OpenAPI Based Strategies And Tools

For Development, Verification And Use Of Web Services

Based On RO-ICM And RO-Verifier Packages

Neda Communications, Inc.

Email: http://www.by-star.net

PLPC-180061

January 4, 2020
Version 0.3

Contents

List of Figures

Part I

Overview And Context

1 Overview

1.1 About Remote Operations And Web Services

The basic concept and model of Remote Operations (RO), has evolved to be primarily web based. Web-Services, REST-APIs, microservices and services-oriented architecture (SOA) have become the dominant model for exposing Remote-Operations through https as web services.

The model of Remote Operations consists of three realms.

1. The Operations Specification Realm – Service Specification – the API
2. The Performer Realm – server/services side
3. The Invoker Realm – client/user/device side

Consistent with this model we address these topics in three related documents.

1.2 One Of Three Related Documents

To address the totality of the domains of Remote Operations and Web Services, we have put together a triad of documents. Each of these focuses on one of the above mentioned aspects of Remote Operations. These three documents are inherently interrelated. We provide an overview of each document below and briefly describe how they are related.

1.2.1 This Document – PLPC-180061: The Over Arching Model Of Web Services

In this document we provide the model and terminology that are then used by specific Performers and Invokers frameworks that we ddescribe in two related documents. Additionally, this document outlines a specific strategy that can be adopted by any organization to methodically verify its web services comprehensively and also with a focus on security in a generalized fashion.

1.2.2 PLPC-180050: Performer Framework (RO-Performer)

In a sister document, titled:

> **Unified Python Interactive Command Modules (ICM) and ICM-Players**
> **A Framework For Development Of Expectations-Complete Direct Commands And Remote Operations**
> **A Model For GUI-Line User Experience**
> http://www.by-star.net/PLPC/180050 — [?]

we put forward the concept of Expectations-Complete Operations as a model for development of RO-Performers. By a "Unified Framework" we are underscoring the notion that based on this model of Expectations-Complete-Operations, we can generate:

- Command Modules (ICM) that faciltate local invocation of the Operations from the command line

- Remote Operation Performers (RO-ICM) that faciltate remote invocation of the Operations as Web Service
- Direct native python invocation of the Operation

Based on the unified specification and implementation of Expectations-Complete-Operations we then can fully automate creation of RO-Performers.

PLPC-180050 documents the model, the framework, the software and the use of this software package.

1.2.3 PLPC-180057: Invoker Framework (RO-Invoker and RO-Verifier)

In a sister document, titled:

> **RO-Verifier:**
> **A Remote-Operations Invocations Framework**
> **Tools And Strategies For Generalized OpenAPI/Swagger Based Verification Of Web-Services**
> http://www.by-star.net/PLPC/180057 — [?]

we are proposing that a specific invocation and verification software package be used for both general verification and security oriented verification of existing and new web services.

This document provides the needed information for obtaining, installing and using RO-Verifier and related software packages.

Scope and span of these web-services verification tools is the entirety of operations specified in the OpenAPI/Swagger file. In that broader context, these tools can be used by service developers for complete regression testing.

1.3 A Generalized Approach

We consider our approach as "Generalized" because it only assumes the following from the service provider.

1. The Complete OpenAPI/Swagger Specification.
2. The Service Access Point. Provided either as a url or as service discovery methods.
3. Necessary credentials for accessing the service and for invoking operations.

In a proper digital ecosystem, these can be assumed to be available for all web services and as such the proposed tools and framework are applicable to all these web services.

Remote-operations/web-services are by nature network exposed. The presence of many web services exposes a large attack surface. Therefore, only a generalized validation approach which is applicable to all OpenAPI/Swagger web services is viable. Furthermore, the proposed generalized verification framework is inherently batch oriented and repeatable. So, the validation process conforms to the regression testing model and can be used re-verify the service as needed.

Since these tools view the performer as a blackbox and engage in invocation verification of the service based on its contract (the OpenAPI/Swagger specification), they are equally applicable to:

- in-house built web-services.
- public open-source web-services
- private (proprietary) closed-source web-services

For in-house developed and maintained web services, the RO-Verifier Framework can be deployed ab initio and used continuously as a quality assurance measure.

The RO-Verifier framework is based on ingesting the OpenAPI/Swagger specification and then mapping it to a set of corresponding functions and data structures in Python. The ingestion and mapping is performed by an open-source software package called: Bravado. Bravado creates its Python library dynamically without a code generator. RO-Verifier is a layer on top of Bravado where the Remote-Operations invocation abstractions are used to allow users to create scenarios for invocations and corresponding verifications. These customized scenarios can then be used to subject the performer (the web service) to regression testing and automated repeatable verification. RO-Verifier framework and the developed scenarios can be integrated with the web service development process to improve reliability and security.

