unpredictable (random enough) to prevent guessing attacks such as when an attacker is able to guess or predict the ID of a valid session through statistical analysis techniques. For this purpose, a good PRNG (Pseudo Random Number Generator) must be used. The session ID value must provide at least 64 bits of entropy (if a good PRNG is used, this value is estimated to be half the length of the session ID). • The application should generate a new session or at least rotate the session id during authentication and re-authentication. • The application should not treat OAuth and refresh tokens as their own session identifiers. Moreover, it should allow users to terminate trust relationships with linked applications. • The CSPs (Credential Service Provider) should re-authenticate the subscriber if it has not been authenticated since a given inactivity period. CSPs should inform the RP (Relying Parties) the last authentication time, to know if re-authentication is needed. • When a user chooses to log out (or otherwise has their session terminated), the user session must be	Recommended	No probado

terminated on the server-side, not solely at the client-side (for example by deleting or reissuing a session ID token/cookie).

- Revoke user sessions on the server-side.
- Redirect users to the login page, or provide a login prompt.
- Where possible, ensure the session is not reused by the application server.
- The application should implement an idle timeout after a period of inactivity and an absolute maximum lifetime for each session, after which users must re-authenticate. If the application has implemented a functionality to be always logged in, the application must force users to re-authenticate for more sensitive actions.
- The length of the timeouts should be inversely proportional with the value of the data protected.
- 30 days
- 12 hours or 30 minutes of inactivity, 2FA optional
- 12 hours or 15 minutes of inactivity, with 2FA

security-logging	Implement Security Logging and Monitoring	Logging is often neglected by developers when thinking of security considerations. However, proper logging practice can provide the crucial forensics needed to investigate after a breach, and perhaps more importantly, to detect security issues as they happen. Most developers are already familiar with using logging for debugging and diagnostic purposes, so it should be easy for them apply concept of security logging. Remediation: • Log all failed authentication attempts, denied access, and input validation errors. • Logs should be written using a format suited to be consumed by a log management solution, and include enough detail to identify the malicious actor. • Logs need to be handled as sensitive data, and their integrity should be guaranteed at rest and transit. • Configure a monitoring system to continuously monitor the infrastructure, network, and the API functioning. • Use a Security Information and Event Management (SIEM) system to aggregate and manage logs across all components and hosts. • Configure custom dashboards and alerts, enabling suspicious activity to be detected and responded to as early as possible.	Recommended	No probado

store-backups-securely	Encrypt Backups securely on the host (data at rest)	Sensitive data and Backups of sensitive data shall be stored securely by encryption (data at rest). • Cryptographically strong symmetric or asymmetric (public-key) encryption should be used to protect the data. • Encryption should be performed before the data is written to disk or other persistent storage. • The key for encrypting and decrypting the data should not be accessible from the same host. • The encryption and decryption operation should be performed on a different host. • A recognized, proven, and tested implementation/library should be used (in preference to a bespoke implementation).	Recommended	No probado

store-passwords-unrecoverable-form	Store passwords in unrecoverable form	To protect user passwords from accidental or deliberate exposure, the application should store cryptographic hashes of passwords instead of the passwords themselves. Do not store passwords in the data store for verification at login. Instead, create a cryptographic hash of the password using a strong hash function that includes a work factor and a built in 'salt' value, like bcrypt or scrypt. This reduces the risk of brute-force attacks and rainbow tables, and allows flexibility to adapt the hashing function to balance security and performance. Here are some general guidelines: The salt length should be at least 32 bits. If bcrypt is used, the work factor should be as large as the verification server performance will allow, typically at least 13. If PBKDF2 is used, the iteration count should be as large as the verification server performance will allow, typically at least 100,000 iterations. Alternatively, use a strong hash function like SHA-384 together with a unique 'salt' value for every account. Apply multiple iterations of the hash to create the additional computational work required to mitigate brute-force attacks. And the secret 'salt' value should be stored separately from the hashed passwords.	Recommended	No probado
template-val	Verify the application protects against template injection attacks	Assure that: • The application protects against template injection attacks by verifying that any user input being included is sanitized or sandboxed. • The application sanitizes, disables, or sandboxes user-supplied scriptable or expression template language content, such as Markdown, CSS or XSL stylesheets, BBCode, or similar.	Recommended	No probado
use-analysis-static-code	Scan the code with static code analysis tools	Periodically leverage a static code analysis tool to analyze the code and detect potentially malicious code, such as time functions, unsafe file operations and networks connections. Scanning of static code, may improve the integrity of the application.	Recommended	No probado

