RO-Invofier:

A Remote-Operations Invocations And Verifications Framework

Generalized OpenAPI/Swagger Based Tools For Use Of Web-Services

Mohsen BANAN

Email: http://mohsen.1.banan.byname.net

PLPC-180057

March 23, 2020
Version 0.7

Contents

I Overview And Context 2

1 Overview 2

1.1 About Remote Operations And Web Services . . . 2

1.2 One Of Three Related Documents . . . 2

1.2.1 PLPC-180061: The Over Arching Model Of Web Services . . . 2

1.2.2 PLPC-180050: Performer Framework (RO-Performer) . . . 2

1.2.3 This Document – PLPC-180057: Invoker Framework (RO-Invoker and RO-Verifier) . . . 3

2 Context 3

2.1 ByStar: The Broader Context . . . 3

2.1.1 Public Presentation Form Of This Topic . . . 3

2.1.2 About ICM (Interactive Command Modules) . . . 3

2.1.3 About BISOS (ByStar Internet Services OS) . . . 4

2.1.4 About ByStar . . . 4

2.1.5 About Mohsen Banan . . . 4

II RO-Verifier: Generalized Web-Services Invocations And Validation Framework 5

3 Overview Of The Web-Services Validation Tools 5

3.1 Command-Line Remote Invocations – rinvoker.py . . . 6

3.2 Python Operations Scenarios – opScnSvc.py . . . 6

4 Installation Of The Web-Services Validation Tools 6

4.1 Obtaining (OAuth) Authorization Tokens And Access Credentials . . . 6

4.1.1 Install The Needed Modules (symCrypt, cryptKeyring) . . . 7

4.1.2 Prepare The KeyRing . . . 7

4.1.3 Store The Invoker Credentials In The Keyring . . . 7

4.1.4 Obtain The Token . . . 7

4.2 Run The Web-Services Validation Tools Against The Canonical Petstore . . . 8

4.3 Source Code And Packages Repositories . . . 8

4.4 Applying This Services Validation Framework To Your Own Swagger Specifications . . . 8

5 Use Of Command Line Remote Invocation (rinvoker) Validation Tools 8

5.1 rinvoker Seed Features – Commands – Parameters – Arguments . . . 8

5.1.1 rinvoker.py Seed Features – Commands . . . 9

5.1.2 rinvoker.py Seed Features – Parameters . . . 9

5.1.3 rinvoker.py Seed Features – Arguments . . . 9

5.2 rinvokerPetstore.py Example 10

6 Scenarios Invocation Validation Tools **10**

6.1 Model Of Invoke – Specification, Verification And Reporting – Scenarios 10

6.1.1 Scenario Specification For Sequences Of Invocations 10

6.2 opScn-Seed (Remote Operation Scenarios) – Commands – Paramters – Arguments 11

6.2.1 opScn Seed Features – Commands 11

6.2.2 OpScn Outputs And Reportings 11

7 Complete Invoker-Applications Development **12**

7.1 Invoker-Apps Development Model 12

7.1.1 Invoker-Apps Can Easily Build On unisos.mmwsIcm Capabilities 12

III Uses Of RO-Verifier For Public Web Services **13**

8 Public Uses Of RO-Verifier **13**

8.1 Specific Web Service Verifcation Scenarios 13

8.1.1 PetStore Verifcation Scenarios 13

8.2 Generalized Web Service Verifcations 13

9 Short Comings Of OpenAPI/Swagger **13**

10 Alternatives To RO-Verifier **14**

10.1 postman 14

10.2 Assertible 14

10.3 OWASP 14

10.3.1 OWASP Zed Attack Proxy (ZAP) 14

10.4 Miscellaneous None Swagger Based Invocation And Verification Tools 14

10.5 Miscellaneous Open-Source Swagger Based Invocation And Verification Tools 15

10.6 Sample Vulnerable Web Applications 15

10.7 Miscellaneous Proprietary Swagger Based Invocation And Verification Tools 15

Bibliography **15**

List of Figures

1 Swagger Based ICM Web Services Invoker Model 5

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 PLPC-180061: The Over Arching Model Of Web Services

In a sister document, titled:

> **OpenAPI Based Strategies And Tools**
> **For Development, Verification And Use Of Web Services**
> **Based On RO-ICM And RO-Verifier Packages**
> http://www.by-star.net/PLPC/180061 — [?]