1.4 Security Of Web Services

Web services are by nature network exposed and therefore present a large attack surface that needs to be continuously secured and continuously verified. The theoretical aspects of this vulnerability and the associated security requirements have been well analyzed in documents such as: NIST-SP-800-204 – "Security Strategies for Microservices-based Application Systems" – [3].

Such documents and analysis are generally abstract and theoretical. The frameworks and approaches that we are putting forward in this document are based on explicit use of a concrete framework and a set of specific software packages.

There are three areas of security oriented blackbox verifications to which these tools apply:

1. Identify and test common security vulnerabilities that are manifested in the OpenAPI/Swagger specification
2. Validation of credentials (invalid and expired tokens) and access control
3. Brute force attacks that evaluate the services response to denial of service type attacks

In that context, in a sense our approach can be considered a form of penetration testing for web services. However, this framework allows for such tests to be considered generalized and become consistently applicable to all web services, instead of isolated penetration testing of web services one at a time.

Based on this approach, the proposed framework and these tools, with discipline and methodically we can continuously validate the entire web services foundation of the applications fabric of a digital ecosystem. The benefit is a more robust and more secure applications environment.

1.5 Completely Libre-Halaal And Open Source

The RO-Verifier is open-source and is subject to Affero General Public License (GPL).

1.6 Document Outline

In Part II, we provide some relevant information about OpenAPI/Swagger specifications and invoker and performer realms of web services.

In Part III we describe different uses of the RO-Verifier framework and suggest some next steps towards adopting RO-Verifier in your own context.

2 Context

2.1 ByStar: The Broader Context

This is part of a bigger picture. The origin of RO-Verifier is the ByStar digital ecosystem.

Many of the architecture principles presented in the ByStar context are equally applicable to any large enterprise.

2.1.1 Public Presentation Form Of This Topic

A presentation form of this subject matter has been publicly available in:

> **RO-Verifier:**
> **A Remote-Operations Invocations Framework**
> **Tools And Strategies For Generalized OpenAPI/Swagger Based Verification Of Web-Services**
> http://www.by-star.net/PLPC/180057 — [?]

2.1.2 About ICM (Interactive Command Modules)

Interactive Command Modules (ICM) is a framework for development of Python and Bash modules that conform to particular structures which permit their functions to map to command-line.

Based on the tools and strategies described in this document, ICMs can become Remote-Operation Interactive Command Modules (RO-ICM) which can then be deployed as Web Services.

Both, the RO-Verifier Framework and the ByStar Web-Services Development Framework are based on the Interactive Command Modules (ICM) model which is documented in:

> **Unified Python Interactive Command Modules (ICM) and ICM-Players**
> **A Framework For Development Of Expectations-Complete Direct Commands And Remote Operations**
> **A Model For GUI-Line User Experience**
> http://www.neda.com/PLPC/180050 — [?]

2.1.3 About BISOS (ByStar Internet Services OS)

ByStar Internet Services OS (BISOS) is a universal Services OS layered on top of Linux. All ByStar services and all ByStar usage environments (devices) use BISOS.

Based on the ICM framework, in the ByStar context, BISOS is documented in:

> **The Universal BISOS: ByStar Internet Services Operating System**
> **Model, Terminology, Implementation And Usage**
> **A Framework For Cohesive Creation, Deployment and Management Of Internet Services**
> http://www.by-star.net/PLPC/180047 — [1]

2.1.4 About ByStar

The strategies and tools being proposed in this document have their origins in the ByStar digital ecosystem and are practiced in ByStar.

ByStar is an autonomy and privacy oriented digital ecosystem. For more information about ByStar, see http://www.by-star.net

The universal BISOS is then used for realization of the ByStar digital ecosystem, which is documented in:

> **The Libre-Halaal ByStar Digital Ecosystem**
> **A Unified and Non-Proprietary Model For Autonomous Internet Services**
> http://www.by-star.net/PLPC/180016

A podcast providing an overview of ByStar is available at:

> **The Libre-Halaal ByStar Digital Ecosystem Elevator Pitch Videos**
> **With Pointers For Digging Deeper**
> http://www.by-star.net/PLPC/180054 — [?]

2.1.5 About Mohsen Banan

Mohsen Banan is the primary author of this document, the primary designer and implementor of RO-Verifier software, the origin of ByStar, the primary architect of BISOS and the primary implementor of the ICM Framework.

