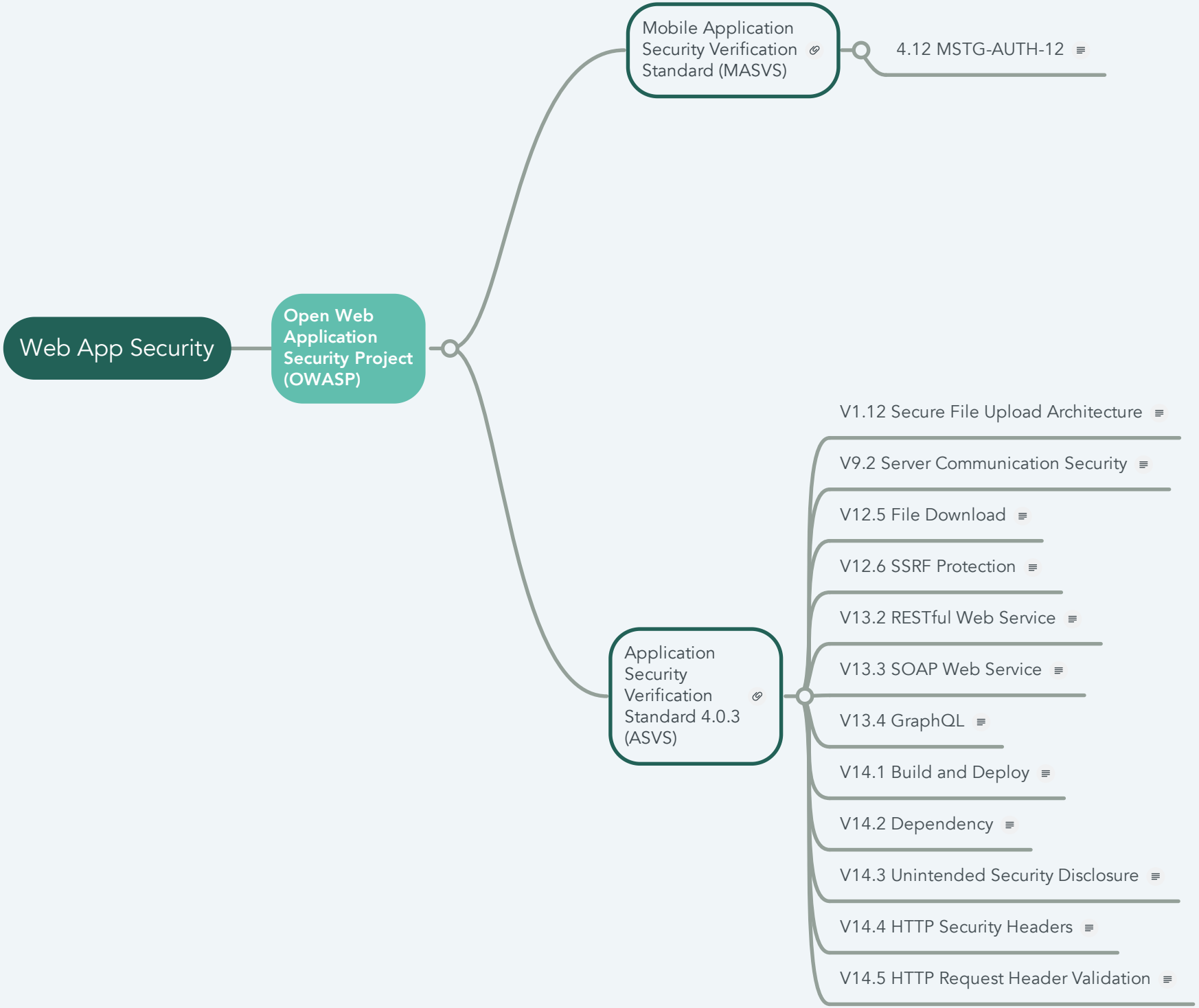
Web App Security
Open Web Application Security Project (OWASP)
Mobile Application Security Verification Standard (MASVS)
4.12 MSTG-AUTH-12
Application Security Verification Standard 4.0.3 (ASVS)
V1.12 Secure File Upload Architecture
V9.2 Server Communication Security
V12.5 File Download
V12.6 SSRF Protection
V13.2 RESTful Web Service
V13.3 SOAP Web Service
V13.4 GraphQL
V14.1 Build and Deploy
V14.2 Dependency
V14.3 Unintended Security Disclosure
V14.4 HTTP Security Headers
V14.5 HTTP Request Header Validation

