Cloud Application Security Assessment (CASA) Letter of Assessment (LOA) for: MailMeteor's MailMeteor Application

Prepared by NCC Group
August 6th, 2022

In May of 2022, NCC Group performed a Full Cloud Application Security Assessment (CASA) against MailMeteor (the "**Application**") for MailMeteor and on behalf of Google, Inc. ("**Developer**") pursuant to the governing contract(s) between NCC Group and Developer. The assessment objective was to identify compliance with the CASA framework within a time-boxed assessment. CASA is defined by the App Defense Alliance (ADA) and is based on the OWASP Application Security Verification Standard (ASVS). For more specific information on the specific requirements assessed, please see Appendix A.

This Letter of Assessment ("**LOA**") confirms that the assessment of the Application has been completed and was found to substantially comply with the requirements in Appendix A.

It is important to note that this LOA represents a point-in-time evaluation. The security and compliance of an application can evolve rapidly, and the results of this assessment are not intended to represent an endorsement of the Application's future compliance or adequacy of current security measures against future threats. This LOA necessarily contains information in summary form and is therefore intended for general guidance only; it is not intended as a substitute for detailed research or the exercise of professional judgment. The information presented here should not be construed as professional advice or service.

Technical Constraints

The following items may impact the completeness and accuracy of the test case results:

- The CASA framework was in active development during the assessment. Some controls may have been modified during or after the testing period.
- Some controls employed ambiguous language. When presented with equally valid interpretations of a control, NCC Group selected the strictest version unless otherwise directed by Google.
- The assessment was designed to support Google's product risk management and assessment scope was limited to functionality that would affect Google Restricted Scopes.

Terms, Limitations and Disclaimers

- Prepared by NCC Group Security Services, Inc. for Developer.
- Portions of this document and the templates used in its production are the property of NCC Group and cannot be copied or disclosed (in full or in part) without NCC Group's prior written permission.
- While precautions have been taken in the preparation of this document, NCC Group the publisher, and the author(s) assume no responsibility for errors, omissions, or for damages resulting from the use of the information contained herein.

- NCC Group provides no warranty or guarantee that any of NCC Group’s services including but not limited to, recommendations, results or assessments will prevent or avoid any future security breaches or unauthorized access to the Application or Developer’s networks or systems.
- CASA is intended to provide more transparency into application security, however the limited nature of testing does not guarantee complete safety of the Application. This independent review may not be scoped to verify the accuracy and completeness of a developer's data safety declarations. Developer remains solely responsible for making complete and accurate declarations in their app's Google listings.
- NCC Group further expressly disclaims all warranties and conditions of any kind, whether express or implied, including, but not limited to the implied warranties and conditions of merchantability, fitness for a particular purpose and non-infringement.

APPENDIX (A)

CASA Full Requirements

Category	#	Description
Configuration Architecture	1.14.6	Verify the application does not use unsupported, insecure, or deprecated client-side technologies such as NSAPI plugins, Flash, Shockwave, ActiveX, Silverlight, NACL, or client-side Java applets.
Password Security	2.1.1	Verify that user set passwords are at least 12 characters in length (after multiple spaces are combined).
Password Security	2.1.2	Verify that passwords of at least 64 characters are permitted, and that passwords of more than 128 characters are denied.
Password Security	2.1.5	Verify users can change their password.
Password Security	2.1.6	Verify that password change functionality requires the user's current and new password.
Password Security	2.1.8	Verify that a password strength meter is provided to help users set a stronger password.
Password Security	2.1.12	Verify that the user can choose to either temporarily view the entire masked password, or temporarily view the last typed character of the password on platforms that do not have this as built-in functionality.
General Authenticator Security	2.2.1	Verify that anti-automation controls are effective at mitigating breached credential testing, brute force, and account lockout attacks. Such controls include blocking the most common breached passwords, soft lockouts, rate limiting, CAPTCHA, ever increasing delays between attempts, IP address restrictions, or risk-based restrictions such as location, first login on a device, recent attempts to unlock the account, or similar. Verify that no more than 100 failed attempts per hour is possible on a single account.
General Authenticator Security	2.2.3	Verify that secure notifications are sent to users after updates to authentication details, such as credential resets, email or address changes, logging in from unknown or risky locations. The use of push notifications - rather than SMS or email - is preferred, but in the absence of push notifications, SMS or email is acceptable as long as no sensitive information is disclosed in the notification.
Authenticator Lifecycle	2.3.1	Verify system generated initial passwords or activation codes SHOULD be securely randomly generated, SHOULD be at least 6 characters long, and MAY contain letters and numbers, and expire after a short period of time.

