Secure Software Development and Product Security

`2021.1`

MiCT development team follows the latest security best practices when developing software, and automates security testing throughout development lifecycle whenever possible.

Security is integrated into all phases of MiCT product development lifecycle, including:

- Secure Design:
  - App Risk classification
  - Security req definition
  - Secure application design / RFC
  - Threat modeling
  - App data flow analysis
- Secure Development and Testing:
  - Secure coding guidelines
  - Peer review
  - Security testing that includes:
    - Linting with security rules
    - Open source security analysis
    - Static secure code analysis
    - Dynamic security analysis
    - Penetration testing
  - Responsible vulnerability disclosure / bug bounty program
- Remediation:
  - Follows defined vulnerability management lifecycle
  - Ensures no high risk security vulnerability is in production

Details about the MiCT software application architecture and security are documented on the product development / engineering wiki.

Policy Statements

MiCT policy requires that:

(a) MiCT software engineering and product development is required to follow security best practices. Product should be "Secure by Design" and "Secure by Default".

(b) Quality assurance activities must be performed. This may include

- peer code reviews prior to merging new code into the main development branch (e.g. master branch); and
- thorough product testing before releasing to production (e.g. unit testing and integration testing).

(c) Risk assessment activities (i.e. threat modeling) must be performed for a new product or major changes to an existing product.

(d) Security requirements must be defined, tracked, and implemented.

(e) Security analysis must be performed for any open source software and/or third-party components and dependencies included in MiCT software products.

(f) Static application security testing (SAST) must be performed throughout development and prior to each release.

(g) Dynamic application security testing (DAST) must be performed prior to each release.

(h) All critical or high severity security findings must be remediated prior to each release.

(i) All critical or high severity vulnerabilities discovered post release must be remediated in the next release or within 30 days, whichever is sooner.

(j) Any exception to the remediation of a finding must be documented and approved by the security team.

Controls and Procedures

Software Development Process

Overview Software development at MiCT follows a release strategy that provides traceability for production software changes. Features, enhancements, and bugs are written up as Issues in Github. An engineer on a small team proposes changes necessary and creates a review for the team (). Continuous integration (https://localhost) kicks off unit and functional tests which pass before changes are merged into the repository. Once the review is complete, the changes are now deployed to the development environment where regression and end-to-end tests are run before the new code replaces the existing in-service code (test then deploy model). Small teams can decide to follow a source-control branching strategy that makes sense: git-flow, github flow.

MiCT practices continuous delivery of code into production through multiple environments: development, testing, production. The deploy process and infrastructure roll-out are written as code (using technologies such as Terraform and AWS Cloudformation) and managed under source control.

MiCT's multiple lower environments (dev, test) provide an ecosystem of sample data sets that exercise the application and services when test automation is

run. The test environment is where the system is stressed for performance and scalability. Performance and scalability changes are driven by metric data captured through monitoring and logging (metrics before and after change – typically captured as part of the issue description/writeup).

Deployments to production are gated by change control process where an issue is opened which identify what is new/changed (Github). Sign-offs are recorded by development, testing, security, and product management. Production roll-outs happen on a regular basis without impact to service. This continuous process allows for security updates to roll out regularly and with urgency. If there is impact to production, a rollback is performed to restore service and whatever caused the problem is reverted from source. This restarts the re-proposal approval process of source changes. This process keeps the set of differences between the development environment and the production environment as low as possible.

In the continuous delivery mindset, features are not released by the deployment of code into production, instead features are enabled in production at the appropriate time (dark launching). Feature toggle enablement in production is gated by a change control ticket (Github) that follows the software roll-out approval process. Feature toggle enablement in production can have a few more dependencies than code. Those dependencies include things like external documentation, early access programs, and internal playbooks for supporting the feature.

Secure Development Standards Traceability of code changes allow for our software to be a living entity. Our current system for documenting changes is Github. Every commit and/or Pull-Request, should have a Github supplied that describes contextually why this change is necessary and reasonable. These artifacts over time allow for one to trace the lineage of why our production software and services change over time.

All MiCT `git` repositories have a company standard configuration from a perspective. This standard is a guideline and can be relaxed, but socialize when those exceptions are needed. One example of an exception, is the `wiki` repository, as editing a wiki and always requiring a PR in this setting slows down 'flow'.