CWE-306-SERVICE	Require authentication before presenting restricted data	The application should ensure users have undergone an Identification and Verification (ID&V) process before allowing access to secret, sensitive or otherwise restricted data. For less sensitive but still restricted data, simple verification of the location of the user may suffice (e.g. IP restrictions). • For non-sensitive but non-public data, access could be restricted by IP address, limiting access to internal networks, workstations, or gateways • For more sensitive data, TLS client-side certificates may be appropriate • Where secret or other sensitive data is handled, a full authentication process to identify and validate users with single or multi-factor authentication may be required	Required	No probado
CWE-319-TRANSPORT	Encrypt data between the client and server/service	Data passed between the client and server should be protected by encryption in transit. • Implement cryptographically strong TLS end-to-end encryption between the client and server, terminating within a secure environment on the server-side. • Consider use of client certificates to prevent interception of (or man-in-the-middle attacks on) the encrypted connection. • Alternatively, asymmetric (public-key) encryption could be utilized and a recognized, proven, and tested implementation/library should be used	Required	No probado

Component: Web Client

Id	Name	Description	State	Result
ASVS-7.2	Verify that all cryptographic modules fail securely, and errors are handled in a way that does not enable oracle padding.	Encryption should fail closed so that data does not get stored in an unprotected state. To protect against padding oracles, you want to make sure that your application does not return a different error when the padding is wrong. The best way to do this is an Encrypt-then-MAC construction, where a Message Authentication Code (MAC) is applied to the ciphertext. If the MAC fails, you don't even need to look at the padding. If the MAC is correct, it is cryptographically unlikely that the padding has been tampered with.	Recommended	No probado

CS-CLIENT-SECRET	Implementar lógica sensible y validación de datos en el lado del servidor	Todos los datos y la funcionalidad del lado del cliente deben considerarse corrompidos y sujetos a manipulación por parte de un actor malintencionado, independientemente de los controles establecidos (por ejemplo, cifrado u ofuscación del código). Esto incluye decisiones lógicas de la aplicación, como controles de acceso, y datos considerados confidenciales, como claves de cifrado. Si bien los controles pueden duplicarse en el lado del cliente para mejorar la experiencia del usuario, es importante que se apliquen en el lado del servidor. La lógica empresarial, en particular los controles de acceso, debe implementarse en el lado del servidor. Los datos secretos deben almacenarse en el lado del servidor, y solo los que se requieren explícitamente deben duplicarse en el lado del cliente. Todos los datos proporcionados desde el lado del cliente deben considerarse corrompidos y las decisiones de seguridad no deben tomarse únicamente sobre estos datos. Por ejemplo, una implementación insegura podría incluir una aplicación compilada (por ejemplo, Flash) que realiza la autenticación en el lado del cliente con una contraseña almacenada. La descompilación del binario (en este caso el archivo flv) o la interceptación y modificación de los controles entre el cliente y el servidor podría exponer la contraseña o permitir el acceso no autorizado al contenido de la aplicación autenticada.	Recommended	No probado

CSD-CLIENT-SECRET	Revise el código, la configuración y los repositorios on-line en busca de información confidencial de manera sistemática.	La información secreta o sensible no debe exponerse en el código fuente o en el software del lado del cliente. • Los repositorios de código y en línea no deben contener secretos o información sensible. • Los archivos de configuración del lado del cliente y del servidor no deben contener credenciales codificadas ni otros datos confidenciales. • Los repositorios, servicios y otros contenidos restringidos asociados en línea deben tener los controles de acceso adecuados. • Cuando la información confidencial deba almacenarse en el lado del servidor, debe estar en la memoria para datos transitorios o cifrada si es persistente. Dependiendo del caso de uso, esta encriptación puede ser asimétrica o de clave pública, o simétrica usando un secreto compartido. Sin embargo, tenga en cuenta que el cifrado reversible en el lado del cliente y el almacenamiento en memoria siguen siendo susceptibles a los ataques del lado del cliente.	Recommended	No probado
CSD-KEY-MGMT	Isolate cryptographic processes - including master secrets - and consider the use of a virtualized or physical hardware key vaults (HSM).	Cryptographic processes require the use of a private key or a secret. If the private key is stored poorly in the same service to be accessed, a compromise of the system could lead to this private key or secret being stolen. This would give the attacker access to all information. The way to prevent this is to keep cryptographic functionalities on a separate service that performs the encryption/decryption on demand. It is better if this service uses virtualized or physical hardware key vaults like a Hardware Security Module because those devices store the private key and secrets securely. Key vaults and such implement security measures to prevent this information being extracted from them	Recommended	No probado

