Software Development

H2O.ai

May 31, 2023

Table of Contents

Software Development

The organization designs and builds AI/ML software with security and privacy as design principles.

H2O.ai Software Development Policy

1.0 Purpose

The purpose of the Systems Development Life Cycle (SDLC) policy is to describe the requirements for developing and/or implementing software and systems at the H2O.ai and to ensure that all development work is compliant as it relates to any and all regulatory, statutory, and/or contractual guidelines. This document establishes guidelines for the development of software and systems that are required to be applied to all developments to ensure their maintainability, transparency, traceability, and security.

This policy provides for secure and privacy-aware methodologies during the design and implementation of H2O.ai products and services. It incorporates best practices in software and operations design and implementation processes with security, privacy and vulnerability reviews throughout their lifecycle so that implementation ensures that H2O.ai products are only used as intended and are not vulnerable to attack or abuse. The goal of the H2O.ai software practices is to maintain the confidentiality, integrity and availability of information resources in order to enable secure and successful business operations.

1.1 Scope

This policy covers all development activities carried out by the engineering team during the stages of SDLC: planning, design, implementation, and maintenance of H2O.ai products and services.

3.0 Policy

3.1 Basics

3.1.1 Segregation of Environments

Software development environments are segregated as follows:

- Dedicated development environment
- Dedicated test environment
- Dedicated production environment

The test environment can be shared between development and test.

3.1.2 Issue management

All product feature requests and issues are tracked in the following issue management systems:

- GitHub
- JIRA

All non-engineering issues must contain a traceability reference to origin (e.g., support system, product management system).

3.1.3 Source code version control

H2O.ai uses GitHub as the primary code version control system.

Source code version system is used to manage product source code, product documentation, and infrastructure

code (e.g., definitions of production, test environments).

3.1.4 License Management

H2O.ai Engineering and Product teams follows the industry standard for products and services license management:

- Every product and Services clearly states out "End User License Agreement"
- Product and Services Licenses management process is automated and documented with separation of ownership from Engineering and Sales by Sales OPS.
- Licenses are tracked and renewed.
- Licenses of dependencies used by products are tracked and periodically reviewed.
- A list of approved FOSS licenses types are allowed in development, any third-party licenses that is not part of the approved list of FOSS licenses are flagged as part fo PR process and escalated to engineering leadership for consideration. Approved FOSS licenses types are annually reviewed and communicated to the wider engineering team.

3.1.5 Data Privacy

H2O.ai follows secure coding practices and industry standard data privacy policy that is clearly explained and available to customers and users on our website. The policy outlines the following:

- Data collection process.
- Limit and protect the information.
- Change management — monitors, logs, and reports on anomalies.
- Data Loss - Protect data loss by securing the customer data at various stages (rest, transmission)
- Data masking — Anonymizes data via encryption/hashing including product logs
- Data protection — Ensures data integrity and confidentiality through change control reconciliation

3.2 Product Planning, Requirements and Analysis Phase

Product requirements are gathered in this phase. This phase is spearheaded by the product manager, along with other stakeholders.

After the
requirements are gathered, they are analyzed for validity and the possibility of incorporating the requirements in the system to be developed is also studied.

The product planning and request analysis should consider security implications and necessary security requirements (e.g., penetration testing).

The product plan is maintained by the product team in a dedicated system.

The product plan contains references to issues (e.g., epics) created in source code version control system.

All new features require a Product Requirements Document (PRD) provided by the H2O.ai product management team .

3.2 Design and Architecture Process

As part of design process H2O.ai engineering team and product team discuss PRDs, architecture, security and acceptable level of quality. As result the engineering team prepares technical specification considering aspects including component architecture, deployment architecture, data flows, database schemas, APIs, UX, and scope definition (e.g., what will NOT be implemented).

The design should include security impact and consider the following security design aspects to achieve high-quality and secure products:

- Data security
    - Data at rest

    - Data in transit
- System security
    - Access security
    - Storage security
    - Communication security
- Feature security
    - Misuse of features
    - Attack vectors
- Least privileges principle

The technical specification is shared with the Engineering Lead, Product Manager, Security Team, and selected team members for their review, comments and updates.

The specification needs to be approved by all requested participants.

3.3 Development Process

H2O.ai engineering team follows the following branching strategy.

Development is performed in branches. Each repository maintains main branch, development branch. It is permitted to use a single branch for both main and development branches. Each repository contains short-lived branches to perform development of features and bug fixes.

3.2.1 Feature Development

A new feature branch is created from development branch for the feature. The name of feature branch contains identification of author and ticket number (e.g., mm/dev/1234_mars_lander)

All development for the feature is done on the feature branch.

When the feature is complete, a pull request is made and submitted for code review.

3.2.2 Pull Requests

Before a Pull Request (PR) can be merged into the development branch, it must be reviewed by another member(s) of the team.

The reviewer must not be the same person who created a pull request.

The following things will be evaluated on a pull request review:

- Does the code follow H2O.ai coding style and naming conventions?
- Does the code pass the code analysis tool?
- Are there tests associated with the feature?
- Is the code readable and/or properly commented?
- Are there any obvious logic flaws or security holes?
- Is there any debug code left behind?
- Is audit logging properly implemented?
- Is the code reusable? (Where applicable)
- Are the tests within the pull request passed?
- Identify any security vulnerabilities.
- Identify third-party dependencies with not-allowed licenses.

