D6.6

Open Source Tool for Automated Trust Verification

Document Identification	
Date	15.02.2019
Status	Draft
Version	Version 1.3

Related WP	WP 3, 4, 5, 6, 8, 9	**Related Deliverable(s)**	D6.5, D5.5, D6.1, D6.2, D2.5
Lead Authors	TUG	**Dissemination Level**	PU
Lead Participants	TUG	**Contributors**	TUG, FHG, DTU
Reviewers	TUBITAK, CORREOS		

This document is issued within the frame and for the purpose of the LIGHT*est* project. LIGHT*est* has received funding from the European Union's Horizon 2020 research and innovation programme under G.A. No 700321.

1. Executive Summary

This document describes how to obtain, build, and maintain the source code for the automated trust verifier (ATV) component.

2. Document Information

2.1 Contributors

Name	Partner
Stefan More	TUG
Georg Wagner	TUG
Olamide Omolola	TUG
Lukas Alber	TUG
Sven Wagner	FHG
Sebastian Mödersheim	DTU

2.2 History

Version	Date	Author	Changes
0.1	11/02/2019	Stefan More	Initial Version
1.0	12/02/2019	Stefan More, Lukas Alber, Peter Lipp	Additional Information from internal feedback
1.1	14/02/2019	Stefan More	Extended document based on reviewer feedback
1.2	14/02/2019	Olamide Omolola	Added API specification and additional information.
1.3	15/02/2019	Olamide Omolola	Implemented reviewer feedback

3. Table of Contents

4. Table of Acronyms

ATV – Automated Trust Verifier

ET – Electronic Transaction

TPL – Trust Policy Language

5. Introduction

The automated trust verifier (ATV) is the central component of LIGHTest. It takes a trust policy (TP) and an electronic transaction (ET) as input and uses other LIGHTest components to verify the ET. The LIGHTest Consortium develops a prototype implementation of the ATV component and plans to provide it to the public under an open source license.

Since the ATV is actively verifying trust policies in the TPL language, a component directly used by the ATV is the Trust Policy interpreter. This component is also mentioned in this document.

The ATV is tested systematically via unit tests. In addition, integration tests are performed in WP8 [1]. After completion, the ATV will be delivered as a library to LIGHTest pilot partners for integration into their pilots (WP9). In addition, other components access the LIGHTest ATV via a REST API. Furthermore, we provide a basic GUI for demonstration purposes.

The ATV prototype developed by the LIGHTest consortium is currently under active development. The scope of this document is therefore limited to the development version.

6. Components

6.1 Technical Infrastructure

The details of the technical infrastructure can be found in D8.1 [1]. This deliverable describes the technical infrastructure for source code hosting as well as automated deployment methods. We use GIT as source code repository and Nexus to manage our software artefacts. The default operating system used for development is Ubuntu Linux.

GIT [2] is a version control system for tracking changes in computer files and coordinating work on those files among multiple people. It is used for source code management in software development, but it can be used to keep track of changes in any set of files.

6.2 Access to Source Code and other Artefacts

The programming language for the ATV is JAVA 8 and Maven [3] is used to build the source code. Maven retrieves dependencies; you need to give Maven access to artefacts hosted on their respective Maven repository in the IAIK GitLab repository hosted at https://extgit.iaik.tugraz.at. This can be done by creating a *personal access token* in GitLab and adding it to your maven *config.xml*.

A access token can be created in GitLab in
Profile → Settings → Access Tokens → Personal Access Tokens

Maven's global settings.xml is located in *~/.m2/settings.xml* and needs to contain the following:

```
<?xml version="1.0" encoding="UTF-8"?>

<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
                            http://maven.apache.org/xsd/settings-1.0.0.xsd">

    <servers>

      <server>
        <id>gitlab-maven</id>
        <configuration>
          <httpHeaders>
            <property>
              <name>Private-Token</name>
              <value>YOUR_TOKEN_HERE</value>
            </property>
          </httpHeaders>
        </configuration>
```

```
        </server>
    </servers>
</settings>
```

6.2.1 Relevant Artefacts

To build the ATV, the following dependency is needed and therefore access to its repositories needs to be configured:

Component name: TrustPolicyInterpreter
GIT repository: https://extgit.iaik.tugraz.at/LIGHTest/trustpolicyinterpreter
Maven repository: https://extgit.iaik.tugraz.at/api/v4/projects/1050/packages/maven

6.3 Obtaining the Source Code for the Automated Trust Verifier

The source code of the ATV can be obtained via the following git command:

```
git clone git@extgit.iaik.tugraz.at:LIGHTest/AutomaticTrustVerifier.git
```

and receive updated versions via the following git command:

```
git pull
```

6.4 Obtaining the Source Code for the Trust Policy Interpreter

The source code of the Trust Policy Interpreter can be obtained via the following git command:

```
git clone git@extgit.iaik.tugraz.at:LIGHTest/trustpolicyinterpreter.git
```

6.5 Getting the right Revision

The development for this version takes place in the branch ‘master’. The branch can be changed via the command

```
git checkout master
```

to get the sources of that particular branch.