CWE-295-CLIENT	Validar el certificado presentado por el servidor	Los certificados presentados por el servidor deben ser válidos, demostrables y criptográficamente seguros para obtener los beneficios de las comunicaciones cifradas TLS. Los certificados no válidos exponen la aplicación y sus usuarios a ataques de suplantación de identidad, como Man In The Middle (MITM), donde los datos pueden ser interceptados o redirigidos a un sitio malicioso y socavar la confianza en el sitio. Validar que el certificado presentado por el servidor esté firmado por una autoridad certificadora de confianza. Asegúrese de que la cadena de confianza del certificado sea legítima y segura. Verifique que se utilicen cifrados criptográficamente seguros en toda la cadena, en particular, algoritmos de hash sólidos para firmas de certificados.	Recommended	No probado
CWE-319-TRANSPORT	Cifre los datos entre el cliente y el servidor / servicio	Los datos transmitidos entre el cliente y el servidor deben protegerse mediante cifrado en tránsito. • Implemente un cifrado de extremo a extremo TLS criptográficamente fuerte entre el cliente y el servidor, terminando dentro de un entorno seguro en el lado del servidor. • Considere el uso de certificados de cliente para evitar la interceptación (o ataques man-in-the-middle) de la conexión cifrada. • Alternativamente, se podría utilizar el cifrado asimétrico (clave pública) y se debería usar una implementación / biblioteca reconocida, probada y testeada	Recommended	No probado

CWE-327	Validar la implementación de TLS en uso	Hay una serie de implementaciones TLS establecidas de código abierto, que han sido objeto de una revisión y prueba exhaustivas por parte de los desarrolladores, así como de terceros en el espacio de desarrollo, seguridad y criptografía. Por lo general, estas implementaciones ampliamente utilizadas tienen un menor riesgo de explotación en comparación con las implementaciones desarrolladas de forma privada, o menos conocidas (aunque vulnerabilidades como Heartbleed destacan que esto no es universalmente cierto) debido a este escrutinio adicional. • Identifique TLS y otras bibliotecas, implementaciones y tecnologías criptográficas en uso. • Verifique la veracidad de las implementaciones y revise las validaciones de terceros junto con los avisos de seguridad o las mejores prácticas relacionadas con ellas. • Algunas bibliotecas TLS conocidas son NSS de Mozilla y OpenSSL. • Cuando se esté utilizando una implementación desconocida o no probada, considere el uso de una opción establecida.	Recommended	No probado

CWE-592	Asegurar la autenticación del lado del servidor	Todos los datos del lado del cliente deben considerarse corrompidos. Como tal, las decisiones sobre autenticación (o Identificación y Verificación) deben tomarse o validarse en el lado del servidor. • Los mensajes devueltos al usuario durante la autenticación, en particular cuando el proceso falla, no deben revelar al usuario si el nombre de usuario era válido. La devolución de errores genéricos evita que un atacante liste identificadores de cuenta válidos para ataques posteriores. • El servidor debe comparar las credenciales con las almacenadas en el lado del servidor (por ejemplo, la contraseña con hash en comparación con las almacenadas para la identidad reclamada).	Recommended	No probado
INS-CLIENT	Asegurar que la aplicación no utilice tecnologías del lado del cliente no admitidas, inseguras o obsoletas	Asegurar que la aplicación no utiliza tecnologías no admitidas, inseguras o obsoletas del lado del cliente como: complementos NSAPI, Flash, Shockwave, ActiveX, Silverlight, NACL o subprogramas Java del lado del cliente.	Recommended	No probado
STRONG-ALGORITHMS-PROTOCOLS	Asegurar que sólo se utilicen algoritmos, cifrados y protocolos sólidos	Asegúrese de que la aplicación utilice algoritmos, cifrados y protocolos sólidos. La práctica general y las longitudes mínimas de clave requeridas dependen de los escenarios que se enumeran a continuación. • Intercambio de claves: intercambio de claves Diffie-Hellman con un mínimo de 2048 bits • Integridad del mensaje: HMAC-SHA2 • Hash de mensaje: SHA2 256 bits • Cifrado asimétrico: RSA 2048 bits • Algoritmo de clave simétrica: AES 128 bits • Hash de contraseña: Argon2, PBKDF2, Scrypt, Bcrypt.	Recommended	No probado