For more information about Mohsen, see http://mohsen.1.banan.byname.net

Part II

Background

3 Web Services: Technology Background

The basic concept and model of Remote Operations (RO), with some variations on emphasis has morphed into various terms and key words. Some of these include:

- Web Services – Where use of http/https as the underlying protocol for remote operations is emphasized and where use of JSON for Arguments and Results encoding is emphasized.
- micorservices – Where building on other services, API-Gateways, service-discovery and containerization is emphasized.
- serverless – Where on demand realization of the performer is emphasized.
- SOA (Service Oriented Architecture) - also called service-based architecture – Where the notion of services being provided to the other components by application components is emphasized.
- RESTful APIs (Representational State Transfer) – Where stateless operations are emphasized.
- SOAP (Simple Object Access Protocol) – Where neutrality of transport is emphasized and where use of XML for Arguments and Results encoding is emphasized.
- Protocol Buffers – Where encoding efficiency is emphasized.
- ESRO (Efficient Short Remote Operations) – Where protocol efficiency is emphasized and constrained systems and IoT are addressed.
- RPC (Remote Procedure Call) – Where the notion of inter-process communication is emphasized and where use of XDR for Arguments and Results encoding is emphasized.
- ROSE (Remote Operation Service Element) – Where the formality of the ISO Reference Model is emphasized and where use of ASN.1 for Arguments and Results encoding is emphasized.

All of these are really just variations and trends on the basic model and concept of Remote-Operations.

The model and terminology of Remote-Operations has diverged over the past 30 years into many branches. Each of these flavors come with their own model and lingo. The models and terminology of these flavors are redundant, ambiguous and inconsistent; resulting into confusion.

This confusion then results into mis-use of the terminology which then morphs and degenerates the underlying concept. Terms are often used without precision and out of place leading to more confusion. For example in common use, RESTful API have become synonymous to Web Services even when REST is not relevant.

Furthermore, the technology behind some of these terms may have become obsolete or may have fallen out of fashion, yet use of their vocabulary continuous.

So, it is an understatement when we say the current model and terminology of Web-Services is messy.

To address this we put forward our own model and terminology.

4 Overview Of Model And Terminology Of Remote-Operations/Web-Services

Our model and terminology for Remote-Operations/Web-Services is based on:

X.880: Remote Operations: Model, Notation and Service Definition

CCITT/ITU and ISO/IEC in X.219 [2] and updated in X.880 and X.881.

Throughout this document and in the RO-Verifier code We use ROSE's model and terminology of Operations (with some augmentations), Invokers and Performers – not the ad hoc clients and servers terminology of web services folklore.

The basic model and concept is depicted in Figure 1.

Where;

- The Remote Operations Invoker, invokes a Remote Operation based on the Service Specification at the Service-Access-Point and provides its Argument.
- The Operation is described by its Argument, Result and Error.
- The Remote Operations Performer, performs a Remote Operation based on the Service Specification at the Service-Access-Point and provides a Result or an Error.

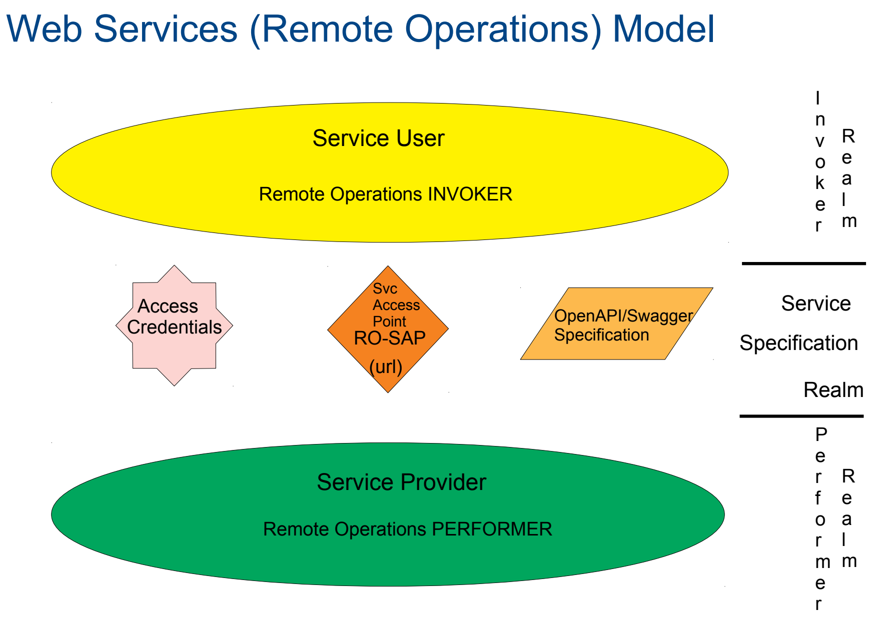
Web Services (Remote Operations) Model
Service User
Remote Operations INVOKER
Invoker Realm
Access Credentials
Svc Access Point RO-SAP (url)
OpenAPI/Swagger Specification
Service Specification Realm
Service Provider
Remote Operations PERFORMER
Performer Realm

Figure 1: The Basic Model Of Web Services (Remote Operations)

This basic simple model is common across the above mentioned key words and terms.