- Code: <#>
- Repo settings: <#>
- Build: <#>

NOTE: The Sandbox project (and repos) do not follow this standard. And certain projects might be excluded (e.g. `wiki`).

Developers follow the branch strategy and code review process below:

1. All development uses feature branches based on the main branch used for the current release. Any changes required for a new feature or defect fix are committed to that feature branch.
    - These changes must be covered under 1) a unit test where possible, or 2) integration tests.
    - Integration tests are *required* if unit tests cannot reliably exercise all facets of the change.
2. Developers are strongly encouraged to follow the commit message conventions suggested by GitHub.
    - Commit messages should be wrapped to 72 characters.
    - Commit messages should be written in the present tense. This convention matches up with commit messages generated by commands like git merge and git revert.
    - Additionally, the commit messages should start with the Github Issue ID when applicable.
3. Once the feature and corresponding tests are complete, a pull request (PR) will be created using the web interface. The pull request should indicate which feature or defect is being addressed and should provide a high-level description of the changes made.
4. Code reviews are performed as part of the pull request procedure. Once a change is ready for review, the author(s) will notify other engineers using an appropriate mechanism, typically by adding reviewers as PR approvers.
    - Other engineers will review the changes, using the guidelines above.
    - Engineers should note all potential issues with the code; it is the responsibility of the author(s) to address those issues or explain why they are not applicable.
    - If changes/commits are made to a PR, it should reset previous approvals and require review and approvals again before the PR can be merged.
    - Once the review process finishes, each reviewer should approve the PR, at which point the original author(s) may merge their change into the main branch (i.e. master).
    - PR can only be merged with at least one approval from a reviewer other than the author.
5. If the feature or defect interacts with sensitive data, or controls access to sensitive data, or contains security/risky infrastructure changes to the target environment, the code changes must be reviewed by the Security team before the feature is marked as complete.
    - This review must include a security analysis for potential vulnerabilities such as those listed in the OWASP Top 10.
    - This review must also verify that any actions performed by authenticated users will generate appropriate audit log entries.

Release Strategy Features, enhancements, and bugs are written up as issues (Github). An engineer on a small team proposes changes necessary and creates a review for the team (). Continuous integration (https://localhost) kicks off unit and functional tests which pass before changes are merged into the repository. Once the review is complete, the changes are now deployed to the development environment where regression and end-to-end tests are run before the new code replaces the existing in-service code (test then deploy model). Small teams can decide to follow a source-control branching strategy that makes sense: git-flow, github flow.

MiCT practices continuous delivery of code into production through multiple environments: development, testing, production. The deploy process and infrastructure roll-out are written as code (Terraform) and managed under source control.

MiCT's multiple lower environments (dev, test) provide an ecosystem of sample data sets that can be used to exercise the application and services when test automation is run. The test environment is where the system is stressed for performance and scalability. Performance and scalability changes are driven by metric data captured through monitoring and logging (metrics before and after change - typically captured as part of the issue description/writeup).

Deployments to production are gated by change management process where an issue is opened which identify what is new/changed (Github). Sign-offs are recorded by development, testing, security, and product management. Production roll-outs happen on a regular basis without impact to service. This continuous process allows for security updates to roll out regularly and with urgency. If there is impact to production, a rollback is performed to restore service and whatever caused the problem is reverted from source. This restarts the re-proposal approval process of source changes.

This process keeps set of differences between the development environment and the production environment as low as possible.

Features may be released via code deployments or features may be enabled in production at an appropriate time (dark launching). Feature toggle enablement in production is gated by the same change management ticket (Github) that follows the software roll-out approval process. Feature toggle enablement in production can have a few more dependencies than code. Those dependencies include things like external documentation, early access programs, and internal playbooks for supporting the feature.

Detailed process and procedures for code promotion and production release: See Configuration and Change Management.

Source Code Management

MiCT development/engineering team uses for source code management. Access to and its configuration standards include:

- All developers must authenticate to gain access to and code repos hosted on according to standards and procedures defined in the Access Policy:
  - Access control to the web interface must be enabled, via SSO and/or MFA if applicable
  - SSH public/private key access may be used for command line or `git` access to the code repos
- All code repos in follow these configuration standards:
  - All repos must have an owner identified and listed
  - All repos are by default `private`
- Certain branch restrictions are enabled, including:
  - The `master` branch cannot be rebased
  - Restrict direct commits into `master`
  - Restrict history rewrites of `master`
  - Restrict deletion of `master`
- Certain pull request (PR) requirements are enforced before merging, including:
  - Must have at least 1 review approval to merge
  - Must have at least 1 successful build to merge
  - all PR tasks must be completed

High Level Application Security Requirements

All MiCT software must be developed to include the following general application security principles and requirements. Web applications must also protect itself against the OWASP Top 10 vulnerabilities.