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 This Document – PLPC-180057: Invoker Framework (RO-Invoker and RO-Verifier)

In this document we are proposing that a specific invocation and verification software package be used for both general verification and security oriented verification of existing and newly developed 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.

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

RO-Verifier: Generalized Web-Services Invocations And Validation Framework

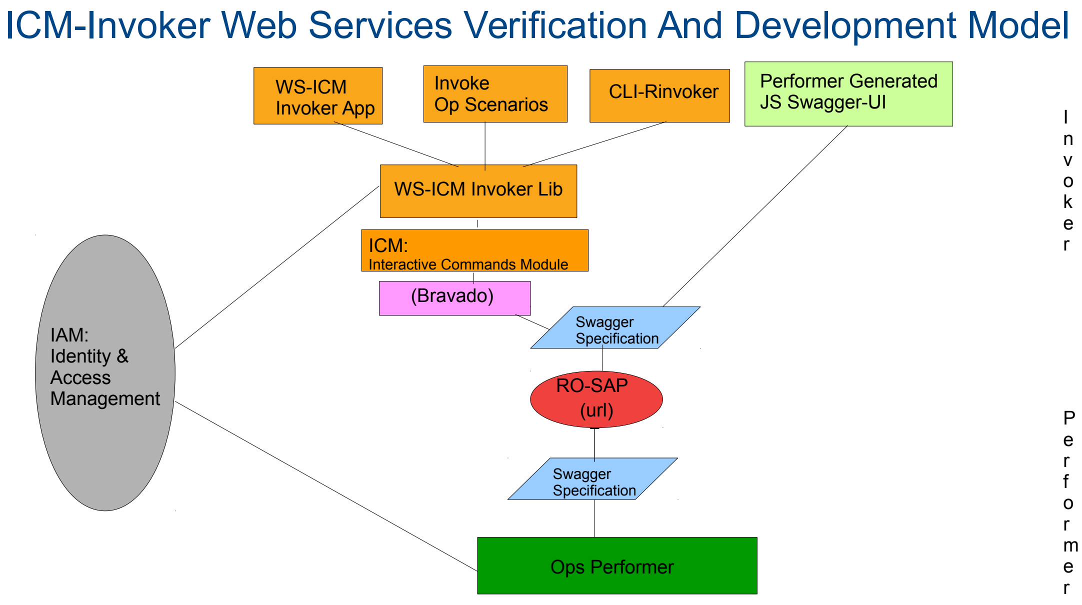

Figure 1: Swagger Based ICM Web Services Invoker Model

3 Overview Of The Web-Services Validation Tools

The Web-Services Validation Tools assume that a correct Swagger Specification is available. These tools do not focus on validating the service specification – there are a variety of tools available for validation of the swagger specifications, that is a different topic. These tools focus on validating the service based on its assumed correct specification.

These tools are invocation tools. They operate purely as the client side and other than verification and use of the service their scope does not extent to the server side (performers).

These tools are purely python based and they are purely Open Source (FOSS, Libre-Halaal). The python packages are available at PyPi and the complete source code is available at github.

You can think of these tools as a layer on top of
Bravado – `https://bravado.readthedocs.io/en/stable/`, `https://github.com/Yelp/bravado`.

Bravado ingests the swagger service specification in json or yaml and maps it to python on the fly. Bravado is a complete replacement to swagger codegen – the traditional code generation phase is eliminated. The Web-Services Validation Tools then create a higher level of convenience for invoking the operations specified in the service specification.

The Web-Services Validation Tools fall into two broad categories of:

1. Command Line Service Invocation Tools – described in Section 3.1.

2. Python Scenarios Service Invocation Tools – described in Section 3.2.

3.1 Command-Line Remote Invocations – rinvoker.py

Based on a given a swagger specification, rinvoker.py maps the json/yaml specification to python (using Bravado) and then python functions corresponding to remote-operation invocations are exposed as command-line using the ICM package (Interactive Command Modules).

All of this happens on the fly. Given a service-specification and a base service url, all operations become available for invocation at command line.

Pointers to examples and additional details are provided below.