In 2019, the industry has converged around the following practices for Remote Operations:

- Use of HTTP/HTTPS and its full set of verbs for the transport of synchronous remote operations.
- Use of JSON for encoding of Arguments, Results and Errors.
- Use of Swagger/OpenAPI for the formal specification of the Remote Operations Service.
- Use of RESTful model for making the service stateless and resilient
- Use of OAuth token for Authentication and Authorization in the context of IAM (Identity and Access Management)

- Use of containers (e.g, docker) and orchestration of containers (e.g., swarm, kubernetes) to allow for scalablity and resilience of performers.
- Use of front-end controller/proxies to load balance, rate limit and protect, in common, the providers.

Sometimes, various of these common and core needed features are bundled together as umbrella frameworks. Existance of such containerization and front-endings do not impact our basic model.

With these specifities in place we now further refine our Web Service (Remote Operations) Model.

5 Web Services (Remote Operations) Model

As illustrated in Figure 1, our model consists of three realms.

1. The Performer Realm – server/services side
2. The Service Specification Realm – the API
3. The Invoker Realm – client/user/device side

The concepts and model of identity, authentication, authorization and access control spans these three realms and is an essential part of the remote operations framework.

Each of these realms have their own characteristics and environments. We expand on these and the associated authorization model below.

5.1 The Service Specification Realm

The services (operations) of the Performer are exposed to the Invoker in the service specification realm.

The service specification realm consists of three fundamental pieces.

- The Operations Specification – the Swagger/OpenAPI specification – The API
- The Remote Operations Service Access Point (RO-SAP) – the url at which the service is exposed.
- Access Credentials

5.1.1 The Operations Specification – the Swagger/OpenAPI specification

OpenAPI Specification (formerly Swagger Specification) is an API description format for Remote Operations that the Performer supports. An OpenAPI specification allows you to describe your entire API, including:

- Available endpoints and operations on each endpoint
- Operation parameters Input and output for each operation
- Authentication methods
- Usage information and comments

OpenAPI/Swagger specification is an ad hoc standard. It is not a product of any formal standardization organization.

The OpenAPI/Swagger specification has facilitated the creation of a great deal of tools (code-generators, libraries, user-interfaces, editors, validators) and facilities. We provide an overview of some of these below.

5.1.2 Remote Operations Service Access Point (RO-SAP)

The point at which the service is delivered is called the Remote Operations Service Access Point (RO-SAP). In common web-services, the base url for the service is the RO-SAP.

The RO-SAP may be explicitly specified or it may be discovered. A discovery process can be used to select an explicit RO-SAP.

5.1.3 Authentication Information And Access Credentials

The Operations Specification (OpenAPI/Swagger file) may include description of Security Requirement Object, Security Schemes and OAuth Flows Object which describe methods and formats.

To invoke operations, the invoker needs provide authentication information and access credentials that the performer expects.

5.2 Performer Realm

The performer is responsible for the realization of the services specified in the Service Specification Realm.

The performer needs to scale, be resilient, be secure and be performant.

There are usually one or few implementation of a performer – typically in one or few programming-languages.

5.2.1 Three Ways Of Specification Of Swagger-file By The Performer

1. Design and write the Service Specification in full in one place – e.g., with swagger-editor. Then use code-generators for both performer and invoker.
2. Design and write the Service Specification in pieces along with implementation (a la Dropwizard, Spring-Boot). Then publish the aggregated swagger for code-generation of invokers. Performer is framework driven – no generated code.
3. Design and write the Service Specification in pieces along with implementation for single process usage. Then generate the aggregated Service Specification for use with code-generators for both performer and invoker.

Method (3) is that of Web Services ICM.

5.3 Invoker Realm

The operations specified in the Service Specification Realm can be invoked by users.

The Invoker Realm includes many users of different capabilities and implemented in many programming-languages. Typically based on the Service Specification, code-generators create code for libraries that client code can use. For dynamic languages such as Python, tools such as Bravado do the equivalent of code-generation on the fly such that in-effect the Service Specification is mapped to the target programming language.