1. Protect sensitive customer data such as PHI, PII and account passwords. Encrypt data stored (at rest).
2. Secure data in transit and customer communications via TLS.
3. Provision strong access control (authentication and authorization). Prevent and report unauthorized access.
4. Log all transactions and activities to be able to tell who did what, when, where, and how. Mask or remove sensitive data in logs.
5. Implement client security at application endpoints (e.g. browser, mobile app).
6. Communicate securely across application endpoints and between service consumers/producers.

7. Use secure defaults to ensure security when in all error conditions.
8. Check and maintain the security of all third party and open source libraries/components/dependencies.
9. Validate all data inputs; encode data outputs when appropriate.
10. Deploy and configure applications securely to production.
11. Perform regular vulnerability analysis and apply security patches promptly.
12. Secure privileged access to production environments and ensure ongoing application monitoring.

All software code must complete a set of security scans/testing prior to being deployed to production, including open source dependency scanning, static and dynamic application security testing, as well as periodic penetration testing.

Pre-production testing is performed with nonproduction data in nonproduction environments. Health checks are performed regularly or automated in production.

Software vulnerability identified through any of the above processes shall be reported and tracked following MiCT Vulnerability Management process as defined in the Vulnerability Management Policy and Procedures.

Secure Design and Application Threat Modeling

MiCT Security Team in collaboration with development team performs full Application Threat Modeling and Risk Assessment on per-application basis using a custom approach that relies on industry standards and best practices.

Major application updates are captured via an **RFC** process. The RFC template includes **Security Consideration** as a required section. This section is used to document abuse cases including:

- risks identified,
- attack vectors, and
- mitigating controls.

Each RFC is required to capture sufficient details of the feature/component to be developed, including use cases, motivation and outcome, and the following design details as applicable:

- authentication/authorization mechanisms,
- network communications,
- data encryption,
- cloud services used,
- logging/auditing,
- data flow diagram/description,
- edge cases, drawbacks, and alternatives.

The RFC must be approved prior to implementation. Security team is included in RFC reviews via the pull request process.

Platform Design and DevOps Security Details Documentation on the MiCT Engineering Wiki may include additional security specifications as well as the security design and implementation details of the MiCT Platform and its supporting operations.

Access Control of the Application (Identification, Authentication, Authorization, Accounting)

MiCT external software application that is customer facing with access to customer specific data, including sensitive information such as PII and ePHI, implements strong access control, covering the Identification, Authentication, Authorization, and Accounting/Auditing (IAAA) of access and user activity.

The implementation ensures that

- the user requesting access is the one claimed (Identification and Authentication);
- only users authorized to access specific data (such as ePHI) are allowed to (Authorization); and
- their access activities are logs (Accounting/Auditing) according to the MiCT auditing standards.

The current implementation leverages AWS Cognito for user identity management and access.

The backend platform implements granular Attribute-Based Access Control (ABAC) for granting access to specific services and data based on the attribute(s) of a principal (i.e. user requesting access – an attribute could be the role or group membership or organization the user belongs to) and the attribute(s) of the requested resource (i.e. data or service – an attribute could be the project this data belongs to).

More implementation details are documented on the internal Engineering wiki.

Free and Open Source Software (FOSS) Security

MiCT security and development team implemented a process to

- Inventory all software dependencies;
- Scan software dependencies for known security vulnerability;
- Fix any and all high risk findings prior to production; and
- Review and identify licensing issues of the 3rd party software and libraries.

The current tool in use is Snyk. Snyk supports Node.js, Ruby, and Java. Documentation can be found at the Snyk website.

Snyk also enables automatic license analysis for dependencies, and this functionality is embedded into the `securityScan()` https://localhost pipeline step.

Snyk can be leveraged in local development environment to scan locally and/or at every build for dev to include dev dependencies.