3.2 Python Operations Scenarios – opScnSvc.py

In many situations the command-line interface may not be adequate for operations invocations as the parameters syntax may be complex, and as the results syntax may complex and as operation invocations sequences may be chained or interdependent. For such situation, a high level python interface called Operation-Scenarios (opScn) is provided.

You can then customize remote operation invocations as concrete scenarios (opScn) specifications.

Pointers to examples and additional details are provided below.

4 Installation Of The Web-Services Validation Tools

If you don't have Python 2.7 already installed, execute the following steps.

- Install Python 2.7
- Install pip and upgrade to the latest
- Optionally, if you are familiar with virtualenv, create a virtualenv.

With Python and pip in place, you can now install the unisos.mmwsIcm package.

- pip install unisos.mmwsIcm

All needed dependencies will be installed by just doing that.

4.1 Obtaining (OAuth) Authorization Tokens And Access Credentials

In the OAuth model, to invoke operations, the invoker needs to present tokens that the performer expects.

"Web Services Validation Tools", includes facilities that provide for obtaining oauth2 tokens.

Typically, the swagger specification includes a section such as:

```
"securityDefinitions": {
  "Oauth": {
    "type": "oauth2",
    "description": "Get OAuth access token \n",
    "tokenUrl": "https://xxx/v1/oauth2/accesstoken",
    "flow": "application",
    "scopes": {
      "resource-access": "Get Resource Access"
      }
    }
  }
```

By presentating its credentials to the token service at the “tokenUrl” RO-SAP, the invoker receives a token that it can then use in future invocations.

Obtaining of an oauth token for a given “serviceName” requires the following:

- Token Service Invocations Syntax (details of how the obtainToken operation is invoked)
- tokenUrl (RO-SAP) – typically included in the ”securityDefinitions” section of the swagger file
- accessKey – typically a user-name – Communicated out of band
- secretKey – typically a password -name – Communicated out of band

Use of the oauth2 tokens involves the following steps:

4.1.1 Install The Needed Modules (symCrypt, cryptKeyring)

```
pip install unisos.symCrypt
pip install unisos.cryptKeyring
```

The utility that facilitates invoker’s credentials storage and that facilitates obtaining of tokens is: getTokenWithCryptKeyring-svc.py.

4.1.2 Prepare The KeyRing

The invoker uses the keyring to store accessKey and secretKey for serviceName.

We encrypt the password (secretKey) in the keyring. For this, the interfaces to the keyring needs to be initialized (prepared). This is a one time activity. This preparartion involves:

```
getTokenWithCryptKeyring-svc.py --rsrc="keyring" -i prepare
```

4.1.3 Store The Invoker Credentials In The Keyring

Storing the invoker credentials in the keyring involves:

```
getTokenWithCryptKeyring-svc.py --rsrc="keyring/serviceName/accessKey"
                           --passwdPolicy="prompt" -i cryptPasswdSet
```

4.1.4 Obtain The Token

The following parameters:

- Token Service Invocations Syntax (details of how the obtainToken operation is invoked)
- tokenUrl (RO-SAP) – typically included in the ”securityDefinitions” section of the swagger file

need to be customized for getTokenWithCryptKeyring-svc.py.

Obatining the token for the serviceName and accessKey then involves:

```
getTokenWithCryptKeyring-svc.py --rsrc="keyring/serviceName/accessKey"
                              -i getTokenForKeyringUser > current.token
```

The token is then stored in a file called current.token.

4.2 Run The Web-Services Validation Tools Against The Canonical Petstore

Upon installation of the python packages, the relevant executable python scripts are placed in the bin directory of where Python was installed. On Windows locate the default bin directory by running: ”where rinvoker.py”. On Linux locate the default bin directory by running: ”which -a rinvoker.py”.

Now, verify that you can run the example commands against the petstore.

In `https://pypi.org/project/unisos.mmwsIcm/`, go through the instruction in the sections titled:
“Binaries And Command-Line Examples”
“Remote Invoker (rinvoker-svc.py) Examples”
“Operation Scenario (opScn-svc.py) Examples”

Just run the mentioned commands and verify that you are seeing the mentioned expected outputs.

If these all work right, then you know that the Web-Services Validation Framework has been properly installed and is operational.