5.4 Operations Authorization And Access Control Model

The performer should only perform the operation if the invoker is authorized to do so. Further the performance of an operation may depend on the idntity of the invoker.

The industry has converged on the model and framework of OAuth at this time. OAuth (Open Authorization) is an open standard for token-based authentication and authorization. OpenAPI allows for reflections of OAuth framework in the specification.

6 Use Of Remote-Operations Services Terminology In RO-Verifier

The RO-Verifier software and framework use a formal model and terminology.

ITU X.880 and X.881 which are harmonized with ISO/IEC 13712-1, provide a model, terminology and service definitions for Remote Operations.

Such a valuable formal model and terminology is absent in the Web Services world and the OpenAIP/Swagger world.

The RO-Verifier software exposes web services capabilities in the Remote Operations model which conform to the ROSE terminology.

A summary of X.880 and X.881 terminology is included below. These reflect the Python interface that RO-Verifier provides.

6.1 ROS Model

Remote operations (ROS) is a paradigm for interactive communication between objects. Objects whose interactions are described and specified using ROS are **ROS-objects**. The basic interaction involved is the invocation of an operation by one ROS-object (the invoker) and its performance by another (the performer).

Completion of the performance of the operation (whether successfully or unsuccessfully) may lead to the return, by the performer to the invoker, of a report of its outcome.

A report of the successful completion of an operation is a **result**; a report of unsuccessful completion an **error**.

During the performance of an operation, the performer can invoke **linked operations**, intended to be performed by the invoker of the original operation.

For correct interworking, certain properties of the operation must be known by both invoker and performer. The properties include:

- whether reports are to be returned, and if so, which ones;
- the types of the values, if any, to be conveyed with invocations of the operation or returns from it;
- which operations, if any, can be linked to it;
- the code value to be used to distinguish this operation from the others that might be invoked.

6.2 ROS Concept Definitions

argument: A data value accompanying the invocation of an operation.

contract: A set of requirements on one or more objects prescribing a collective behavior.

error: A report of the unsuccessful performance of an operation.

linked operation: An operation invoked during the performance of another operation by the (latter's) performer and intended to be performed by the (latter's) invoker.

object: A model of (possibly a self-contained part of) a system, characterized by its initial state and its behavior arising from external interactions over well-defined interfaces.

operation: A function that one object (the invoker) can request of another (the performer).

parameter (of an error): A data value which may accompany the report of an error.

result: A data value which may accompany the report of the successful performance of an operation.

ROS-object: An object whose interactions with other objects are described using ROS concepts.

synchronous: A characteristic of an operation that, once invoked, its invoker cannot invoke another synchronous operation (with the same intended performer) until the outcome has been reported.

6.3 ROS Service Definitions

RO-INVOKE: The RO-INVOKE service enables an invoking AE to request an operation to be performed by the performing AE.

RO-RESULT: The RO-RESULT service enables the performing AE to return the positive reply of a successfully performed operation to the invoking AE.

RO-ERROR: The RO-ERROR service enables the performing AE to return the negative reply of an unsuccessfully performed operation to the invoking AE.

RO-REJECT-U: The RO-REJECT-U service enables one AE to reject the request or reply of the other AE if the ROSE-user has detected a problem.

RO-REJECT-P: The RO-REJECT-P service enables the ROSE-user to be informed about a problem detected by the ROSEprovider.

Argument: This parameter is the argument of the invoked operation.

6.4 Adding REST To ROSE

Our use of the X.219 (Remote Operation Service Element – ROSE) model, then needs to be extended to include the RESTFul model.

The basic concepts of ROSE (Remote Operations Services Element) can easily be augmented by the basic concept of REST (Representational State Transfer).

In the Web Services context we can go from Remote Operations to REST's object, method model by introducing the notion of "RO-Sap-endpoint" (Remote Operation Service Access Point Endpoint).

6.5 Mapping Of Our ROS Terminology To OpenAPI/Swagger Terminology

We intend to add a table representing a mapping between the informal terminology of Web Services and OpenAPI/Swagger to the formal terminology presented here in future revisions of this document.

7 Swagger/OpenAPI Resources And Ecosystem

7.1 Swagger / Open API Specification (OAS) Versions 2 and 3

The Swagger Editor is an open source editor to design, define and document swagger specifications.

An instance of swagger editor is hosted at: `https://editor.swagger.io/`.

Amongst swagger validators, swagger-cli `https://www.npmjs.com/package/swagger-cli` has proven to be reliable.

7.2 Swagger Code Generators

Swagger code generators that can create performer stubs and invoker interfaces exist for many languages and many frameworks.

