OGC Testbed-14

Secure Client Test Engineering Report

Table of Contents

Publication Date: 2019-03-06

Approval Date: 2018-12-13

Submission Date: 2018-11-22

Reference number of this document: OGC 18-030

Reference URL for this document: http://www.opengis.net/doc/PER/t14-D003

Category: Public Engineering Report

Editor: Sara Saeedi

Title: OGC Testbed-14: Secure Client Test Engineering Report

OGC Engineering Report

COPYRIGHT

WARNING

This document is not an OGC Standard. This document is an OGC Public Engineering Report created as a deliverable in an OGC Interoperability Initiative and is not an official position of the OGC membership. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an OGC Standard. Further, any OGC Engineering Report should not be referenced as required or mandatory technology in procurements. However, the discussions in this document could very well lead to the definition of an OGC Standard.

LICENSE AGREEMENT

Permission is hereby granted by the Open Geospatial Consortium, ("Licensor"), free of charge and subject to the terms set forth below, to any person obtaining a copy of this Intellectual Property and any associated documentation, to deal in the Intellectual Property without restriction (except as set forth below), including without limitation the rights to implement, use, copy, modify, merge, publish, distribute, and/or sublicense copies of the Intellectual Property, and to permit persons to whom the Intellectual Property is furnished to do so, provided that all copyright notices on the intellectual property are retained intact and that each person to whom the Intellectual Property is furnished agrees to the terms of this Agreement.

If you modify the Intellectual Property, all copies of the modified Intellectual Property must include, in addition to the above copyright notice, a notice that the Intellectual Property includes modifications that have not been approved or adopted by LICENSOR.

THIS LICENSE IS A COPYRIGHT LICENSE ONLY, AND DOES NOT CONVEY ANY RIGHTS UNDER ANY PATENTS THAT MAY BE IN FORCE ANYWHERE IN THE WORLD. THE INTELLECTUAL PROPERTY IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE DO NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE INTELLECTUAL PROPERTY WILL MEET YOUR REQUIREMENTS OR THAT THE OPERATION OF THE INTELLECTUAL PROPERTY WILL BE UNINTERRUPTED OR ERROR FREE. ANY USE OF THE INTELLECTUAL PROPERTY SHALL BE MADE ENTIRELY AT THE USER’S OWN RISK. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR ANY CONTRIBUTOR OF INTELLECTUAL PROPERTY RIGHTS TO THE INTELLECTUAL PROPERTY BE LIABLE FOR ANY CLAIM, OR ANY DIRECT, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM ANY ALLEGED INFRINGEMENT OR ANY LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR UNDER ANY OTHER LEGAL THEORY, ARISING OUT OF OR IN CONNECTION WITH THE IMPLEMENTATION, USE, COMMERCIALIZATION OR PERFORMANCE OF THIS INTELLECTUAL PROPERTY.

This license is effective until terminated. You may terminate it at any time by destroying the Intellectual Property together with all copies in any form. The license will also terminate if you fail to comply with any term or condition of this Agreement. Except as provided in the following sentence, no such termination of this license shall require the termination of any third party end-user sublicense to the Intellectual Property which is in force as of the date of notice of such termination. In addition, should the Intellectual Property, or the operation of the Intellectual Property, infringe, or in LICENSOR’s sole opinion be likely to infringe, any patent, copyright, trademark or other right of a third party, you agree that LICENSOR, in its sole discretion, may terminate this license without any compensation or liability to you, your licensees or any other party. You agree upon termination of any kind to destroy or cause to be destroyed the Intellectual Property together with all copies in any form, whether held by you or by any third party.

Except as contained in this notice, the name of LICENSOR or of any other holder of a copyright in all or part of the Intellectual Property shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Intellectual Property without prior written authorization of LICENSOR or such copyright holder. LICENSOR is and shall at all times be the sole entity that may authorize you or any third party to use certification marks, trademarks or other special designations to

indicate compliance with any LICENSOR standards or specifications.

This Agreement is governed by the laws of the Commonwealth of Massachusetts. The application to this Agreement of the United Nations Convention on Contracts for the International Sale of Goods is hereby expressly excluded. In the event any provision of this Agreement shall be deemed unenforceable, void or invalid, such provision shall be modified so as to make it valid and enforceable, and as so modified the entire Agreement shall remain in full force and effect. No decision, action or inaction by LICENSOR shall be construed to be a waiver of any rights or remedies available to it.

None of the Intellectual Property or underlying information or technology may be downloaded or otherwise exported or reexported in violation of U.S. export laws and regulations. In addition, you are responsible for complying with any local laws in your jurisdiction which may impact your right to import, export or use the Intellectual Property, and you represent that you have complied with any regulations or registration procedures required by applicable law to make this license enforceable.

Chapter 1. Summary

This Engineering Report (ER) describes the development of compliance tests and their implementation in the OGC Test, Evaluation, And Measurement (TEAM) Engine to validate a client's ability to make secure requests according to the OGC Web Services Security Candidate Standard [https://portal.opengeospatial.org/files/?artifact_id=76516&version=1]. The goal of the candidate standard is to allow the implementation of Information Assurance (IA) controls and to advertise their existence in an interoperable way with minimal impact to existing implementations using a backward-compatible approach.

This ER covers the following topics from OGC Testbed-14 Compliance Interoperability & Testing Evaluation (CITE) thread:

- developing a client validator to test compliance of client software with the OGC Web Services Security Candidate Standard
- capturing the results of two use cases with different authentication methods
- making recommendations to the OGC Web Services Security Standards Working Group (SWG) based on the experiences made while developing the validator

NOTE	At the time of conducting the work described in this ER, OGC Web Services Security was still a candidate standard. As of January 28th, 2019 the specification has now been approved by the OGC Technical Committee and published as an OGC® Implementation Standard.

