Review Prior to Validation

athenahealth

Application Security Best Practices

Partners/Clients/Vendors use our API to build in-house or external applications.
The following document provides an easy-to-reference set of best practices to help development teams create more secure applications.

- Error Handling and Logging
- Data Protection
- Authentication
- Input and Output Handling
- Access Control
- Session Management
- Configuration and Operations

Error Handling and Logging

1. **Display Generic Error Messages**
    - Error messages should not reveal details about the internal state of the application. For example, file system path, user ids, and stack information should not be exposed to the user via error message.
    - Be careful with your approach to confirming whether or not data exists. For example, if you expose a way to perform a patient search (e.g. during patient portal registration), ensure that the error messages are generic and do not reveal whether a user is or is not a patient at that facility or with a particular provider group.
2. **Log Access to Sensitive Data and Store Securely**
    - Logs should be stored and maintained appropriately to avoid information loss or tampering by intruders.
    - Any access to sensitive data should be logged. This is particularly important for corporations that have to meet regulatory requirements like HIPAA, PCI, or SOX.
    - If applicable, any authentication activities, whether successful or not, should be logged.

Data Protection

1. **Use HTTPS everywhere (transmitting data securely)**
    - Ideally, HTTPS should be used for your entire application. If you have to limit where it's used, then HTTPS must be applied to any authentication pages as well as to all pages after the user is authenticated. If sensitive information (e.g., personal information) can be submitted before:
2. **Implement a strong password policy and reset system (if not using SSO)**
    - If applicable, implement a policy that requires passwords to meet specific strength criteria during registration.
    - In addition, any password reset option must not reveal whether or not an account is valid. This prevents username "harvesting."
3. **Store user passwords securely (if not using SSO)**

- If the app allows for password authentication (not utilizing athenahealth SSO), user passwords must be stored using secure hashing techniques with strong algorithms such as PBKDF2, bcrypt, or SHA-512.

4. **Limit the use and storage of sensitive data**
    - Conduct an evaluation to ensure that sensitive data elements are not being unnecessarily transported or stored. Where possible, use tokenization to reduce data exposure risks.
    - Make sure that URLs posted to athenaNet do not contain exposed patient data that could be used publicly without authentication.
5. **Please use TLS 1.1 or higher**
    - TLS versions 1.0 as well as SSL versions 1.0, 2.0 and 3.0, are older protocols with known vulnerabilities that have been deprecated.

Authentication

1. **Do not hardcode API credentials**
    - Never store credentials directly within the application code. While it can be convenient to test application code with hardcoded credentials during development, this significantly increases risk and should be avoided.
    - We highly recommended that you do not check the key and secret into your codebase.
2. **Store API credentials securely**
    - The best practice for a production deployment is to maintain the key and secret in a centralized place (in this case a key server).
    - The key server is responsible for calling the OAuth endpoint to retrieve and cache the access token (until expiration). It should then make that token available for any users in your environment who need to make an API call.
3. **Compliant Patient User Authentication for Surfacing PHI**
    - Surfacing PHI in places that do not require some form of authentication is considered a violation (ex.: a link that sends a user to a webpage with patient info).
        i. Also note that surfacing PHI to users that do not need that information can constitute a HIPAA violation.
    - As mentioned earlier, revealing whether or not a patient exists in a practice (for example, Planned Parenthood or a substance abuse rehabilitation center) can constitute a PHI violation.

Input and Output Handling

1. **Conduct contextual output encoding**
    - All output functions must contextually encode data before sending that data to the user.
2. **Validate the source of output (pulling data from athenaNet)**
    - Always ensure you are pulling the correct information (e.g. the correct appointment, patient, etc.)
3. **Validate the source of input (pushing data into athenaNet)**
    - Before submitting data into athenaNet, make sure that data has been validated and sanitized to maintain integrity.

- Take measures to ensure that duplicate patients are not being created if new patients are being pushed into athenaNet.
- Only practice users can decide to update the chart, not patients.
- For any patient-entered data, make sure a physician or clinician is reconciling this information.
- Please note that normally, documents will automatically appear in the clinical inbox for providers to review but can be set to bypass the clinical box and go straight into the patient chart.

Access Control

1. **Apply the principle of "least privilege"**
   - When an account is created, rights must be specifically added to that account to grant access to resources.
   - Practices should not be allowed to access data belonging to other practices through the same MDP Partner unless explicitly agreed upon.
   - A Vendor/Partner should not have access to a Client's athenaNet Preview/Production UI unless explicitly agreed upon by vendor/partner and client.
   - Exposing PHI to users that do not need that information can constitute a HIPAA violation.
2. **Do not use unvalidated forwards or redirects**
   - Avoid using simple/naked URLs that can allow an attacker to access private content without authentication.

Session Management

1. **Re-generate access tokens**
   - Tokens should be regenerated when the user authenticates to the application and/or when the user privilege level changes.
   - As a reminder, tokens need to be refreshed every hour.
2. **Implement an idle session timeout**
   - When a user is not active, the application should automatically log the user out.
3. **Implement an absolute session timeout**
   - Users should be logged out after an extensive amount of time (e.g., 4-8 hours) has passed since they logged in. This helps mitigate the risk of an attacker using a hijacked session.
4. **Place a logout button on every page**
   - The logout button or logout link should be easily accessible to users on every page after they have authenticated.
   - When the user logs out of the application, the session and corresponding data on the server must be destroyed.
5. **Set the cookie expiration time**
   - The session cookie should have a reasonable expiration time. Non-expiring session cookies should be avoided.

Review Prior to Validation

Configurations and Operations

1. **Client requests to run reports of data not accessible via API**
    - Generally, if this data cannot be pulled via API, there is a valid reason for it (i.e. Medications or Insurances Packages are proprietary).
    - There are many cases in which it is acceptable and risk-free to run a report for a client, but to be safe, all API-related reports should always be reviewed by Compliance.
2. **Be aware that some of our endpoints are more computationally intense than others and thus subscription endpoints should be considered**
    - GET /patients vs. GET/patients/search endpoint
        i. If possible, users should leverage GET /patients/search (if they want athenaNet type patient searching) or GET /patients/enhancedbestmatch
    - GET Appointments/booked vs. GET appointments/changed endpoint
        i. The changed endpoint should be used to pull updates to the schedule rather than pulling all booked appointments every day.
        ii. The booked endpoint should be used at the start of the week
    - If you are approved to conduct large data pulls (e.g. GET /patients or GET /appointments/booked), these should be done during “off hours” (8PM - 8AM EST). We may also recommend that you implement call throttling to reduce additional risk.
3. **All functionality should adhere to our API Rate Limits as indicated on the MDP Developer Portal**