		These initial secrets must not be permitted to become the long term password.
Credential Recovery	2.5.1	Verify that a system generated initial activation or recovery secret is not sent in clear text to the user.
Credential Recovery	2.5.2	Verify password hints or knowledge-based authentication (so-called "secret questions") are not present.
Credential Recovery	2.5.3	Verify password credential recovery does not reveal the current password in any way.
Credential Recovery	2.5.4	Verify shared or default accounts are not present (e.g. "root", "admin", or "sa").
Credential Recovery	2.5.5	Verify that if an authentication factor is changed or replaced, that the user is notified of this event.
Credential Recovery	2.5.6	Verify forgotten password, and other recovery paths use a secure recovery mechanism, such as time-based OTP (TOTP) or other soft token, mobile push, or another offline recovery mechanism.
Out of Band Verifier	2.7.3	Verify that the out of band verifier authentication requests, codes, or tokens are only usable once, and only for the original authentication request.
Out of Band Verifier	2.7.4	Verify that the out of band authenticator and verifier communicates over a secure independent channel.
Single or Multi-factor One Time Verifier	2.8.6	Verify physical single-factor OTP generator can be revoked in case of theft or other loss. Ensure that revocation is immediately effective across logged in sessions, regardless of location.
Single or Multi-factor One Time Verifier	2.8.7	Verify that biometric authenticators are limited to use only as secondary factors in conjunction with either something you have and something you know.
Fundamental Session Management Security	3.1.1	Verify the application never reveals session tokens in URL parameters.
Session Binding	3.2.1	Verify the application generates a new session token on user authentication.
Session Termination	3.3.1	Verify that logout and expiration invalidate the session token, such that the back button or a downstream relying party does not resume an authenticated session, including across relying parties.
Session Termination	3.3.3	Verify that the application gives the option to terminate all other active sessions after a successful password change (including change via password reset/recovery), and that

		this is effective across the application, federated login (if present), and any relying parties.
Session Termination	3.3.4	Verify that users are able to view and (having re-entered login credentials) log out of any or all currently active sessions and devices.
Cookie-based Session Management	3.4.1	Verify that cookie-based session tokens have the 'Secure' attribute set.
Cookie-based Session Management	3.4.2	Verify that cookie-based session tokens have the 'HttpOnly' attribute set.
Cookie-based Session Management	3.4.3	Verify that cookie-based session tokens utilize the 'SameSite' attribute to limit exposure to cross-site request forgery attacks.
Cookie-based Session Management	3.4.4	Verify that cookie-based session tokens use the "__Host-" prefix so cookies are only sent to the host that initially set the cookie.
Cookie-based Session Management	3.4.5	Verify that if the application is published under a domain name with other applications that set or use session cookies that might disclose the session cookies, set the path attribute in cookie-based session tokens using the most precise path possible.
Token-based Session Management	3.5.1	Verify the application allows users to revoke OAuth tokens that form trust relationships with linked applications.
Token-based Session Management	3.5.2	Verify the application uses session tokens rather than static API secrets and keys, except with legacy implementations.
Token-based Session Management	3.5.3	Verify that stateless session tokens use digital signatures, encryption, and other countermeasures to protect against tampering, enveloping, replay, null cipher, and key substitution attacks.
Defenses Against Session Management Exploits	3.7.1	Verify the application ensures a full, valid login session or requires re-authentication or secondary verification before allowing any sensitive transactions or account modifications.
General Access Control Design	4.1.1	Verify that the application enforces access control rules on a trusted service layer, especially if client-side access control is present and could be bypassed.
General Access Control Design	4.1.2	Verify that all user and data attributes and policy information used by access controls cannot be manipulated by end users unless specifically authorized.
General Access Control Design	4.1.3	Verify that the principle of least privilege exists - users should only be able to access functions, data files, URLs, controllers, services, and other resources, for which they possess specific authorization. This implies protection against spoofing and elevation of privilege.