Static Application Security Testing (SAST)

Tools:

- The Lint and GitLeaks Platform - static and dynamic code analysis. Together with the FOSS security tool, the SAST testing was made available to use via the automated `security-scan` docker image, as well as integrated into MiCT https://localhost pipeline library via the `securityScan()` step
- Manual code security reviews are periodically performed by the Security team as follows:
  - Periodic review as part of weekly activities
  - Every pull request which includes security team as approvers
  - By request - Github issue created on the Security project/board

Dynamic Application Security Testing (DAST)

Dynamic testing is available by request to the Security team and is performed using OWASP ZAP. Security team is currently looking into automation of this type of scanning.

Additionally, manual baseline dynamic testing is available by request to the Security team.

Integration of Dynamic testing into automatic tests Upon availability of automatic UI tests for applications being developed internally, an extension based on OWASP ZAP is used to transparently work with any UI tests available and automatically add basic dynamic tests into build pipeline.

Penetration Testing

External Penetration Testing An external penetration testing is performed at least once a year by a qualified security researcher / ethical hacker on the security team internally and/or with an external security consulting firm.

Responsible Disclore and Bug Bounty Program

Public Vulnerability Disclosure / Bug Bounty Program All technology contains bugs, including both functional defects and security vulnerability.

MiCT uses a third party platform to maintain a public vulnerability disclosure / bug bounty program. This program takes advantage of crowd-sourced security researchers to perform penetration testing of MiCT's platform and its public facing resources.

The current program provider is HackerOne.

Outsourced Software Development

MiCT requires all outsourced software development to follow the same rigor and process as internal engineering. Outsourced developers must develop in our secure environment, accept and follow our security policies and procedures, and comply with the same secure coding standards, including:

- Receive regular OWASP or equivalent secure coding training.
- Follow the same source control, code review, and security code scanning procedures as defined.
- Install endpoint compliance agent that checks to make sure firewall, encryption, patching, password policy, screensaver password, and other required protection is properly configured.

Additionally, the third party firm providing outsourced development services must demonstrate that they have conducted the appropriate screening during hiring.

HIPAA Best Practices for Software Development

Use only HIPAA eligible services in the Cloud (AWS) Because we use the services provided by AWS for our production environment with contains electronic protected health information (ePHI), AWS is a "business associate" of ours. It is required by HIPAA for MiCT and AWS to enter into a "business associate agreement" (BAA).

Our fully executed BAA with AWS can be found on the internal document repository.

We must only used HIPAA-eligible services covered under the BAA to process and store ePHI. Non-eligible services may be used in support of our cloud infrastructure as long as it does not have access to ePHI.

A list of HIPAA eligible services can be found here. AWS regularly updates its services to meet HIPAA compliance requirements. Check the page once in a while to find out if new services have been added.

Additional References:

- Architecting for HIPAA in the Cloud
- AWS Shared Responsibility Model
- AWS HIPAA Compliance

- AWS HIPAA Compliance Whitepaper

Separate access and data between prod and non-prod accounts It is a compliance requirement of multiple regulations to ensure separation of duties between production and non-production environments, including both access and data. Additionally, we should not use production data for dev or test, unless the data has been properly sanitized/masked.

Examples of regulations and certifications that have this explicit requirement include HIPAA, SAS70/SSAE-17, SOC, PCI, and ISO 27001/27002, many of which are on our target list to be compliant with or certified to.

Do not log ePHI Not only it is a security best practice to avoid sensitive data such as ePHI in application logging, it may be a contract violation (per AWS BAA) to do so. We must not send any ePHI to non HIPAA eligible services in AWS, such as CloudTrail.

Include the right language in notices Specific language must be included in terms, consents, and notices. For example, we must collect email addresses from patients when they sign up for the PHC, and specify in the terms that we may use the email address provided as a formal method of communication, including breach notification, should a breach occurs that impact their PHI.

Data protection Follow the requirements listed in the following documents:

- Data Classification Model;
- Data Handling Requirements;
- Data Protection Policy and Procedures; and
- Backup and Recovery Process.

Risk Analysis and Compliance Assessment Read the latest assessment report for more details on MiCT's HIPAA compliance.

Production System Monitoring and Paging

Software and systems deployed in production are monitored 24/7 for health check and other major/critical error conditions. The on call team is paged via PagerDuty in the event an error or failure is detected.

Notifications via additional channels such as Slack and email are also configured.