4.3 Source Code And Packages Repositories

Complete sources are at:
`https://github.com/bisos-pip/mmwsIcm`

The PYPI page is at:
`https://pypi.org/project/unisos.mmwsIcm`

4.4 Applying This Services Validation Framework To Your Own Swagger Specifications

When you have a formal swagger service specification in place, use of these service validation tools can be very convenient and productive.

Simply follow the instructions that were provided for the canonical petstore and replace petstore’s service specification with your own.

Follow the documentation to build your own scenarios for service specification and use the provided framework to combine multiple scenarios to form regression tests.

5 Use Of Command Line Remote Invocation (rinvoker) Validation Tools

You can invoke a swagger specification’s operations directlly from the command line using the rinvoker.py command.

Typically for each swagger specification you create a customized version based on rinvoker as a seed.

In the following sections we describe common features and parameters and arguments of rinvoker and its derivatives and provide rinvokerPetstore.py as an example.

5.1 rinvoker Seed Features – Commands – Parameters – Arguments

rinvoker is an ICM (Interactive Command Module) and its command syntax is based on the ICM model.

The specific commands, parameters and arguments that are implemented on rinvoker are enumerated below.

5.1.1 rinvoker.py Seed Features – Commands

rinvoker.py commands are enumerated below.

- Cmnd: -i svcOpsList

  ```
  svcOpsList command digests the Service Specification (swagger-file)
  specified on command line as --svcSpec= parameter and produces a
  complete list of ALL remotely invokable commands with their corresponding
  --resource, --opName and url or body arguments.

  Applicable options, parameters and arguments are:

    * Parameter (Mandatory) : --svcSpec=
    * Parameter (Optional)  : --perfSap=  --headers=
  ```

- Cmnd: -i rinvoker

  ```
  rinvoker command invokes the "opName" operation at "resource" with
  specified arguments.

  Applicable options, parameters and arguments are:

    * Parameter (Mandatory) : --svcSpec=  --resource=  --opName=
    * Parameter (Optional)  : --perfSap=  --headers=

    * Arguments             : name=value  bodyStr=jsonStr
  ```

5.1.2 rinvoker.py Seed Features – Parameters

rinvoker.py parameters are enumerated below.

- Parameter: –svcSpec= (url, or swagger-file)
  The swagger file as a url or as a json/yaml file is specified with the –svcSpec= parameter.
- Parameter: –perfSap= (url)
  The Performer Service Access Point Address (perfSap) is specified as a URL with the –perfSap= parameter.
- Parameter: –header= (file)
  Additional headers (e.g., a token) can be included with the –svcSpec= parameter.
- Parameter: –resource= (string, corresponding to SvcSpec)
  The resource to be invoked should be specified with the –resource= parameter
- Parameter: –opName= (string, corresponding to SvcSpec)
  The operation name to be invoked should be specified with the –opName= parameter

5.1.3 rinvoker.py Seed Features – Arguments

rinvoker.py arguments are enumerated below.

- Argument: name=value (string=string corresponding to SvcSpec's URL Params)
- Argument: bodyStr=jsonStr (bodyStr=string corresponding to SvcSpec's Body)

5.2 rinvokerPetstore.py Example

rinvoker allows you to list all possible invocations based on a service specification (swagger file). For example:

```
rinvoker.py --svcSpec="http://petstore.swagger.io/v2/swagger.json" -i svcOpsList
```

rinvoker allows you to fully specify an invocation on command line. For example:

```
rinvoker.py --svcSpec="http://petstore.swagger.io/v2/swagger.json"
     --resource="user" --opName="createUser" -i rinvoke
     bodyStr="{...}"
```

6 Scenarios Invocation Validation Tools

You can specify one or more invocations as a ”scenario”.

Scenarios are python scripts that specify operations and their arguments and expectations of results.

Scenarios allow for the results of operations to be used as arguments of future operations.

6.1 Model Of Invoke – Specification, Verification And Reporting – Scenarios

Invoke Scenarios Are pure python specification of sequence of invocations.

Invoke-Expect Scenarios Are pure python specification of sequence of invocations subject to preparations and post-invoke verification and reporting.

OpInvoke class allows for complete invoke specification and complete results to be fully captured.