The performer swagger code generators, generate a server stub for your API. The only thing left is to implement the server logic – and your API is ready to go live!

The invoker swagger code generators, generate client libraries for your API in over 40 languages.

See `https://github.com/swagger-api/swagger-codegen#swagger-code-generator` for more details.

Swagger Parser reads OpenAPI definitions into current Java POJOs and is available at:
`https://github.com/swagger-api/swagger-parser`

Swagger Core is a Java implementation of the OpenAPI Specification and is available at:
`https://github.com/swagger-api/swagger-core`

Swagger Hub is a resource for OAS based service development. It is based at:
`https://swagger.io/tools/swaggerhub/`

7.3 Frameworks That Produce Swagger Specifications

Many rich web services frameworks support production and maintenance of swagger specifications. Java's Dropwizzard and SpringBoot are two such frameworks.

7.4 Swagger UI

The Swagger UI is an open source tool which allows you to visually render documentation for an API defined with the OpenAPI (Swagger) Specification. It allows you to generate interactive API documentation that lets your users try out the API calls directly in the browser.

An instance of the swagger ui is available online at: `https://swagger.io/tools/swagger-ui/`

7.5 OAS-2 To OAS-3 Transition

A public web-based interface is available at: `https://mermade.org.uk/openapi-converter`

7.6 The Pet Store Example

In order to provide a live example for swagger tools chain development an example application called the "Pet Store" has been created.

See `https://petstore.swagger.io/` for more details.

7.7 Directory Of Publicly Available Swagger Specifications

A large collection of publicly available swagger specifications is collectively maintained.

See `https://github.com/APIs-guru/openapi-directory` for more details.

8 Specification Of Proper Order Of Operation Invocations

The OpenAPI specification defines what Operations can be invoked and performed but it does not specify the order in which these Operations can and should be invoked. Often the Arguments to an Operation are obtained as Results of previous Operations.

These order and dependency semantics are typically included as documentation that is packaged with the OpenAPI Specification.

8.1 Time Sequence Diagrams

Some times the order of invocation of Operations is graphically described as time sequence diagrams.

8.2 Postman Collections

Some times the order of invocation of Operations is communicated to users as postman collections.

8.3 RO-Verifier Operations Scenarios

Based on the RO-Invofier platform, we can formally express order of invocation of Operations based on OpScn (Operation Scenarios).

The model of OpScn for formal expression of the expected order of invocation of Operations has the benfit of being exact and complete and repeatable.

Operation Scenarios can be re-executed to verify that they remain valid as the service evolves.

Additionally Operation Scenarios can be executed and produce Time Sequence Diagrams.

9 Common Core Features Of Remote Operations

There are a set of common core features that are present in development and deployment of all remote-operations.

9.1 About Common Core Features Of Remote-Operations

Some of these core features are inherently cross-reflected (they span the invoker, service specification and the performer). We collectively label these core features as "Service Reflected Core Features". Service Reflected Core Features are manifested in the Service Specification (OpenApi-Spec) and involve both the invoker and the performer.

Some core features are inherently performer specific. We collectively label these core features as "Service Transparent Core Features". These features are not reflected in OpenAPI/Swagger specification and the invoker is not directly impacted by their presence. But their availability (or absence of) may be verified by the RO-Verifier tools.

Broadly speaking common core features of remote-operations are:

Service Reflected Core Features

- Identity and Access Management – Authentication, Authorization and Access Control
- Service Discovery
- Secure Communication Protocols (confidentiality and integrity)

Service Transparent Core Features

- Resiliency or Availability Improvements
- Load Balancing
- Circuit Breaker
- Rate Limiting
- Blue/Green Deployments
- Canary Releases
- Security Monitoring

For each of the service reflected core features, there are service-specification and invoker and performer dimensions. We discuss these in Section 9.2.

For service transparent core features, there are some sophisticated infrastructures that address them collectively. We discuss these in Section 9.2.

9.2 Service Reflected Core Features

The core features that directly impact the invoker need to be expressed (reflected) in the service definition.

We describe these below.

9.2.1 IAM And OAuth (Token Based Authorization)

Identity and authorization by nature span the service definition, the performer and the invoker. Each of these are discussed below.

9.2.1.1 Reflections Of Token Based Authorization In OpenAPI/Swagger

The initial form of authentication to web services involves the use of cryptographic keys. Authentication tokens encoded based on OAuth 2.0 framework provide an option for enhancing security. Additionally, a centralized architecture for provisioning and enforcement of access policies governing access to all web services is required due to the sheer number of services.

A standardized, platform-neutral method for conveying authorization decisions through a standardized token (e.g., JSON web tokens (JWT), which are OAuth 2.0 access tokens encoded in JSON format) is also required.