6.6 Building the Source Code

The project is written in version 1.8 of the Java programming language and uses several open source libraries (a detailed list is given in the project’s *pom.xml*).

The project uses Maven as build system and can be built and tested using the command

```
mvn verify
```

To only build the project, use the command

```
mvn build
```

As an alternative to building the source code on the command line, you can also use your favourite IDE.

Since Maven is also taking care of dependencies, you need to setup access to several GitLab Maven repositories first to build the ATV development version with the Maven commands described above.

6.7 Running the ATV

Since the ATV is a library, there are multiple ways of accessing it.

For demonstration purposes, there is a simple GUI in:

```
at/tugraz/iaik/lightest/verifier/Main.java
```

Provided the Java classpath is setup, the GUI can be started using the following command inside *target/classes*:

```
java at.tugraz.iaik.lightest.verifier.Main
```

As an easier alternative, you can also use your favourite IDE.

6.8 Trust Policy Format

A Trust Policy must be given to the ATV in TPL format, as specified in D2.5 [4]. The D2.5 contains numerous examples on creating Trust Policies

6.9 Electronic Transaction Format

An Electronic Transaction Format describes the contents of an electronic transaction in machine-readable format. It resides within the Electronic Transaction. An example of the Electronic Transaction Format written in XML is given below:

```
1. <?xml version="1.0"?>
2. <Transaction>
3.   <Creator>
4.     <FirstName>John</FirstName>
5.     <LastName>Mark</LastName>
6.     <ContactNo>1234567890</ContactNo>
7.     <Email>johnmark@xyz.com</Email>
8.     <Address>
```

```
            <City>Bangalore</City>
            <State>Karnataka</State>
            <Zip>560212</Zip>
        </Address>
    </Creator>
    <Contract>
        <Product>Laptop</Product>
        <LastName>Patil</LastName>
        <ContactNo>1234567890</ContactNo>
        <cost>3000</cost>
    </Contract>
</Transaction>
```

6.10 API Specification

The API document for the ATV is located at https://extgit.iaik.tugraz.at/LIGHTest/atv-mockup/blob/master/atv.yaml. This is based on the OpenAPI specification [5]

A live mock-up of the API document is available at http://virtserver.swaggerhub.com/EULIGHTest/atv/1.0.0/ and it was created using an open-source virtual server from Swaggerhub

An example of a query to the live mock-up is

```
curl -X GET "http://virtserver.swaggerhub.com/EULIGHTest/atv/1.0.0/verifyInstance/" -H "accept: application/json"
```

The REST API specification for the ATV is given below:

```
swagger: '2.0'
info:
  version: 1.0.0
  title: Verification Instance
  description: A sample API for verification
  termsOfService: 'http://helloreverb.com/terms/'
  contact:
    name: IAIK Lightest Team
    email: lightest@iaik.tugraz.at
    url: 'http://iaik.tugraz.at'
  license:
    name: MIT
    url: 'http://opensource.org/licenses/MIT'
# host: iaik.tugraz.at
# basePath: /api/atv
# schemes:
#   - http
consumes:
```

```
- application/json
produces:
- application/json
paths:
/verifyInstance:
  get:
    description: >-
      Returns all verification instances from the system that the user has
      access to
    operationId: findInstances
    produces:
      - application/json
      - application/xml
      - text/xml
      - text/html
    parameters:
      - name: tags
        in: query
        description: tags to filter by
        required: false
        type: array
        items:
          type: string
        collectionFormat: csv
      - name: limit
        in: query
        description: maximum number of results to return
        required: false
        type: integer
        format: int32
    responses:
      200:
        description: instance response
        schema:
          type: array
          items:
            $ref: '#/definitions/instance'
      default:
        description: unexpected error
        schema:
          $ref: '#/definitions/errorModel'

  post:
    description: Creates a new verifyInstance in the store.
    operationId: addInstance
```

```
    produces:
      - application/json
    parameters:
      - name: instance
        in: body
        description: Instance to create
        required: true
        schema:
          $ref: '#/definitions/newInstance'
    responses:
      '200':
        description: instance response
        schema:
          $ref: '#/definitions/instanceVerification'
      default:
        description: unexpected error
        schema:
          $ref: '#/definitions/errorModel'
'/verifyInstance/{id}':
  get:
    description: Returns a instance based on a single ID
    operationId: findInstanceById
    produces:
      - application/json
      - application/xml
      - text/xml
      - text/html
    parameters:
      - name: id
        in: path
        description: ID of instance to fetch
        required: true
        type: integer
        format: int64
    responses:
      '200':
                description: instance response
                schema:
                  $ref: '#/definitions/instanceVerification'
              default:
                description: unexpected error
                schema:
                  $ref: '#/definitions/errorModel'
            delete:
              description: deletes a instance based on the ID supplied
```

```
      operationId: deleteInstance
      parameters:
        - name: id
          in: path
          description: ID of instance to delete
          required: true
          type: integer
          format: int64
      responses:
        '204':
          description: instance deleted
        default:
          description: unexpected error
          schema:
            $ref: '#/definitions/errorModel'
  definitions:
    instance:
      type: object
      required:
        - id
      properties:
        id:
          type: integer
          format: int64
          example: 42
    newInstance:
      type: object
      required:
        - id
        - policy
        - transaction
      properties:
        id:
          type: integer
          format: int64
          example: "<id to store the result under here>"
        policy:
          type: string
          format: binary
          example: "<policy in tpl here>"
        transaction:
          type: string
          format: binary
          example: "<transaction zip here>"
    instanceVerification:
```

```
      type: object
      required:
        - id
        - verificationResult
        - result
      properties:
        id:
          type: integer
          format: int64
          example: 42
        verificationResult:
          type: string
          example: "trusted, because of ..."
        result:
          type: integer
          format: int64
          example: 10
    errorModel:
      type: object
      required:
        - code
        - message
      properties:
        code:
          type: integer
          format: int32
          example: 500
        message:
          type: string
          example: "something went wrong"
# Added by API Auto Mocking Plugin
host: virtserver.swaggerhub.com
basePath: /EULIGHTest/atv/1.0.0
schemes:
 - https
 - http
```

