RESTful API & IOPS Resources

Security

The RESTful API Infrastructure is an application programming interface (API) framework used to develop and deploy RESTful APIs within the MEDITECH EHR. The APIs deployed on this framework can be used by internal MEDITECH applications or external third party applications.

Interoperability Services (IOPS) is a set of APIs and provides the MEDITECH EHR with next-generation interoperability capabilities. The platform runs interoperability-specific applications (collections of APIs) — enabling the MEDITECH EHR to take advantage of more advanced features and integrations.

The RESTful API Infrastructure and Interoperability Services are installed on the same server. These services are stateless — allowing the system to be easily scaled up by adding additional servers should the throughput limit be reached.

The APIs made available through the RESTful API Infrastructure are protected by multiple layers of network security, including firewalls, load balancers or proxies, and encryption. It is further secured by supporting industry standard protocols for Authorization (OAuth2.0) and Authentication (OpenID Connect and SAML 2.0).

This document aims to describe the security model.

Areas of Responsibility

MEDITECH is not a security consultancy. We provide the information within this document for informational purposes only and it is not warranted to be exhaustive or comprehensive. Our knowledge of your organization's infrastructure is incomplete and so we cannot provide a complete list of areas to examine. This material should not be relied on as a substitute for professional legal or security advice.

MEDITECH provides the software components of a REST API Infrastructure with the exception of the underlying Operating System and related REST API database. This REST API infrastructure has been developed with security in mind, and a third party application security test was performed on the instance here at MEDITECH. MEDITECH will regularly commission a third party to perform security tests on the latest versions of the infrastructure, and will continue to monitor a variety of sources to ensure that any critical updates will be provided to customers.

The customer and/or their hosting provider are responsible for provisioning the hardware, the specified operating system, and a supported REST API database.

Components

These are the key components of the REST API infrastructure. It is important to understand which organization is responsible for which components. Ultimately the customer is responsible for the security of these systems whether they are provided by MEDITECH, a hosting provider, or the customer.

Table 1 - Software Components

Component	Details	Provided by
REST API Infrastructure	This is the core of the system.	MEDITECH
REST API DB	A stable release of PostgreSQL must be used.	Customer
Redis Cache	The cache is used to reduce the strain on the REST API database and to improve performance. It is also used for quota and session tracking.	MEDITECH
SSL Certificate	Encryption of outside traffic to the RESTful API infrastructure. It is recommended that the load balancer or proxy handle encryption.	Customer

