BACKGROUND

Users have been asking for better ways to scan their RESTful APIs with WebInspect. Currently, the most common workflow is to capture HTTP traffic in a proxy while manually exercising the API. While this approach works, it is time consuming and error prone.

Recently, REST API documentation formats have been maturing. WADL has been around for a while, but lack of adoption has prevented it from gaining traction, and lack of standardization means that vital information about request payloads may be absent. Swagger on the other hand has greatly increased in popularity. The community and toolset growing around Swagger are pushing it to become the de facto REST API documentation standard. The adoption of Swagger v2 as the OpenAPI standard likely makes this a foregone conclusion. It means that Swagger will have staying power and demand for integration and support for Swagger will only increase. Furthermore, a Swagger API definition contains enough information to accurately describe each endpoint along with the expected payload. With Swagger, we can finally automate the cumbersome task of scanning a REST API with WebInspect.

WISwag Tool

At HPE Fortify, we have solved this problem through our WISwag tool. This command line tool will parse a Swagger API definition and convert it into a format that WebInspect understands.

To get started download WISwag.zip from HPE Live networks marketplace.

https://hpln.hpe.com/contentoffering/wiswag-for-webinspect

To install, simply unzip WISwag.zip to a folder on your computer and then run WISwag from the command line. Though you will see WebInspect dependencies in the WISwag.zip archive, the tool is meant to run standalone. Do not unzip the WISwag.zip file into your WebInspect install location as it will break your existing WebInspect installation.

Process Overview

The process for scanning a REST API is as follows.

Stage	Description
1.	Get the REST API definition from your development team.
2.	Do one of the following: l If you do not have a settings file, use the WISwag.exe tool to convert the REST API definition into a Fortify WebInspect settings file. This option also generates a workflow macro and custom parameter rules, and embeds them in the settings file. See "Converting the API Definition to a Settings File" on the next page. l If you have a settings file, use the WISwag.exe tool to convert the REST API definition into a Fortify WebInspect workflow macro. See "Converting the API Definition to a Macro" on the next page.
3.	Use the command line interface to conduct a scan of your REST API using the webmacro or settings file..

WISwag.exe Parameters

The WISwag.exe parameters are defined in the following table.

Parameter	Description
-i	Specifies the input file and location. The input file can be an API definition file or a configuration file. To override default settings and control which endpoints are processed, use a configuration file. For more information, see "Using a Configuration File" on the next page. The location can be a URL or a local file. **Examples:** http://mysite.com/api_def.json C:/myapi.json
-m	Specifies the macro file to create using the .webmacro extension.
-s	Specifies the settings file to create using the .xml extension.

Converting the API Definition to a Macro

You can convert the API definition into a Fortify WebInspect workflow macro that you can then use to scan your REST API. To do this, enter the following command at the command line prompt:

WISwag.exe –i http://<*input_file_location*> -m ./<*macro_filename*>.webmacro

Afterward, open the macro in the Web Macro Recorder tool and explore its contents.

Converting the API Definition to a Settings File

You can convert the API definition into a Fortify WebInspect settings file. The settings file is configured to run as Audit Only and contains a workflow macro and custom parameter rules derived from the

REST API definition.

To do this, enter the following command at the command line prompt:

WISwag.exe –i http://<*input_file_location*> -s ./<*settings_filename*>.xml

Open the scan settings in Fortify WebInspect and explore the contents. You should find that a workflow macro and custom parameter rules are already defined.

Using a Configuration File

If you use a REST API definition file to create the workflow macro and settings file, then the macro and settings file will include only default values and settings. For more advanced control over the HTTP requests generated by the WISwag tool, you can pass a configuration file to the WISwag tool instead of a REST API definition. This advanced configuration is useful in cases where control over specific operations or parameters is required. For example, you might need to be exclude certain operations, such as logout or delete operations, from a Fortify WebInpsect scan. You can accomplish this by listing the operation IDs in the 'excludeOperations' property. Operation IDs are defined in the REST API definition. Sometimes a white-list approach is easier when only a few operations need to be tested. In this case, use the 'includeOperations' list.