crypto-operations	Asegurar que la aplicación implementa operaciones criptográficas conocidas	La criptografía es uno de los temas más avanzados de la seguridad de la información, cuya comprensión requiere formación y experiencia. Es difícil acertar porque hay muchos enfoques de encriptación, cada uno con ventajas y desventajas que los arquitectos y desarrolladores de soluciones web deben comprender a fondo. Además, la investigación criptográfica seria suele basarse en las matemáticas avanzadas y la teoría de los números, lo que supone una importante barrera de entrada. En lugar de crear una capacidad criptográfica desde cero, se recomienda que se utilicen soluciones abiertas y revisadas por pares, como el proyecto Google Tink, Libsodium y el almacenamiento seguro integrado en muchos framewoks de software y servicios en la nube. Usar criptografía de forma segura es más complicado que elegir los algoritmos criptográficos correctos. También requiere implementaciones seguras, API seguras, administración segura de claves, generación segura aleatorea y protocolos seguros. Remediación: * Los algoritmos de cifrado o hash, longitudes de clave, cifrados de números aleatorios, se reconfigurarán, actualizarán o intercambiarán en cualquier momento para protegerlos contra interrupciones criptográficas. * Los datos cifrados se autenticarán mediante firmas, modos de cifrado autenticados o HMAC para garantizar que el texto cifrado no sea alterado por una parte no autorizada. * Las operaciones criptográficas serán en tiempo constante, sin operaciones de 'cortocircuito' en comparaciones, cálculos o retornos, para evitar filtraciones de información.	Recommended	No probado

follow-jwt-standard-generation-token	Implementar la generación de tokens para serverless siguiendo la recomendación del estándar JWT	JSON Web Token (JWT) es un estándar abierto (RFC 7519) que define una forma compacta y autónoma de transmitir información de forma segura entre las partes. Esta información puede ser verificada porque está firmada digitalmente. Un token JWT se crea durante la autenticación y es verificado por el servidor (o servidores) antes de cualquier procesamiento. Sin embargo, los JWT a menudo no son guardados por el servidor después de la creación inicial. Los JWT generalmente se crean y luego se entregan a un cliente sin que el servidor los guarde de ninguna manera. La integridad del token se mantiene mediante el uso de firmas digitales para que un servidor pueda verificar posteriormente que el JWT sigue siendo válido y no ha sido manipulado desde su creación. Este enfoque no tiene estado y es portátil en la forma en que las tecnologías de cliente y servidor pueden ser diferentes y aun así interactuar. Asegure la implementación de los siguientes requisitos para proteger el token sin estado para firmas digitales, cifrado y otras contramedidas siguiendo los requisitos del estándar JWT. **Remediación:** Para crear un JWT, se realizan los siguientes pasos. El orden de los pasos no es significativo en los casos en los que no hay dependencias entre las entradas y salidas de los pasos. 1. Cree un conjunto de reclamaciones de JWT que contenga las reclamaciones deseadas. Tenga en cuenta que los espacios en blanco están explícitamente permitidos en la representación y no es necesario realizar una canonicalización antes de la codificación. 2. Sea el mensaje los octetos de la representación UTF-8 del conjunto de reclamaciones JWT. 3. Cree un encabezado JOSE que contenga el conjunto deseado de parámetros de encabezado. El JWT DEBE cumplir con las especificaciones [JWS] o [JWE]. Tenga en cuenta que los espacios en blanco están explícitamente permitidos en la representación y no es necesario	Recommended	No probado

realizar una canonización antes de la codificación. 4.Dependiendo de si el JWT es un JWS o un JWE, existen dos casos:

- Si el JWT es un JWS, cree un JWS utilizando el mensaje como la carga útil de JWS; DEBEN seguirse todos los pasos especificados en [JWS] para crear un JWS.
- De lo contrario, si el JWT es un JWE, cree un JWE utilizando el mensaje como texto sin formato para el JWE; DEBEN seguirse todos los pasos especificados en [JWE] para crear un JWE.