6.1.1 Scenario Specification For Sequences Of Invocations

In pure python you can specify invocation of each operation, for example:

```
thisRo = ro.Ro_Op(
   svcSpec=petstoreSvcSpec,
   perfSap=petstoreSvcPerfSap,
   resource="pet",
   opName="getPetById",
   roParams=ro.Ro_Params(
      headerParams=None,
      urlParams={ "petId": 1},
      bodyParams=None,
      ),
   roResults=None,
   )
rosList.opAppend(thisRo)
```

Validation And Reporting Of Invokations

Building on the previously mentioned Operation Specification, in pure python you can the specify Operation Expectations, for example:

```
thisExpectation = ro.Ro_OpExpectation(
   roOp=thisRo,
   preInvokeCallables=[sleep1Sec],
   postInvokeCallables=[ verify_petstoreSvcCommonRo, ],
   expectedResults=None,
   )
roExpectationsList.opExpectationAppend(thisExpectation)
```

preInvokeCallables(ro.Ro_OpExpectation) can include a function that initializes the DB or sleepFor1Sec.

postInvokeCallables(ro.Ro_OpExpectation) can include a function that verifies the result was as expected and then reports success or failure.

6.2 opScn-Seed (Remote Operation Scenarios) – Commands – Paramters – Arguments

Commands drived from the opScn seed are ICMs (Interactive Command Module) and their command syntax are based on the ICM model.

6.2.1 opScn Seed Features – Commands

opScn-seed provides the following commands and parameters:

- Cmnd: -i roListInv

  ```
  roListInv command serially invokes the list of ro.Ro_Op() opersations
  specified in the loaded scenario files.

  roListInv displays the invocation and its results. But does not do any verifications.

  Applicable options, parameters and arguments are:

    * Parameter (Mandatory) : --load=
  ```

- Cmnd: -i roListExpectations

  ```
  roListExpectations command serially invokes the list of ro.Ro_OpExpectation()
  specified in the loaded scenario files.

  roListExpectations displays the invocation and its results and
  additionally runs the list of preInvokeCallables and
  postInvokeCallables.

  postInvokeCallables can include functions that verify the results of the invocation
  were as expected.

  Applicable options, parameters and arguments are:

    * Parameter (Mandatory) : --load=
  ```

6.2.2 OpScn Outputs And Reportings

The output format is:

```
 * ->:: Invoke Request
 * <-:: Invoke Response
 * ==:: Invoke Validation (SUCCESS or FAILURE)
```

Additional information for each is include with ”***” tags.

This output format can then be used in outline or org-mode.

7 Complete Invoker-Applications Development

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

7.1 Invoker-Apps Development Model

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

7.1.1 Invoker-Apps Can Easily Build On unisos.mmwsIcm Capabilities

The following modules:

- Bravado does invoker code-generation on the fly.
- unisos.mmwsIcm.opInvoke – Abstracts invoke-specifications
- unisos.mmwsIcm.wsInvoker – Allows for invokation and verification of opInvoke

provide a consistent framework for Invoker-Apps development.

Part III

Uses Of RO-Verifier For Public Web Services

8 Public Uses Of RO-Verifier

The RO-Verifier has been used to validate and verify a number of Web Services based on their swagger files.

Generally speaking, uses of RO-Verifier fall into two categories:

Specific Service Verifications: Where we take a web service and its swagger file and custmize a set of OpScenarios for verificatin of that web service.

Generalized Service Verifications: Where we apply a set of exisiting common OpScenarios and apply them to a repository of web services.

8.1 Specific Web Service Verifcation Scenarios

8.1.1 PetStore Verifcation Scenarios

Verifcation scenarios for the PetStore web service are maintained at:
`https://github.com/bxexamples/roVerifier-petstore`

8.2 Generalized Web Service Verifcations

A set of scripts that pass all swagger files availble at:
`https://github.com/APIs-guru/openapi-directory` through the rinvoker.py are available at:
`https://github.com/bxexamples`

The strategy and approach that we are proposing here heavily counts on industry convergences on:

1. OpenAPI/Swagger as an industry convergence point for web services specifications
2. Python as an industry convergence point for convenient development of scenarios for invocation of web services

Given a 5 year time horizon, there is hardly any risk with (2). However there are some risks with (1).

While it is clearly the case that OpenAPI/Swagger is currently the industry convergence point for web services specifications, that can change.