Configuration File Format

The configuration file has the following format:

```
{
apiDefinition : 'http://mysite.com/api_def.json', /* can also be a local
file (ex. C:/myapi.json) */
host : 'localhost:8080', /* replace the host in every generated request */  schemes : ['https',
'http'], /* generate output for both of these schemes */  preferredContentType :
'application/json', /* if given a choice, prefer json
*/
excludeOperations : [ 'logoutUser', 'deleteUser' ], /* generate no output


for these operations */ parameterRules :
[
{
name : 'userId', value : 42, location : 'path', type : 'number',
includeOperations : ['createNewUser', 'getUser'] /* only apply this rule
to these operations */
},
{
name : 'file',
value : 'my file payload', filename : 'myfile.txt', location : 'body',
type : 'file'
},
{
name : 'Authorization',
value : 'Basic QWxhZGRpbjpPcGVuU2VzYW1l',  location : 'header',
inject : true /* add this header to every generated request */
}
]
}
```

Configuration Properties

The configuration properties are described in the following table.

Property	Required/ Optional	Description
apiDefinition	Required	Identifies the URL or file location of a supported REST API definition.
host	Optional	Overrides the host in the REST API definition. **Example:** localhost:8080
schemes	Optional	Overrides the schemes defined in the REST API definition, expressed as an array of schemes.
		Example: ['http','https'] If defined, a series of requests will be generated for each scheme. Otherwise, a series of requests will only be generated for the first scheme listed in the REST API definition.
preferredContentType	Optional	Sets the preferred content type of the request payload. If preferredContentType is in the list of supported content types for an operation, the generated request payload will be of that type. Otherwise, the first content type listed in an operation will be used.
excludeOperations	Optional	Defines a black-list of operation IDs that should be excluded from the output, expressed as an array of operation IDs. **Example:** ['operation1', 'operation2', 'operationN']
includeOperations	Optional	Defines a white-list of operation IDs that should be included in the output, expressed as an array of operation IDs . **Example:** ['operation1', 'operation2', 'operationN']

parameterRules	Optional	Defines specific values for a parameter when the default value is not appropriate or when the parameter is not defined in the API definition. **Example:** A parameter, such as an authorization header which is not defined in the API definition, needs to be injected into every request. The property is expressed as an array of 'parameterRule' objects. The 'parameterRule' objects are described in "Parameter Rule Objects" on the next page.

Parameter Rule Objects

The 'parameterRule' objects are described in the following table.

Object	Required/ Optional	Description
name	Required	Specifies the parameter name to match. To override a property when you have a name conflict, specify the type of object from the API definition in front of the parameter name, separated by a slash in the format '<*type_of_object*>/<*parameter_name*>'. For example, if you have a parameter named "name" and a nested parameter also named "name", you must specify the type of object for the nested parameter as shown below. { name : 'name', value : 'Romeo', location : 'body', type : 'string', includeOperations : ['addPet'] }, { name : 'tag/name', value : 'Juliet', location : 'body', type : 'string', includeOperations : ['addPet'] },
value	Required	Specifies the parameter value to substitute or inject.

location	Optional	Identifies the parameter location to match. Options are: • 'body' • 'header' • 'path' • 'query' • 'any' The default is 'any' and matches all locations.
type	Optional	Identifies the parameter type to match. Options are: • 'number' • 'boolean' • 'string'
		• 'file' (See filename below.) • 'date' • 'any' The default is 'any' and matches all types.
filename	Optional	Replaces the filename attribute of a matching multipart or form file entry. Valid only if type is 'file'.
inject	Optional	Replaces parameter values. Options are: • true - injects the parameter in the specified location regardless of whether a matching name or type is found. • false - replaces only parameter values that match the specified name, location, and type.
base64Decode	Optional	Specifies whether 'value' is base64 encoded binary data. Options are: • true - 'value' is assumed to be base64 encoded binary data and will be decoded into a byte array when inserted into a generated HTTP request. • false - 'value' is not base64 encoded binary data. The default is false.
includeOperations	Optional	Applies this parameter rule to the operation IDs in the list, expressed an array of operation IDs. **Example:** ['operation1', 'operation2', 'operationN']

excludeOperations	Optional	Does not apply this parameter rule to the operation IDs in the list, expressed as an array of operation IDs. **Example:** ['operation1', 'operation2', 'operationN']