Beagle Security

SaaS application security checklist

A practical checklist for product teams

SaaS applications change constantly, and security needs to keep pace. Use this checklist to review the security of your application across authentication, APIs, business logic, data, dependencies, infrastructure, and the development lifecycle.

Check each item as you validate it.

Authentication & session management

Strong authentication protects everything behind the login layer.

- [ ] **Enforce multi-factor authentication (MFA) for all user accounts, especially admin roles**

  MFA adds another layer of protection when passwords are compromised. Pay particular attention to accounts with administrative privileges.

- [ ] **Use short lived tokens and enforce reauthentication for sensitive actions**

  Limit how long an exposed session or token remains useful, and require users to authenticate again before performing sensitive operations.

- [ ] **Implement secure password policies with breach detection**

  Password requirements should be technically enforced, with mechanisms in place to identify compromised credentials.

- [ ] **Test login flows for credential stuffing, brute force and account enumeration vulnerabilities**

  Test how the application responds to repeated login attempts and whether its responses reveal whether an account exists.

- [ ] **Validate OAuth flows and third party login integrations for token leakage and misconfiguration**

  Review authentication flows beyond your own login system. Incorrect OAuth configuration or exposed tokens can create another path into the application.

API security

APIs are a major part of modern SaaS applications and a critical part of the attack surface.

- [ ] **Inventory every API endpoint - authenticated and unauthenticated, internal and external**

  You need visibility into the complete API surface to know what needs to be secured and tested, including endpoints that aren't publicly exposed.

- [ ] **Enforce authentication on every endpoint; never rely on obscurity**

  An endpoint being undocumented or difficult to discover isn't an access control. Verify that authentication is enforced where it is required.

- [ ] **Implement rate limiting and abuse detection on all public-facing endpoints**

  Public APIs should have controls that limit excessive requests and help identify abusive activity.

- [ ] **Validate and sanitize all input server side; never trust client supplied data**

  Treat data received from the client as untrusted. Validation and sanitization should happen on the server before that data is processed.

- [ ] **Test REST and GraphQL APIs for injection, broken object-level authorization (BOLA), and excessive data exposure**

  API testing should go beyond checking whether endpoints are reachable. Verify that they properly handle authorization, input, and the data they return.

- [ ] **Use automated API security testing as part of the CI/CD pipeline**

  Automated testing helps security checks keep pace with frequent API changes and catches issues before they move further through the development lifecycle.

Business logic & access controls

Security issues aren't always about technical vulnerabilities. They can also come from how features, workflows and permissions behave.

- [ ] **Document the intended workflows and rules for every key feature**

  Clearly defined workflows give your team something concrete to test against and make unexpected behavior easier to identify.

- [ ] **Test multi-step flows for sequence bypass - can a user skip step 2 and go straight to step 3?**

  Don't assume users will follow the intended sequence. Test whether required steps can be skipped or reached out of order.

- [ ] **Validate that privilege levels are enforced not just in the UI, but also on the server side**

  Hiding an option in the interface isn't enough. Verify that the backend also prevents users from performing actions outside their permissions.

- [ ] **Test tenant isolation - can a user in one tenant access data belonging to another?**

  For multi-tenant applications, verify that tenant boundaries are properly enforced and users cannot cross those boundaries to access another customer's data.

- [ ] **Check that free tier users cannot access paid or restricted functionality through direct API calls**

  Test restricted functionality outside the normal UI flow to make sure access controls cannot be bypassed through direct requests.

Data security

SaaS applications often handle sensitive customer data, making proper data protection essential.

- [ ] **Encrypt all sensitive data at rest using AES-256 or an equivalent**

  Sensitive information should remain protected when stored, including data held within the application's underlying storage systems.

- [ ] **Enforce HTTPS across all endpoints; check for mixed content or certificate issues**

  Secure connections should be consistently enforced across the application. Check for configuration issues that could expose traffic or weaken transport security.

- [ ] **Implement data minimization - collect and retain only what the product requires**

  The less unnecessary sensitive data your application holds, the less data there is to expose if something goes wrong.

- [ ] **Audit logging for all access to sensitive customer data**

  Maintain a record of access to sensitive information so activity can be reviewed when needed

- [ ] **Test for sensitive data exposure in API responses, error messages and logs**

  Sensitive information can appear in places developers don't expect. Review responses, errors, and logs for data that shouldn't be exposed.

Dependency & supply chain security

Third party libraries and integrations can introduce security risks if they're not properly managed.

- [ ] **Maintain an updated inventory of all third party libraries and dependencies**

  Know which external components your application relies on and keep that inventory current as dependencies change.

- [ ] **Run automated software composition analysis (SCA) scans on every build**

  SCA helps identify security issues in third-party components as part of the build process rather than after deployment.

- [ ] **Monitor for known CVEs (Common vulnerabilities & exposure) in dependencies and patch promptly**

  Track known vulnerabilities affecting your dependencies and address vulnerable components without unnecessary delay.

- [ ] **Evaluate the security posture of third party APIs and integrations before adoption**

  An external service becomes part of your application's attack surface once you integrate it. Assess its security before relying on it

- [ ] **Review permissions granted to third party OAuth applications**

  Third-party applications can receive access to your environment through OAuth. Review what they can access and whether those permissions are necessary

Infrastructure & configuration

Small configurations can create significant security exposure.

- [ ] **Apply least privilege to all service accounts, IAM roles and database access**

  Give accounts and services only the access they need to perform their intended functions.

- [ ] **Audit cloud storage permission regularly - no public S3 buckets or equivalent**

  Review storage permissions regularly and look for configurations that could unintentionally expose data publicly.

- [ ] **Disable unused ports, services and endpoints**

  Every unnecessary service or exposed endpoint can add to the attack surface. Remove what the application no longer needs.

- [ ] **Enforce secrets management - no API keys or credentials in source code or environment accessible in repositories**

  Keep credentials and API keys out of places where they can be accidentally exposed or committed to repositories.

- [ ] **Run infrastructure level security scans alongside application level testing**

  Application security testing alone doesn't cover the entire environment. Include infrastructure-level checks alongside application testing.

Security testing in the development cycle

A one time security test isn't enough for an application that changes continuously.

- [ ] **Run SAST in the IDE and on every pull request to catch vulnerabilities at the code level**

  Find security issues as code is being written and reviewed rather than waiting until the application is deployed.

- [ ] **Run automated DAST and API security testing on every deployment to staging**

  Test deployed applications as they change so new security issues can be identified during the development process.

- [ ] **Include authenticated testing - scanners which only test unauthenticated endpoints miss the majority of SaaS vulnerabilities**

  Testing only the unauthenticated surface leaves functionality behind the login largely unchecked. Include authenticated areas in security testing.

- [ ] **Schedule regular automated pentests against production, not just staging**

  Production can differ from staging in meaningful ways. Regular testing helps identify issues in the environment customers actually use.

- [ ] **Include business logic testing explicitly - not as an afterthought to pattern-based scanning**

  Pattern-based vulnerability scanning doesn't fully capture how application features behave. Test the application's actual workflows and rules as part of security testing.

Final check

- [ ] Have all applicable items been reviewed?
- [ ] Have identified security issues been addressed or tracked?

Ready to put your checklist into action?

Beagle Security continuously tests your web applications and APIs, helping your team find, validate, and prioritize security risks as your application evolves.

Start testing with Beagle Security here.