General Access Control Design	4.1.5	Verify that access controls fail securely including when an exception occurs.
Operation Level Access Control	4.2.1	Verify that sensitive data and APIs are protected against Insecure Direct Object Reference (IDOR) attacks targeting creation, reading, updating and deletion of records, such as creating or updating someone else's record, viewing everyone's records, or deleting all records.
Operation Level Access Control	4.2.2	Verify that the application or framework enforces a strong anti-CSRF mechanism to protect authenticated functionality, and effective anti-automation or anti-CSRF protects unauthenticated functionality.
Other Access Control Considerations	4.3.2	Verify that directory browsing is disabled unless deliberately desired. Additionally, applications should not allow discovery or disclosure of file or directory metadata, such as Thumbs.db, .DS_Store, .git or .svn folders.
Input Validation	5.1.1	Verify that the application has defenses against HTTP parameter pollution attacks, particularly if the application framework makes no distinction about the source of request parameters (GET, POST, cookies, headers, or environment variables).
Input Validation	5.1.2	Verify that frameworks protect against mass parameter assignment attacks, or that the application has countermeasures to protect against unsafe parameter assignment, such as marking fields private or similar.
Input Validation	5.1.5	Verify that URL redirects and forwards only allow destinations which appear on an allow list, or show a warning when redirecting to potentially untrusted content.
Sanitization and Sandboxing	5.2.4	Verify 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.
Sanitization and Sandboxing	5.2.5	Verify that the application protects against template injection attacks by ensuring that any user input being included is sanitized or sandboxed.
Sanitization and Sandboxing	5.2.6	Verify that the application protects against SSRF attacks, by validating or sanitizing untrusted data or HTTP file metadata, such as filenames and URL input fields, and uses allow lists of protocols, domains, paths and ports.
Sanitization and Sandboxing	5.2.7	Verify that the application sanitizes, disables, or sandboxes user-supplied Scalable Vector Graphics (SVG) scriptable content, especially as they relate to XSS resulting from inline scripts, and foreignObject.
Sanitization and Sandboxing	5.2.8	Verify that the application sanitizes, disables, or sandboxes user-supplied scriptable or expression template language

		content, such as Markdown, CSS or XSL stylesheets, BBCode, or similar.
Output Encoding and Injection Protection	5.3.3	Verify that context-aware, preferably automated - or at worst, manual - output escaping protects against reflected, stored, and DOM based XSS.
Output Encoding and Injection Protection	5.3.4	Verify that data selection or database queries (e.g. SQL, HQL, ORM, NoSQL) use parameterized queries, ORMs, entity frameworks, or are otherwise protected from database injection attacks.
Output Encoding and Injection Protection	5.3.5	Verify that where parameterized or safer mechanisms are not present, context-specific output encoding is used to protect against injection attacks, such as the use of SQL escaping to protect against SQL injection.
Output Encoding and Injection Protection	5.3.6	Verify that the application protects against JSON injection attacks, JSON eval attacks, and JavaScript expression evaluation.
Output Encoding and Injection Protection	5.3.8	Verify that the application protects against OS command injection and that operating system calls use parameterized OS queries or use contextual command line output encoding.
Output Encoding and Injection Protection	5.3.9	Verify that the application protects against Local File Inclusion (LFI) or Remote File Inclusion (RFI) attacks.
Output Encoding and Injection Protection	5.3.10	Verify that the application protects against XPath injection or XML injection attacks.
Deserialization Prevention	5.5.1	Verify that serialized objects use integrity checks or are encrypted to prevent hostile object creation or data tampering.
Deserialization Prevention	5.5.2	Verify that the application correctly restricts XML parsers to only use the most restrictive configuration possible and to ensure that unsafe features such as resolving external entities are disabled to prevent XML eXternal Entity (XXE) attacks.
Error Handling	7.4.1	Verify that a generic message is shown when an unexpected or security sensitive error occurs, potentially with a unique ID which support personnel can use to investigate.
Client-Side Data Protection	8.2.1	Verify the application sets sufficient anti-caching headers so that sensitive data is not cached in modern browsers.
Client-Side Data Protection	8.2.3	Verify that authenticated data is cleared from client storage, such as the browser DOM, after the client or session is terminated.
Sensitive Private Data	8.3.1	Verify that sensitive data is sent to the server in the HTTP message body or headers, and that query string parameters from any HTTP verb do not contain sensitive data.