We call this expression of standardized, platform-neutral method for conveying authorization decisions in the OpenAPI/Swagger specification: “Reflections Of IAM In OpenAPI/Swagger”.

Proper security oriented verification of a web service demands proper reflection of reflections of IAM in OpenAPI/Swagger specification.

Swagger 2.0 facilities are incomplete in this regard but OpenAPI 3.0 is richer in this regard.

Based on conventions and local standardization it is possible to reflect a given IAM in OpenAPI/Swagger specification to the extent that automated verification of authentication and access control is possible.

9.2.1.2 Invoker’s Uses Of Token Based Authorization

9.2.2 Reflections Of Service Discovery In The Specification

The invoker needs to know or discover the service access point (SAP) address of the performer.

Performer’s SAP-Address can be included in the specific service specifictaion that is communicated to the invoker.

This is typically communicated in the form of uri.

9.2.3 Reflections Of Encryption In The Specification

The invoker and the performer need to agree on the details of communication protocols between them. Such details can be included in the service definition.

This is typically communicated in the form of uri which for example can require use of TLS for confidentiality.

9.3 Service Transparent Core Features

There are a variety of common capabilities and features that can surround and augment remote-operations performers.

These features include but are not limited to resilience assurance, load balancing, response caching, service aware health checks and monitoring. Such features are performer specific and are not reflected in the service specification and do not directly impact invokers.

Here we provide a short description for the labels that we use for these core features:

Resiliency or Availability Improvements: When an instance of a performer fails, other instances are deployed in its stance.

Load Balancing: There is a need to have multiple instances of the same service, and the load on these instances must be evenly distributed to avoid delayed responses or service crashes due to overload.

Circuit Breaker: This is a feature to set a threshold for the failed responses to an instance of a performer when the failure is above the threshold. This avoids the possibility of a cascaded failure.

Rate Limiting (throttling): The rate of requests coming into a performer must be limited to ensured continued availability of service for all clients.

Blue/Green Deployments: When a new version of a performer is deployed, requests from invokers using the old version can be redirected to the new version.

Canary Releases: Only a limited amount of traffic is initially sent to a new version of a performer since the correctness of its response or performance metric under all operating scenarios is not fully known.

Security Monitoring: Security monitoring of the performer detects, alerts and responds to inappropriate behavior of invokers.

Based on this recognition, a variety of sophisticated infrastructures have been developed to surround an instance of a performer and provide some or all of the above mentioned core features.

At this time, three basic architectural models and machineries exist to address the above. These are:

- Containerization Of Performers (docket, swarm, hubernetes)
- API Gateway (such as APIgee)
- Service Mesh

Service transparent core features are not a focus of this document, but RO-Verifier can be used to verify the existence of many of the expected performer core features.

Part III

Adoption And Use Of RO-Verifier Framework And Tools – Next Steps

10 Adoption And Use Of RO-Verifier Framework And Tools

Web services are the foundation of application fabrics in modern digital ecosystems.

Such application fabrics need to be continously verified and made secure. Not addressing such continous verifications and not securing this foundation amounts to negligence.

Any approach to this problem which is not generalized and reproducible and repeatable is not scalable and is temporary (not enduring). Such approaches are incomplete.

Any generalized and reproducible approach to web services need to be centered around their service specifications.

So, the only remaining question is: "What service specification centered tools should be used to validate your web-services?"

Through out this document we have made a case for the RO-Verifier.

If you are not convinced that the proposed RO-Verifier is the right approach, then which framework and which tools do you believe should be used instead?

The primary benefit of the proposed approach is enduring improvements in security and reliability of all OpenAPI/Swagger based web services.

The generalized validation framework of (invoke, invoke-verification and invoke-reporting) permits for disciplined complete external blackbox validation of web-services based on their service specifications.

RO-Verifier is a general purpose tool. It can be used for different purposes. RO-Verifier makes it simple and convenient for you to create different invocation scenarios for various purposes. There are several distinct purposes for which RO-Verifier can be used. We enumerate some such broad purposes below.

1. Functionality Regression Testing And Verification By Performer Developers
2. Security Oriented Blackbox Repeatable Verification
3. Penetration Testing
4. Complete End-User Oriented Python Applications Development

These purposes often overlap and evolve and morph. For example the scenarios and tools developed for security oriented verifications can be shared with performer developers and then they can jointly be further developed.

11 Uses Of RO-Verifier For General And Security Oriented Verifications Of Existing Web Services

The scope of RO-Verifier framework is not inherently just security oriented. However, in this section we emphasize the security oriented use of these tools.