7. References

[1] The LIGHTest Project, „Technical Infrastructure for Development and Testing,“ 2018.

[2] GIT, „Getting Started - Git Basics,“ [Online]. Available: https://git-scm.com/book/en/v2/Getting-Started-Git-Basics. [Zugriff am 15 February 2019].

[3] Apache, „Apache Maven Project,“ [Online]. Available: https://maven.apache.org/. [Zugriff am 14 February 2019].

[4] LIGHTest, „LIGHTest,“ [Online]. Available: https://www.lightest.eu//static/deliverables/D2.5.pdf. [Zugriff am 14 February 2019].

[5] Swagger, „OpenAPI Specification,“ [Online]. Available: https://swagger.io/specification/v2/). [Zugriff am 14 February 2019].

8. Project Description

LIGHTest project to build a global trust infrastructure that enables electronic transactions in a wide variety of applications

An ever increasing number of transactions are conducted virtually over the Internet. How can you be sure that the person making the transaction is who they say they are? The EU-funded project LIGHTest addresses this issue by creating a global trust infrastructure. It will provide a solution that allows one to distinguish legitimate identities from frauds. This is key in being able to bring an efficiency of electronic transactions to a wide application field ranging from simple verification of electronic signatures, over eProcurement, eJustice, eHealth, and law enforcement, up to the verification of trust in sensors and devices in the Internet of Things.

Traditionally, we often knew our business partners personally, which meant that impersonation and fraud were uncommon. Whether regarding the single European market place or on a Global scale, there is an increasing amount of electronic transactions that are becoming a part of peoples everyday lives, where decisions on establishing who is on the other end of the transaction is important. Clearly, it is necessary to have assistance from authorities to certify trustworthy electronic identities. This has already been done. For example, the EC and Member States have legally binding electronic signatures. But how can we query such authorities in a secure manner? With the current lack of a worldwide standard for publishing and querying trust information, this would be a prohibitively complex leading to verifiers having to deal with a high number of formats and protocols.

The EU-funded LIGHTest project attempts to solve this problem by building a global trust infrastructure where arbitrary authorities can publish their trust information. Setting up a global infrastructure is an ambitious objective; however, given the already existing infrastructure, organization, governance and security standards of the Internet Domain Name System, it is with confidence that this is possible. The EC and Member States can use this to publish lists of qualified trust services, as business registrars and authorities can in health, law enforcement and justice. In the private sector, this can be used to establish trust in inter-banking, international trade, shipping, business reputation and credit rating. Companies, administrations, and citizens can then use LIGHTest open source software to easily query this trust information to verify trust in simple signed documents or multi-faceted complex transactions.

The three-year LIGHTest project starts on September 1st and has an estimated cost of almost 9 Million Euros. It is partially funded by the European Union's Horizon 2020 research and innovation programme under G.A. No. 700321. The LIGHTest consortium consists of 14 partners from 9 European countries and is coordinated by Fraunhofer-Gesellschaft. To reach out beyond Europe, LIGHTest attempts to build up a global community based on international standards and open source software.

The partners are ATOS (ES), Time Lex (BE), Technische Universität Graz (AT), EEMA (BE), G+D (DE), Danmarks tekniske Universitet (DK), TUBITAK (TR), Universität Stuttgart (DE), Open Identity Exchange (GB), NLNet Labs (NL), CORREOS (ES), Ubisecure (FI) and University of Piraeus Research Center (GR). The Fraunhofer IAO provides the vision and architecture for the project and is responsible for both, its management and the technical coordination.
The Fraunhofer IAO provides the vision and architecture for the project and is responsible for both, its management and the technical coordination.