Client Communication Security	9.1.1	Verify that TLS is used for all client connectivity, and does not fall back to insecure or unencrypted communications.
Client Communication Security	9.1.2	Verify using up to date TLS testing tools that only strong cipher suites are enabled, with the strongest cipher suites set as preferred.
Client Communication Security	9.1.3	Verify that only the latest recommended versions of the TLS protocol are enabled, such as TLS 1.2 and TLS 1.3. The latest version of the TLS protocol should be the preferred option.
Malicious Code Search	10.2.2	Verify that the application does not ask for unnecessary or excessive permissions to privacy related features or sensors, such as contacts, cameras, microphones, or location.
File Integrity	12.3.1	Verify that user-submitted filename metadata is not used directly by system or framework filesystems and that a URL API is used to protect against path traversal.
File Execution	12.3.3	Verify that user-submitted filename metadata is validated or ignored to prevent the disclosure or execution of remote files via Remote File Inclusion (RFI) or Server-side Request Forgery (SSRF) attacks.
File Execution	12.3.6	Verify that the application does not include and execute functionality from untrusted sources, such as unverified content distribution networks, JavaScript libraries, node npm libraries, or server-side DLLs.
File Download	12.5.1	Verify that the web tier is configured to serve only files with specific file extensions to prevent unintentional information and source code leakage. For example, backup files (e.g. .bak), temporary working files (e.g. .swp), compressed files (.zip, .tar.gz, etc) and other extensions commonly used by editors should be blocked unless required.
File Download	12.5.2	Verify that direct requests to uploaded files will never be executed as HTML/JavaScript content.
Generic Web Service Security	13.1.3	Verify API URLs do not expose sensitive information, such as the API key, session tokens etc.
Generic Web Service Security	13.1.5	Verify that requests containing unexpected or missing content types are rejected with appropriate headers (HTTP response status 406 Unacceptable or 415 Unsupported Media Type).
RESTful Web Service	13.2.2	Verify that JSON schema validation is in place and verified before accepting input.
RESTful Web Service	13.2.3	Verify that RESTful web services that utilize cookies are protected from Cross-Site Request Forgery via the use of at least one or more of the following: double submit cookie pattern, CSRF nonces, or Origin request header checks.

RESTful Web Service	13.2.5	Verify that REST services explicitly check the incoming Content-Type to be the expected one, such as application/xml or application/json.
SOAP Web Service	13.3.1	Verify that XSD schema validation takes place to ensure a properly formed XML document, followed by validation of each input field before any processing of that data takes place.
Dependency	14.2.1	Verify that all components are up to date, preferably using a dependency checker during build or compile time.
Unintended Security Disclosure	14.3.2	Verify that web or application server and application framework debug modes are disabled in production to eliminate debug features, developer consoles, and unintended security disclosures.
Unintended Security Disclosure	14.4.1	Verify that every HTTP response contains a Content-Type header. Also specify a safe character set (e.g., UTF-8, ISO-8859-1) if the content types are text/*, /+xml and application/xml. Content must match with the provided Content-Type header.
Unintended Security Disclosure	14.4.2	Verify that all API responses contain a Content-Disposition: attachment; filename="api.json" header (or other appropriate filename for the content type).
Unintended Security Disclosure	14.4.3	Verify that a Content Security Policy (CSP) response header is in place that helps mitigate impact for XSS attacks like HTML, DOM, JSON, and JavaScript injection vulnerabilities.
Unintended Security Disclosure	14.4.5	Verify that a Strict-Transport-Security header is included on all responses and for all subdomains, such as Strict-Transport-Security: max-age=15724800; includeSubdomains.
Unintended Security Disclosure	14.4.7	Verify that the content of a web application cannot be embedded in a third-party site by default and that embedding of the exact resources is only allowed where necessary by using suitable Content-Security-Policy: frame-ancestors and X-Frame-Options response headers.
HTTP Security Headers	14.5.2	Verify that the supplied Origin header is not used for authentication or access control decisions, as the Origin header can easily be changed by an attacker.
HTTP Security Headers	14.5.3	Verify that the Cross-Origin Resource Sharing (CORS) Access-Control-Allow-Origin header uses a strict allow list of trusted domains and subdomains to match against and does not support the "null" origin.

The “Category” and “#” columns refer to the related OWASP Application Security Verification Standard (ASVS) requirements upon which the listed CASA requirement is based.