11.1 Validating OpenAPI Specification

Given a web service and its swagger file, the first step is to just pass the existing specification through the command line verification tools and expect RO-Verifier to make available to you all operations and their parameters on the command line.

If there are any errors at this stage, it is very likely that the swagger file is invalid.

11.2 Security Oriented Uses Of RO-Verifier For Verification Of Existing Web Services

Security oriented validation uses of RO-Verifier fall in the following categories.

1. Validation of the web service's authentication, authorization and access control based on its service specification.
2. Verification of common security vulnerabilities that are manifested in the service specification.

11.2.1 Token Based Access Verifications

The next step is typically that of obtaining valid tokens and invoking any given operation that requires an authorization token, invoke with:

- Valid Tokens
- Expired Tokens
- Invalid Tokens

Such verifications can be automated based on positive and negative RoScenario expectations.

The next verification layer is that of subjecting all operations Of WS-Specification To authorized verifications, following these steps:

- Obtain Tokens With Adequate Privileges
- Obtain Tokens With Inadequate Privileges
- Based On Sequence-Diagrams, Create RO-Verifier-Scenarios With Authorized And Unauthorized Credentials
- Verify That All Authorized Operations Were Successful
- Verify That All Unauthorized Operations Were Rejected (Not-Performed)

11.2.2 Regression Oriented Verification Of Exposed Attack Surfaces

A human review of the service specification may point to potential security vulnerabilities for which we can easily develop scenarios that can exercise those areas. An example is common use of unbounded and unqualified strings in swagger specifications.

Verification of access control to each and every operation within the service specification can well be implemented using RO-Verifier.

11.2.3 Abusive And Denial Of Service Oriented Scenarios

The steps below can be followed to establish whether or not the performer has deployed any rate limiting defenses against abusive and denial of service oriented attacks.

- Based On Swagger Specification, Identify Scenarios That May Be Performer Resource Intensive.
- Repeatedly Invoke These Scenarios.
- Verify That Such Repeated And Abusive Invokations Are Rate Limited Or Prohibited.

11.3 Deliver The Tools And Scenarios To The Use Case's Developers

After running through the scenarios for its security oriented validation, that package of scenarios and the verification tools can be delivered to the web service developers so that they can use it as regression testers on an on-going basis.

12 Adoption Of RO-Verifier By Web Service Performer Developers

Of course, Best Current Practices (BCP) for development of OpenAPI/Swagger based web services based on the chosen framework (Dropwizard, SpringBoot, etc.) should be developed and followed.

All such BCPs needs to have rich scenarios oriented invokers that can be used as regression testers as the web service evolves. It is best to integrate these web services verification framework with the web services development framework, ab initio.

In the ByStar digital ecosystem, the web services development framework, Remote Operations Interactive Command Modules (RO-ICMs) are inherently integrated with the testing framework as they are all based on Direct Operations Interactive Command Modules (DO-ICMs). This model of fully integrating DO-ICMs, with RO-ICMs and the validation framework which are then augmented by player user interfaces is documented in:

> **Unified Python Interactive Command Modules (ICM) and ICM-Players**
> **A Framework For Development Of Expectations-Complete Direct Commands And Remote Operations**
> **A Model For GUI-Line User Experience**
> http://www.by-star.net/PLPC/180050

13 Use Of RO-Verifer For Development Of Complete Invoker Apps

RO-Verifier framework can also be used to develop complete invoker-application in python.

Invoker-Apps development model is an extension of opScenarios facilities.

14 Adoption Of RO-Verifier For Enterprise Wide Continous Verfication

Chief Security Officer (CSO) or Chief Technology Officer (CTO) of an IT department in a private enterprise or government agency who wishes to develop enterprise infrastructures to host distributed systems based on microservices architecture needs to consider the totality of the web services surface.

The first step in that context demands a central repository of all WS-Sepcifications (swagger files) in one place. Based on that repository, a series of consistent verification facilities could then be centrally developed.

15 Commercial Maintenance And Support For RO-Verifier Software

Neda Communications – http://www.neda.com – provides commercial maintenance and support services for RO-Verifier software.

References

[1] Inc. " " Neda Communications. " the libre-halaal bystar reference model terminology, architecture and design ". Permanent Libre Published Content "180047", Autonomously Self-Published, "December" 2014. http://www.by-star.net/PLPC/180047.

[2] Remote Operations: Model, Notation and Service Definition. The International Telegraph and Telephone Consultative Committee, March 1988. Recommendation X.219.

[3] Ramaswamy Chandramouli. Security strategies for microservices-based application systems. Technical Report NIST-SP-800-204, August 2019.