5. Si se realizará una operación de cifrado o firma anidada, deje que el mensaje sea el JWS o JWE, y vuelva al paso 3, utilizando un valor "cty" (tipo de contenido) de "JWT" en el nuevo encabezado JOSE creado en ese paso. De lo contrario, deje que el JWT resultante sea el JWS o JWE.

pseudo-random-number-generator	Utilice un generador de números pseudoaleatorios criptográficamente seguro	Utilice un algoritmo que los expertos en el campo consideren fuerte, seguro y que garantice una implementación bien probada con semillas de longitud adecuada. En general, si un generador de números pseudoaleatorios no se anuncia como criptográficamente seguro, entonces probablemente sea un PRNG estadístico y no debería usarse en un contexto sensible a la seguridad. Los generadores de números pseudoaleatorios pueden producir números predecibles si se conoce el generador y se puede adivinar la semilla. Una semilla de 256 bits es un buen punto de partida para producir un número "suficientemente aleatorio". Remediation: Ensure that all random numbers, especially those used for cryptographic parameters (keys, IV's, MAC tags), random file names, random GUIDs, and random strings are generated in a cryptographically strong fashion. Ensure that random algorithms are seeded with sufficient entropy, containing at least 112 bits of entropy (typically a six digit random number is sufficient). If less than 112 bits of entropy, ensure salting with a unique and random 32-bit salt and hashed with an approved one-way hash. Use a secure pseudo-random number generator ensuring it leverages an appropriate length seed. The recommendation by FIPS 140-2 is 256-bit seed. Ensure that the challenge nonce was generated with at least 64 bits and it is unique over the lifetime of the cryptographic device. Tools like NIST RNG Test tool (as used in PCI PTS Derived Test Requirements) can be used to comprehensively assess the quality of a Random Number Generator by reading e.g. 128MB of data from the RNG source and then assessing its randomness properties with the tool. The following libraries are considered weak random numbers generators and should not be used. C library: random(), rand() instead use getrandom(2)	Recommended	No probado

Java library: java.util.Random() instead use java.security.SecureRandom
For secure random number generation, refer to NIST SP 800-90A. CTR-DRBG, HASH-DRBG, HMAC-DRBG. Refer to NIST SP800-22 A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications, and the testing toolkit.

secure-communication-ra-and-csp	Use secure communication between CSP and RA	In situations where the verifier and CSP are separate entities, communications between RA (Registration Authority) and CSP (Credential Service Provider), should be authenticated with a mutually-authenticated secure channel, such as a client-authenticated TLS connection, using approved cryptographic algorithms. NIST's digital identity model involves some entities: - CSP (Credential Service Provider): A credential service provider is a trusted entity that issues security tokens or electronic credentials to subscribers. A CSP forms part of an authentication system, most typically identified as a separate entity in a Federated authentication system. A CSP may be an independent third party, or may issue credentials for its own use. Credential Service Provider is typically also the Identity Provider (IDP). An example of a CSP would be an online site whose primary purpose may be, for example, internet banking - but whose users may be subsequently authenticated to other sites, applications or services without further action on their part. - Registration Authority (RA): A trusted entity that establishes and vouches for the identity of a subscriber to a CSP. The RA may be an integral part of a CSP, or it may be independent of a CSP, but it has a relationship to the CSP. - Verifier: This refers to an entity that verifies the claimant's identity by verifying the claimant's possession and control of one or two authenticators, using an authentication protocol. - RP (Relying party): This refers to an entity that relies on the subscriber's authenticator(s) and credentials or a verifier's assertion of a claimant's identity, typically to process a transaction or grant access to information or a system. - Applicant: This refers to a subject undergoing the processes of enrollment and identity proofing. - Claimant: This refers to a subject whose identity is to be verified using	Recommended	No probado

		one or more authentication protocols. - Subscriber: This refers to a party who has received a credential or an authenticator from a CSP.		
use-analysis-static-code	Escanear el código con herramientas de análisis de código estático	Aproveche periódicamente una herramienta de análisis de código estático para analizar el código y detectar código potencialmente malicioso, como funciones de tiempo, operaciones de archivos inseguras y conexiones de red. El escaneo de código estático puede mejorar la integridad de la aplicación.	Recommended	No probado