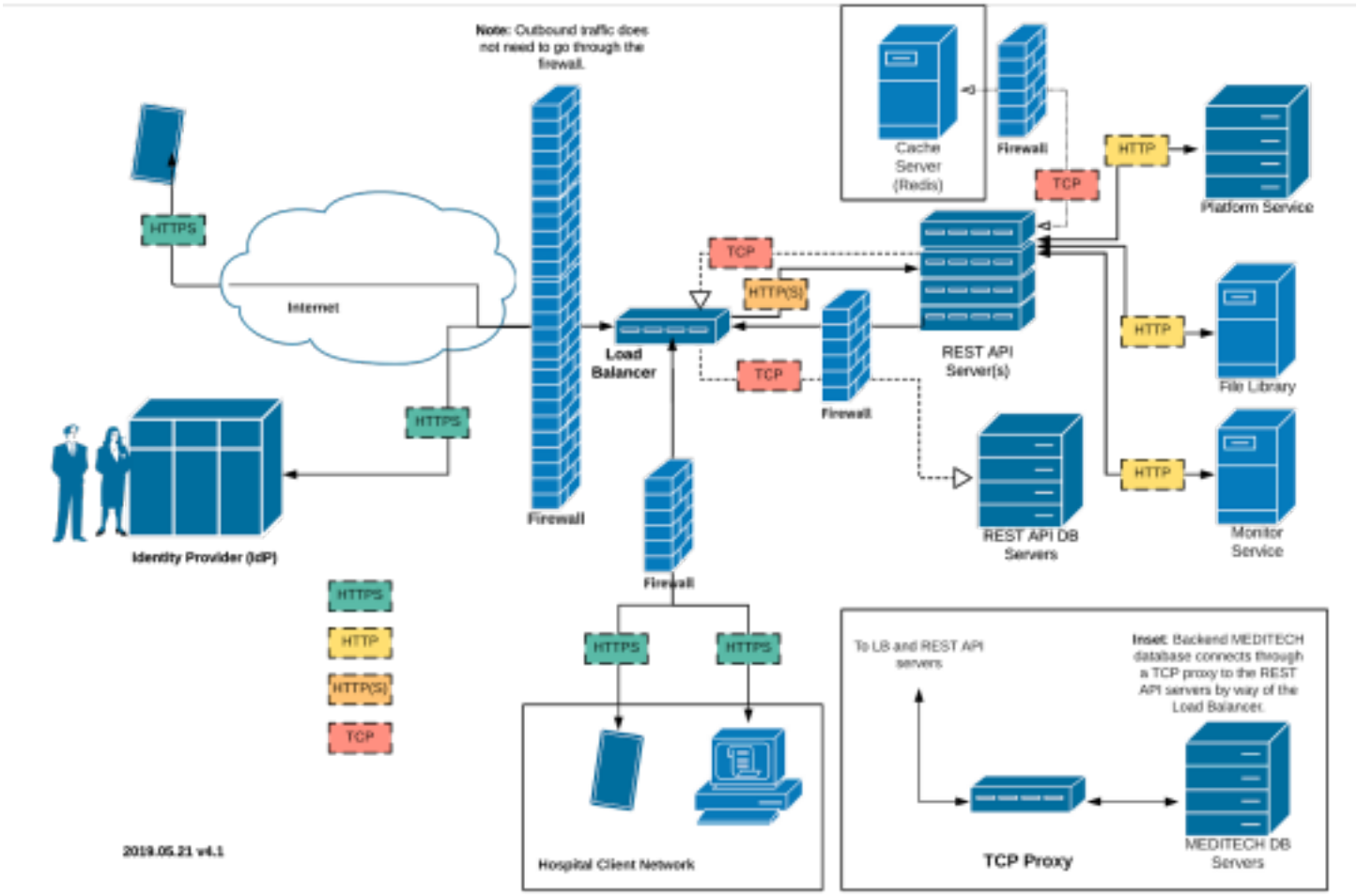
Table 2 - Hardware Components

All of the hardware is provided by the customer and/or their hosting provider.

Component	Details
Load Balancer *or* Proxy	A load balancer *or* proxy server is required.
Servers	One or more virtual or physical servers to host the following:
	REST API server
	Redis cache server (may be installed on the REST API server)
	Database server
	Monitor service
	File Library

Network Diagram

Figure 1.



Security Discussion by Component

Firewall

The firewall protects against many threats such as DoS. In some environments, a firewall may be used to restrict access to the APIs over the internet to a set of well-defined sources.

Load Balancer (LB)/Proxy Server

The load balancer performs multiple functions in this configuration. It returns an error if the request is malformed (i.e., not a valid HTTP request) and is used to offload SSL/TLS, allowing organizations to manage their certificates in a central location (instead of propagating certificates to multitudes of servers). It also makes setting SSL/TLS parameters, such as minimum protocol version and cipher suites, easier without the need to update or reconfigure the RESTful API Infrastructure. The load balancer also allows the site to mitigate TLS/SSL vulnerabilities — this is important as MEDITECH cannot deliver TLS/SSL vulnerability fixes as fast as your load balancer vendor. A proxy server may also be used instead of a Load Balancer. Within this documentation we generally refer to the load balancer because that is preferred.

There should be a firewall between the LB and the Internet, but there is no need for a full DMZ. The LB has security features built into it already. The LB prevents outside traffic from reaching any other servers but the REST API servers. However, there is still a need for a firewall between any

hospital clients and the LB. There should also be a firewall for traffic going outbound from the REST API servers to the LB.

One additional feature of a balancer is the ability to scale out the RESTful API Infrastructure to handle more requests or to increase performance by balancing requests across the nodes in an installation. Please speak with your load balancer/proxy vendor to determine whether or not the appliance should be placed in a DMZ.

