

Secure Development and System Acquisition

I. Purpose

This policy ensures that development environments are secure and encourages the use of secure coding and development practices. Security needs to be considered at all stages of the development lifecycle from design through to implementation.

Specific coding languages and development tools have different vulnerabilities and require different “hardening” techniques; it is important that these are identified and developers are made aware of their responsibilities to follow them.

II. Scope

These standards apply to all persons involved in the acquisition, development and maintenance of Nested Knowledge's IT environment and are applicable to all Nested Knowledge systems, including applications/systems developed for the Internet.

III. Secure Development Policy

Restrictions on Changes to Software Packages

Modifications to software packages need to be discouraged, limited to necessary changes and all changes should be strictly controlled. Vendor supplied software packages are designed for the mass-market and are generally not designed for organizations making their own changes to them. Usually, the ability to make such changes is locked out by the vendor and customization limited to within the package.

Open-Source Software

Where open-source software is used, it is generally possible for changes to be made by the organization, however, this should be restricted and controlled to ensure that the changes made do not have an adverse impact on the internal integrity or security of the software.

Secure Development Environment

Development environments need to be protected against malicious or accidental development and update of code that may create vulnerabilities or compromise confidentiality. Production data is only used in development when a data-dependent bug arises; in these cases, only the relevant data is made available to the development environment, and it is short-lived, being made inaccessible once the issue is resolved.

Development environments are independent from production, lacking configuration, credentials, and network access to Production services and data.

Outsourced Development

Nested Knowledge source code is only accessible to and developed by internal employees.

Supplier Adherence to SDLC

Nested Knowledge employees develop all application code. External platforms (e.g. AWS) may provide architecture, orchestrate deployment of, and execute application code. External libraries (e.g. OSS) are consumed by application code as modules. External providers offer data to code, most typically in the form of an API. External platforms, providers, and libraries are verified with the following:

- A review of technical documentation
 - Does the platform/library maintain a release history?
 - Does the platform/library publish their development strategies? Does it conform to typical SDLC standards?
 - Does the solution advertise itself as production ready?
- A review of update / release history
 - Does the supplier make regular updates for security patches
 - Automate scanning for existing vulnerabilities using NPM vulnerability scanning, the pypi Safety DB, and GitHub vulnerability monitoring.
 - Does the supplier actively maintain the solution?
 - Does the supplier follow a standardized release versioning system? (e.g. SemVer)
 - Are there reports of major version changes that weren't reported as such?
- Does the solution include a public issue tracker?
 - Are issues (bugs & security vulnerabilities) addressed quickly?
- Does the solution have a policy and/or history of disclosing vulnerabilities and breaches?
- When available, code & version history review.
 - Do all commits to the repository leave it in deployable condition?
 - Are all commits reviewed & verified by a maintainer / peer?
 - Is code well structured & organized? Does it compile? Does it have/pass a linter?
 - Are new features / patches developed on branches or external to the main deployment branch?
 - Does the solution minimize & verify its dependencies?

Per these criteria, the supplier is assessed for risk in inclusion as a dependency to application code.

Testing Procedures

Our software is tested & verified in two environments, development & production.

In development, code is tested via the following methods:

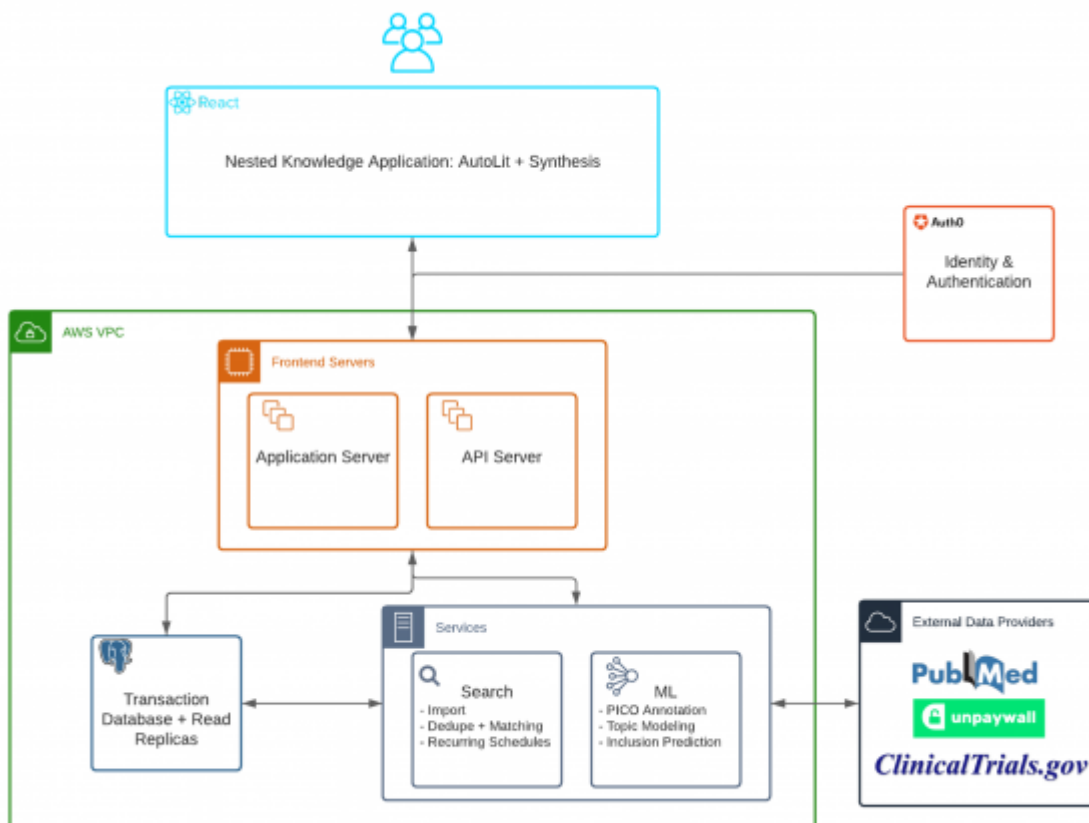
- Automated unit tests, integration tests, functional tests, and linters
 - Unit tests cover shared modules and core functions (e.g. authorization logic)