9 Short Comings Of OpenAPI/Swagger

At this time OpenAPI/Swagger ecosystem is rich with a large number of code-generators and interpreters and is growing. But, there are some fundamental flaws with OpenAPI/Swagger approach, these include:

- Swagger does not have any facilities for specifying the expected order of invocation of operations. The expected order of invocations is often expressed as flow graphs. The scenarios model of RO-Verifier can be used to augment the Swagger specification in this regard.
- Swagger is un-necessarily married to https/http
- Swagger does not support multiple remote operations protocols

- Swagger does not well express the remote operations model and terminology
- Swagger does not well abstract encoding rules beyond json and xml
- Swagger does not play well with IoT (too inefficient at protocol level)

These deficiencies question long term viability of OpenAPI/Swagger.

Competitors such as Protocol Buffers and Graph APIs have already appeared. Better remote operations specification standards will likely emerge within this 5 year horizon.

Nevertheless, OpenAPI/Swagger is clearly the right choice right now.

And adapting this very same basic approach to other points of convergence of remote operations specification standards is possible.

Our choice of abstracting to Remote-Operations services at the scenarios layers assists with preservation of investments when underlying providers change.

10 Alternatives To RO-Verifier

Here we mention availability of various tools related to Web Services API Verification.

Flexibility of pure python scenarios creation makes RO-Verifier the most convivial tool. Simplicity, elegance, open-source-ness and power of RO-Verifier framework distinguishes it from other tools in this space.

10.1 postman

Postman `https://www.getpostman.com/` is one of the most used tools—if not the most used—when it comes to REST API troubleshooting. Postman is a Chrome app tool used, in its simplest implementation, for executing requests and validating responses. The popularity of Postman is well deserved, as it delivers simple to complex features for everyday users to quickly test HTTP based requests.

Postman is primarily UI based – not batch oriented.

There are some Swagger plug-ins for postman. The end result becomes too heavy and convoluted.

10.2 Assertible

`https://assertible.com/` Introduces itself as: "Quality Assurance for the Web". Assertible tests and monitors the executions of your web requests and allows for assertions using JavaScript.

It is primarily a web based tool and not open-source.

10.3 OWASP

10.3.1 OWASP Zed Attack Proxy (ZAP)

Security proxies could also be employed to watch the traffic and/or perform scanning themselves OWASP ZAP

10.4 Miscellaneous None Swagger Based Invocation And Verification Tools

curl: cURL A basic command line tool for making a web request and retrieving the response.

REST-assured: `http://rest-assured.io/` A Java based library. Does not have any swagger integration

HTTPMaster: `https://www.joecolantonio.com/12-open-source-api-testing-tools-rest-soap-services/`

10.5 Miscellaneous Open-Source Swagger Based Invocation And Verification Tools

Rapid7 AppSpider: https://blog.rapid7.com/2016/05/18/automating-my-restful-web-services-scanning-final
Security scanning tool that can ingest Swagger files.

Qualys Web Application Scanning (WAS): https://www.computerweekly.com/blog/CW-Developer-Network/Qualys-ups-security-automation-with-a-bit-of-Swagger Security scanning tool that can ingest Swagger files.

Swagger-EZ: https://rhinosecuritylabs.com/application-security/simplifying-api-pentesting-swagger-fil

Integrating Security Tools: Potential python tools:
https://github.com/redhat-cip/restfuzz
https://github.com/ant4g0nist/Susanoo
https://github.com/flipkart-incubator/astra

10.6 Sample Vulnerable Web Applications

OWASP DevSlop https://www.owasp.org/index.php/OWASP_DevSlop_Project

OWASP Juice Shop Project

https://payatu.com/tiredful-api-vulnerable-rest-api-app/

https://github.com/rapid7/hackazon

https://github.com/wishtack/wishtack-websheep

10.7 Miscellaneous Proprietary Swagger Based Invocation And Verification Tools

Microsoft REST-ler: Automatic Intelligent REST API Fuzzing
Microsoft has developed a tool (unreleased) and published a paper on an approach
https://www.microsoft.com/en-us/research/publication/rest-ler-automatic-intelligent-rest-api-fu
[2]

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] Vaggelis Atlidakis, Patrice Godefroid, and Marina Polishchuk. Rest-ler: Automatic intelligent rest api fuzzing. Technical Report MSR-TR-2018-11, April 2018.