TLS/SSL

All requests to the RESTful API Infrastructure must be over HTTPS and should be protected with a valid certificate (i.e., not a self-signed certificate), especially for production systems.

Redis Cache Server

The Redis service, which MEDITECH will install on the Cache Server, reduces latency and increases performance on requests by reducing the number of hits to the database. The cache is memory-only, meaning it is never persisted to disk.

It is suggested that the Redis service runs on its own server (as seen in the network diagram, figure 1). However, the site may choose to have MEDITECH install the Redis service on one of the API servers if additional servers cannot be obtained. If the decision is to combine the two servers, it is suggested that you increase the RAM available to that API server. Please refer to the "Getting Started" guide for complete details on implementation of the Cache Server.

As detailed in the security documentation here on the official Redis site,

> "Redis is designed to be accessed by trusted clients inside trusted environments. This means that usually it is not a good idea to expose the Redis instance directly to the internet or, in general, to an environment where untrusted clients can directly access the Redis TCP port or UNIX socket."

Redis is designed for high performance, hence it has no brute force protection, and limited security features are built in. With that being said, It is highly recommended that sites enable the password authentication feature on their Redis Cache Server by following the instructions located here. Redis Labs also provides the following recommendation related to network security:

> "Access to the Redis port should be denied to everybody but trusted clients in the network, so the servers running Redis should be directly accessible only by the computers implementing the application using Redis."

Having the Redis cache service running on a REST API server on loopback is one option but means that the REST API service cannot be scaled up and run on other servers. When there is a separate cache server, an organization should follow the recommendations and put it in its own zone. This can be done using a virtual or hardware firewall which restricts traffic to the Redis Cache server. Only the REST API ISWEB server(s) should be able to communicate with the cache server.

REST API Database

A supported database is required. This database stores the configuration and runtime details of the RESTful services. It does not store patient data nor any other data that is in the MEDITECH database.

This database is used for the following purposes:

- Configuration of the infrastructure (ports, APIs, etc…)
- Storage of Tokens
- API definitions and resource schemas
- API audit logs are housed in this DB.

The only data in this REST API database is from the infrastructure itself.

Note: All requests are audited and the urls being logged *will contain patient identifiers (e.g. PHI)*. There is no way to anonymize those entries.

As noted in the "Getting Started" guide, organizations should choose a database which they are most comfortable with as MEDITECH does not provide support for maintaining or configuring the database service(s).

The RESTful API Infrastructure supports the PostgreSQL database.

Due to licensing restrictions, MEDITECH cannot provide this database. Each organization must provision the database for themselves. Please refer to the getting started guide for required versions of this software.

It is important to properly secure this database in the following ways:

- It must be unavailable from internet
- Network segmentation - it should only be accessible to the REST API servers and related services.
- Access to the REST API database and its tables should be limited to DBA users and the two accounts used by the REST API services.

Web Application Firewalls (WAF)

Some organizations have asked whether or not a Web Application Firewall (WAF) would provide any additional security. Every organization will need to make its own evaluation. The REST API infrastructure is essentially an API gateway — it provides a single point of entry to the RESTful APIs. As such, it was designed with security as the top priority. It prevents malformed, unknown, and unauthorized requests from reaching the backend EHR database. Furthermore, no patient data is stored within the REST API infrastructure.

Network Configuration Conclusions

Based on the foregoing points, this is a set of basic recommendations. Every organization must do its due diligence in considering how best to secure its REST API infrastructure.

1. The Cache server(s) **should** be isolated from the rest of the network by a firewall - only the REST servers should be able to connect. (This is unnecessary if they are configured as localhost only, though a separate cache server is preferable since it allows for easy scalability.)
2. Internet traffic **must** pass through a firewall before reaching your load balancer or proxy.
3. Both Internet and internal HTTP/HTTPS traffic for the REST servers **must** go through a load balancer or proxy.

4. We **suggest** SSL offloading at the load balancer/proxy for increased performance.

5. The connection between the REST servers and the Cache, Platform, Explorer, and TCPProxy services needs to have minimal latency and, therefore, we **suggest** that the REST servers be located in the same network zone as the servers running those services; placing REST into the DMZ will mean extra latency will be added to those connections and this will negatively impact the performance of the system.