Web App Security

1. Open Web Application Security Project (OWASP)

1.1. Mobile Application Security Verification Standard (MASVS)

Link: https://github.com/OWASP/owasp-masvs/releases/tag/v1.4.2

1.1.1. 4.12 MSTG-AUTH-12

> Authorization models should be defined and enforced at the remote endpoint.

1.2. Application Security Verification Standard 4.0.3 (ASVS)

Link: https://raw.githubusercontent.com/OWASP/ASVS/v4.0.3/4.0/OWASP%20Application%20Security%20Verification%20Standard%204.0.3-en.pdf

1.2.1. V1.12 Secure File Upload Architecture

> 1.12.2 Verify 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 (CSP) to reduce the risk from XSS vectors or other attacks from the uploaded file.

1.2.2. V9.2 Server Communication Security

> 9.2.5 Verify that backend TLS connection failures are logged.

1.2.3. V12.5 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.
>
> 12.5.2 Verify that direct requests to uploaded files will never be executed as HTML/JavaScript content.

1.2.4. V12.6 SSRF Protection

12.6.1 Verify that the web or application server is configured with an allow list of resources or systems to which the server can send requests or load data/files from.

1.2.5. V13.2 RESTful Web Service

13.2.1 Verify that enabled RESTful HTTP methods are a valid choice for the user or action, such as preventing normal users using DELETE or PUT on protected API or resources.

13.2.2 Verify that JSON schema validation is in place and verified before accepting input.

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.

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.

13.2.6 Verify that the message headers and payload are trustworthy and not modified in transit. Requiring strong encryption for transport (TLS only) may be sufficient in many cases as it provides both confidentiality and integrity protection. Per-message digital signatures can provide additional assurance on top of the transport protections for high-security applications but bring with them additional complexity and risks to weigh against the benefits.

1.2.6. V13.3 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.

13.3.2 Verify that the message payload is signed using WS-Security to ensure reliable transport between client and service.

1.2.7. V13.4 GraphQL

13.4.1 Verify that a query allow list or a combination of depth limiting and amount limiting is used to prevent GraphQL or data layer expression Denial of Service (DoS) as a result of expensive, nested queries. For more advanced scenarios, query cost analysis should be used.

13.4.2 Verify that GraphQL or other data layer authorization logic should be implemented at the business logic layer instead of the GraphQL layer.

V14.1 Build and Deploy

1.2.8. V14.1 Build and Deploy

14.1.3 Verify that server configuration is hardened as per the recommendations of the application server and frameworks in use.

1.2.9. V14.2 Dependency

14.2.3 Verify that if application assets, such as JavaScript libraries, CSS or web fonts, are hosted externally on a Content Delivery Network (CDN) or external provider, Subresource Integrity (SRI) is used to validate the integrity of the asset.

1.2.10. V14.3 Unintended Security Disclosure

14.3.3 Verify that the HTTP headers or any part of the HTTP response do not expose detailed version information of system components.

1.2.11. V14.4 HTTP Security Headers

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.

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).

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.

14.4.4 Verify that all responses contain a X-Content-Type-Options: nosniff header.

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.

14.4.6 Verify that a suitable Referrer-Policy header is included to avoid exposing sensitive information in the URL through the Referer header to untrusted parties.

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.

V14.5 HTTP Request Header Validation

1.2.12. V14.5 HTTP Request Header Validation

14.5.1 Verify that the application server only accepts the HTTP methods in use by the application/API, including pre-flight OPTIONS, and logs/alerts on any requests that are not valid for the application context.

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.

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.