Once the pull request is reviewed, it can be:

- Approved and merged into development branch by author of the pull request.
- Rejected with changes
    - Reviewer comments on what the necessary changes are.

After a pull request is reviewed, approved and merged, the associated feature branch may be deleted. PRs that have been rejected for changes will not result in a new pull request being created. The engineering team

member makes the required changes, commits to the associated feature branch and comment on the PR that the changes are completed. The reviewer will then verify the changes are correct and appropriate. If the changes pass review, the PR will be approved and merged. The feature branch may be deleted at this point.

3.2.3 Releases

When the Develop branch contains the required features for a release, a release branch is forked off of develop and labeled with the correct version number. Once this branch is created, no new features will be added to the release. Bug fixes and other release-oriented tasks may go into this branch.

Once the release is stable and ready to ship, the release branch is merged into the Main branch via a pull request. This provides one last opportunity to review the contents of the upcoming release. Also, the release branch should be merged back into Develop as it may contain bug fixes. Once the release is merged into Main, a tag is taken and an official release is created. This is what will be deployed to production. At this point, the release branch may be deleted.

The release produces bill of materials including:

- list of artifacts
- list of test reports
- list of licenses used by the product dependencies
- vulnerability report for all dependencies

3.2.4 Hotfixes and Special Patches

Occasionally, a bug is discovered in production that requires immediate patching. When this happens, a hotfix branch is created. The hotfix branch is forked from main or the appropriate release version of the branch. This is the only branch that should fork from main. After the fix has been made, a PR is created to merge the fix back into main. The Standard PR review process shall apply. The hotfix branch must also be merged into Develop to prevent the bug from reappearing in a future release. Once the hotfix branch is merged, it may be deleted. An official release is not necessary for hotfixes and the commit log will show the merging of the hotfix into the main code streams.

Appropriate test plan is executed prior to releasing the HotFix. The communication of the hotfix follows the set out process.

3.2.5 Branching Strategy Summary

The overall flow of the branching strategy is:

1. A develop branch is created from main
2. A release branch is created from develop
3. Feature branches are created from develop
4. When a feature is complete it is merged into the develop branch
5. When the release branch is done, it is merged into develop and main
6. If an issue in the main/production is detected, a hotfix branch is created from master
7. Once the hotfix is complete, it is merged to both develop and main

3.2.6 Code quality

The following high level tasks are performed to achieve the needed code quality of each products.

- Code style
- Code linting
- Reasonable code documentation.
- Internal product documentation in Confluence.
- Code review.
- Unit Testing.

3.2.7 Change security

Each code change must contain well identified author associated with H2O.ai.

Personal accounts cannot be used for code changes.

3.2.8 Code security

The static analysis of code should be performed for critical components.

3.2.9 Service Vulnerability Reviews

Products and services released by H2O.ai require regular reviews for vulnerabilities and attacks.

- External scan and penetration tests will be performed annually.
- Internal scan includes dependency vulnerabilities scan and license scan
  - No GPL-like license in production part of software.

3.2.10 Error Handling and Logging

H2O.ai adheres to the following practices for error handling and logging:

- Sensitive information must not be disclosed in error responses.
- Debug or stack trace information will never be displayed on production systems.
- All logging should be implemented on a trusted system.
- Access to logs should be restricted to specific personnel.
- Passwords and other credentials must never appear in logs.
- All system events must be logged.

3.2.11 Data Protection

H2O.ai adheres to the following practices for data protection:

- Implement least privilege. User access should be restricted to only the functionality and data that is required based on job needs.
- All temporary copies of data should be removed once processing is complete.
- All access credentials must be encrypted using a one-way salted hash.
- All transmission of data must be over secure channels.
- Application comments should be removed from client-side code.
- Caching should be disabled on pages containing sensitive information.
- Production data shall not be used in non-production systems or environments. Where the use of production data is required, restrict access to only authorized users, and data must be masked, in addition to having approval in place for the exception or deviation of the standard process.

3.2.12 Communication Security

H2O.ai adheres to the following practices for communication security and as described in the Key Management and Cryptography Policy:

- All transmission of data must be done over an encrypted channel using Transport Layer Security (TLS).
- Failed TLS connections must not fall back to an insecure connection.

3.3 Testing

Products developed but H2O.ai adhere to the following test strategy

1. Unit tests for each module in the product.
2. Integration tests for the product.
3. End-end testing along with performance, load and stress tests.

All of the above will be documented in test plans and the results will be stored in a test repository.

3.4 Deployment and Delivery

To deploy a new version of product, the security and QA team must provide approval. The DevOps team deploys the new version to production environment.

3.5 Monitoring and Maintenance

3.5.1 Dependencies security

The artifacts of released products are monitored for dependency vulnerabilities. If a dependency vulnerability is observed, it is evaluated and processed according to the Incident Management policy.

3.5.3 Backward Compatibility and Product End of Life

H2O.ai follows the following software development processes. to ensure software releases are backward compatible and also the process around when software is not guaranteed to be backward compatible.

- Software compatibility matrix
- Backward compatibility testing
- Documentation communication of backward compatibility matrix
- Product support and End of Life Matrix