6. There is a control panel where client IDs and secrets are entered — this is only available on loopback. The admins at site must RDP to the server which houses the database in order to access this panel.

Applications

In order to put the layers into context, it is necessary to describe the flow of a typical application.

Authorization

1. The application redirects the user to our authorization endpoint.

2. The authorization endpoint does one of two things:
    a. Display the available authentication realms (if there is more than one), in which case it redirects the user to the identity provider after a realm is selected.
    b. Redirect the user to the only identity provider if there is a single realm.

3. The authentication completes[1] and the user is redirected back to a special endpoint on our service that processes the authentication information. This includes a step where the service (not the client) contacts the provider to validate the information, thereby mitigating most MITM attacks.

4. The infrastructure evaluates whether or not the application and user combination has access to the requested scopes. It does this by confirming:
    a. a. All scopes in the request are valid.
    b. b. The user passes the ACLs for all scopes.
    c. c. The application passes the ACLS for all scopes.

5. An authorization code is generated and the user is redirected back to a registered endpoint belonging to the application.

6. The application makes a request to the RESTful API Infrastructure to trade the authorization code for an access token . The connection is authenticated by the application providing its OAuth2 client ID and secret.

7. The application can now make requests to APIs on the behalf of the user.

API Invocation

1. The application makes an HTTPS request to the RESTful API Infrastructure and includes the

[1] What happens on the identity provider is outside the scope of this document.

access token.

2. The infrastructure determines if the scopes granted to the token during authorization cover the API being invoked. If not, an error is returned.

3. The infrastructure then determines if there are any quota lockouts — and if there are, returns an error.

4. A second level of authorization based on resource ownership is performed using the identity information tied to the token.

    ○ For example, Patient A is considered to be the owner of their own patient resource. Only the patient (or others specifically granted access such as providers or healthcare proxies) can read the medical record for that patient. This does not mean the patient can do everything with their record, but only what a patient is allowed to do — in this example, obtain a copy of it.

5. The invocation — whether allowed, denied, or failed due to error — is logged in our audit table.

Application Registration

In order for an application to be able to obtain a token, they must first be added to the configuration by an administrator. This generates a client ID and client secret, which is then used by that application when obtaining tokens. The ID and secret do NOT grant access to APIs and are only used during authorization.

Important: The client secret should only be transmitted and stored securely. **DO NOT** send it via email.

User Login

MEDITECH requires the use of an Identity Provider (IdP) to authenticate patients (or their representatives) and providers. The IdP your organization deploys must support the OpenID Connect protocol. MEDITECH does not provide an identity provider. Before selecting an IdP for your organization, please work with MEDITECH so due diligence can be done to confirm its compatibility with RESTful API Infrastructure.

Note: The most immediate use case for the RESTful API Infrastructure and IOPS is to meet MU3 requirements, which will only use the IdP to authenticate patients. Future use cases will include allowing providers to use applications to access data. At that time, an IdP will need to be able to authenticate providers. This may mean a single IdP or multiple IdPs depending on your needs.

OAuth2

The OAuth2 standard is used to secure the APIs from unauthorized access.

Scope ACLs

Scopes have their own set of ACLs which prevent applications or users from obtaining a token with those scopes. Most scopes have a default ACL that denies token generation (i.e., whitelisting), while some other scopes have a default ACL that allows it (i.e., blacklisting). Which method MEDITECH chooses for a scope depends on the intended usage of the scope.

The evaluation of whether or not a token may be generated with the requested scopes is done during the Authorization stage after user authentication. This is because ACLs can also be used to deny/allow based on the authenticated user in addition to the authenticated application.

The Admin Panel can be used to view and configure these ACLs.

Quotas

Quotas allow a site to limit the number of requests made within a period of time. This helps alleviate strain on the MEDITECH database if an application is misbehaving. It also acts as another layer of security by temporarily locking out an application or user from accessing APIs.

Auditing

All requests to the RESTful API Infrastructure are audited. This enables identifying what data was accessed by bad actors, should a breach of a user's or application's credentials occur. It could also be used by analytics tools to report on suspicious activity. We are still developing the tools that can be used to review the audit entries.