- Functional tests cover our most critical features with complex input/output behaviors (e.g. literature search import)
- Client and frontend servers are covered by linters to identify common errors and improve code quality
- All code changes must pass tests and linters prior to acceptance.
- Peer review & testing
 - The reviewing developer reads code, identifying corner cases, assessing security compliance, and verifying correctness
 - A reviewing developer and product manager must verify functionality before any code change is accepted.

In production, new changes in the release are verified by a developer and product manager upon release. If a feature does not function as intended, the release is immediately rolled back and only proceeds once a patch is reviewed and accepted.

System Design and Architecture

The Nested Knowledge client application is served by an application server and hydrated by an API server; both of these servers run behind a load balancer. The API server communicates with the Search and ML backend services as well as the database. Certain functions of backend services communicate with external (public) APIs. The frontend servers, services, and database all run in a Virtual Private Cloud (VPC) for network isolation. The frontend load balancers are exposed to receive requests from the public internet and are protected with a WAF using AWS's Core Ruleset. The client and server applications communicate with an external service, Auth0, for authentication; all communications with Auth0 are encrypted.



Security Requirements

General

All application data are stored and processed inside the VPC. Data leaving the VPC are either encrypted to the authenticated & authorized client, or, in the case of external providers contain minimal & nonsensitive information (e.g. DOIs for unpaywall, search strings for PubMed). HTTP traffic is served over HTTPS (redirect required); email traffic (through AWS SES) is encrypted from application to sender (TSL), and sender to receiver (TSL). Within the VPC, communications between the database and all services are encrypted.

Authentication

Nested Knowledge does not manage user passwords or authentication (handled by [Auth0](#) and Auth0 Lock). All communications with Auth0 from the client are encrypted (TSL), ensuring passwords are not communicated in plain text. Passwords stored by Auth0 are similarly salted & encrypted (bcrypt). Communications relayed by the client are similarly encrypted & RSA signed.

Software Applications on NK-Owned Devices

Our [application profile](#) in the Business Continuity Plan describes the criticality of software applications used by Nested Knowledge employees on employee-owned devices.

Vulnerability and Patch Management

Nested Knowledge uses automated scanning to identify potential vulnerabilities in our operating environment and software dependencies. The risk of a vulnerability is assessed using the [CVSS framework](#).

Our cloud services provider (AWS) provides automated vulnerability reports on operating systems & databases running the application. Application vulnerability scanning is performed by SecurityScorecard on an ongoing basis. Alerts derived from these reports are addressed by release engineers:

- The applicability & risk of the vulnerability is verified and added to our issue tracker. If a vulnerability is deemed nonapplicable or low severity, it is prioritized amongst other development initiatives. If medium or high severity, the vulnerability is given highest priority, to be addressed by an engineer with next availability.
- The viability of the recommended patch is assessed.
 - Bug fix / minor version patches may be applied with minimal testing.
 - Larger patches requiring updates to our application must pass through our code review process. Any high risk vulnerability is given highest priority in our queue, including dropping actively developed features to procure compatibility for the patch.

- The patch is applied during the next scheduled release window. If a vulnerability is deemed high severity, it triggers an immediate release upon review completion.

Our dependency management tools (NPM, pip, apt-get) provide automated vulnerability reporting & patch alerts. Every build in our development environment produces a vulnerability report among dependencies. Alerts derived from these reports are addressed by developers.

- The applicability & risk of the vulnerability is verified. If a vulnerability is deemed nonapplicable or low severity, it is added to our issue tracker. If applicable or medium or high severity, the vulnerability is assigned to an engineer immediately.
- The viability of the recommended patch is assessed. Developers will modify the application code as needed to accommodate the patch (e.g. updating to a new API or changing a configuration). All changes are tested and passed through code review.
 - If the vulnerability lacks a patch, the team may:
 - Contribute an upstream patch, for open source dependencies.
 - Mitigate code paths triggering the vulnerability and begin researching alternative dependencies.
 - Replace or remove the dependency, with application code changes
- The dependency and/or code change is deployed during the next scheduled release window. If a vulnerability is deemed high severity, it triggers an immediate release upon review completion.

Revision History

This policy will be updated at least on an annual basis or when a significant change occurs.

Author	Date of Revision/Review	Comments/Description
K. Cowie	01/24/2023	Reviewed
K. Holub	03/30/2023	Updating vulnerability scanning to include SecurityScorecard
K. Kallmes	11/19/2021	Draft approved
P. Olaniran	9/22/2022	Minor Revisions

[Return to Policies](#)

From:

<https://wiki.nested-knowledge.com/> - **Nested Knowledge**

Permanent link:

<https://wiki.nested-knowledge.com/doku.php?id=wiki:policies:dev>

Last update: **2024/01/25 23:02**