Develop Processes for Providing Access and Onboarding apps

It will be up to your organization to develop a process by which patients are provided access to their health information via an app. As noted earlier in this document, the methods by which this will be counted towards the measure are outlined in our Best Practice documentation for Provider to Patient Exchange. You will also need an organizational strategy for how to vet and onboard any app that a patient presents. MEDITECH will not be providing a list of endorsed apps at this time.

HHS has provided the following clarifications regarding a covered entity's liability in relation to third party apps:

- The HIPAA access right, health apps, & APIs
- What is the liability of a covered entity in responding to an individual's access request to send the individual's PHI to a third party?

The Law Firm Manatt also wrote this interesting article — Risky Business? Sharing Data with Entities Not Covered by HIPAA.

Conclusion

Securing the REST API infrastructure is a team effort. While MEDITECH provides certain key components and initial guidance, each organization is ultimately responsible for securing and maintaining their infrastructure. It is recommended that each organization work closely with their hardware integrator, MEDITECH, and security consultants in order to properly secure their REST API infrastructure.

Terms

ACL
Access Control List: A mechanism of defining who and what can access which resources.

Blacklisting
A way of granting access where access is allowed, unless an exception is made. To put it into an analogy: "Anyone may use the coffee pot except children under 12." This is the opposite of whitelisting.

Client ID
This is the unique identifier given to an OAuth Client, otherwise known as an application or service. It can be thought of as the username for an application.

Client Secret
This is a secret value given to an OAuth Client. When paired with the Client ID, it allows an application to authenticate as part of the authorization flow.

DoS
Denial of Service: An attack against a server that leads to clients being unable to connect or receive responses due to exhausting the resources of the server (CPU, IO throughput) or network.

Localhost
This is a special network interface, provided by the OS, which allows processes on the machine to communicate with each other. It usually has the standard "127.0.0.1" address. This address can only be reached by other processes running on the machine and not from the network.

MITM
A man-in-the-middle attack is one where a client's requests are sent to an insecure server that acts as a proxy to the real server.

Quota
A quota is a way of limiting the number of requests that can be made within a period of time. Quotas should be based on real-world application workflows.

Resource
A resource is any file or uniquely identifiable data, such as a patient record or an order.

Scope
A scope defines a set of related APIs.

SSL Offloading
This is a feature of most load balancers and proxies. It means that SSL encryption/decryption is offloaded from the server(s) behind the load balancer/proxy. This decreases the memory and CPU usage of the processes on those servers as they no longer need to handle encryption.

Token
A token is a string used to provide identity without exposing usernames and passwords to applications.

Whitelisting

A way of granting access where access is denied, unless an exception is made. To put it into an analogy: "No one may access the server room except network engineering." This is the opposite of blacklisting.

References

1. Redis Labs: Redis Security
2. HHS: The HIPAA access right, health apps, & APIs
3. HHS: FAQ on Liability
4. Manatt: Risky Business? Sharing Data with Entities Not Covered by HIPAA
5. Malwarebytes Labs Blog: Who is managing the security of medical management apps?

Disclaimer

MEDITECH is not a security consultancy and the information we provide ("material") — either on our Information Security web pages or as a pdf — is not warranted to be exhaustive or comprehensive. This material has been provided for informational purposes only and we make no guarantees that use of this information will secure your organization. Our knowledge of your organization's infrastructure is incomplete so we cannot provide a complete list of areas to examine. This material should not be relied on as a substitute for professional legal or security advice. Additionally, caution is needed when making some of the changes mentioned here. Changes must be tested before being made in a live environment. Any questions about this information should be directed to the MEDITECH Security Team.

Version

Document Status	**Final**
Version #	3.0
Effective Date	2/27/2024
Approved By	Richard Amicangelo
Approved date	2/27/2024

Change Log

Date/Version	Changed by	Details
May 15, 2018 V 1.0	Lucas Lacroix	Initial version of document.
May 21, 2019 V 2.0	Justin Armstrong; Lucas Lacroix	Updated to include more detail and a network diagram.
Feb 14, 2024 V 3.0	Richard Amicangelo; Justin Wright	Updated with more detail