1.1. Requirements & Use Cases

The requirements addressed by this ER are to document the following two use cases:

- test whether a client can operate in compliance with the standard according to Use Case I of the OGC Web Services Security Candidate Standard [https://portal.opengeospatial.org/files/?artifact_id=76516&version=1] (section 5.2) which is a server that only does "server-side authentication using HTTPS but no client authentication or authorization". It can be tested for one OGC service, preferably a Web Map Service (WMS) or Web Feature Service (WFS).
- test whether a client can operate in compliance with the standard according to Use Case II of the OGC Web Services Security Candidate Standard [https://portal.opengeospatial.org/files/?artifact_id=76516&version=1] (section 5.3) which is a server that has an "authentication method in place for client authentication". It can be tested for one OGC service (preferably WMS or WFS) with authentication according to requirement class OpenID Connect (section 7.11.2 of OGC Web Services Security Candidate Standard [https://portal.opengeospatial.org/files/?artifact_id=76516&version=1]).

1.2. Prior-After Comparison

OGC Testbed-14 developed a client validation engine (client validator) to:

- enhance the CITE tests by developing a client validation engine.

- test compliance of software products (client software) with the OGC Web Services Security Candidate Standard to ensure interoperable processing of advertised security controls.

The OGC Web Services Security Candidate Standard [https://portal.opengeospatial.org/files/?artifact_id=76516&version=1] defines an annotation mechanism for Capabilities documents or responses to the *GetCapabilities* request, ensuring interoperability between a secured OGC Web Service (OWS) instance and a client application. It further overrides existing HTTP protocol limitations and exception handling from existing OWS standards for the purpose of achieving interoperability with mainstream Information Technology (IT) security standards and their implementations. The standard has been informed by security-related work in OGC Testbed-11 [1], OGC Testbed-12 [2], and OGC Testbed-13 [3].

1.3. Recommendations for Future Work

This ER extends the OGC Web Services Security Candidate Standard [https://portal.opengeospatial.org/files/?artifact_id=76516&version=1] to allow for proper specification of different authentication methods.

The ER stimulates future work regarding OGC standardization requirements for client applications to work on modern security aspects. More details can be found in the section titled "Summary of Findings".

1.4. Document contributor contact points

All questions regarding this document should be directed to the editor or the contributors:

Contacts

Name	Organization
Sara Saeedi (Editor)	University of Calgary
James Badger (Contributor)	University of Calgary
Michael Leedahl (Contributor)	DigitalGlobe
Luis Bermudez	OGC

1.5. Foreword

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. The Open Geospatial Consortium shall not be held responsible for identifying any or all such patent rights.

Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the standard set forth in this document, and to provide supporting documentation.

Chapter 2. References

The following normative documents are referenced in this document.

NOTE: Only normative standards are referenced here, e.g., OGC, ISO or other SDO standards. All other references are listed in the bibliography.

- OGC: OGC 06-121r9, OGC® Web Services Common Standard, 2010 [https://portal.opengeospatial.org/files/?artifact_id=38867&version=2]
- OGC: OGC 17-007, OGC Web Service Security Candidate Standard, 2018 [https://portal.opengeospatial.org/files/77693]
- ISO: ISO/IEC 10181-2, Information technology — Open Systems Interconnection — Security frameworks for open systems: Authentication framework, 1996 [http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=18198]
- ISO: ISO/IEC 10181-3, Information technology — Open Systems Interconnection — Security frameworks for open systems: Access control framework, 1996 [http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=18199]
- OASIS: UDDI Spec Technical Committee Draft, 2004 [http://www.uddi.org/pubs/uddi_v3.htm]
- OASIS: Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0, 2005 [http://docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf]

Chapter 3. Terms and definitions

For this report, the definitions specified in Clause 4 of the OWS Common Implementation Standard OGC 06-121r9 [https://portal.opengeospatial.org/files/?artifact_id=38867&version=2] shall apply.

- Abstract Test Suite (ATS)

```
A set of testable assertions about the functionality of a standard, which an
implementation must support to achieve compliance with the standard. ATS is based
on the conformance clauses defined in the standard. (source: OGC 08-134r10)
```

- Authentication

```
ISO 10181-2 defines all basic concepts of authentication in Open Systems: it
identifies different classes of authentication mechanisms, the services for their
implementation and the requirements for supporting protocols. It further defines
requirements for the management of identity information.
```

- Authorization

```
ISO 10181-3 defines all basic concepts for access control and authorization in Open
Systems and the relation to other frameworks such as the Authentication and Audit
Frameworks.
```

- Capabilities Document

```
Web service's metadata encoded in XML which can be received by the
GetCapabilities() operation among all OWS services.
```

- Conformance

```
a standard's "abstract conformance" to Standards Packages for that standard (see
ISO 19105:2000 Geographic information - Conformance and Testing at
http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=260
10).
```

- Compliance

```
a state of a specific software product, which implements an OGC Standard and has
passed the Compliance Testing Evaluation. (source: OGC 08-134r10)
```

- HTTP Basic Authentication

```
The most straightforward technique for enforcing access controls to an HTTP web
browser which is sent in the HTTP header is an HTTP Basic Authentication. The HTTP
basic authentication is a method to provide a user name and password when making a
request.
```

- Interoperability

```
capability to communicate, execute programs or transfer data among various
functional units in a manner that requires the user to have little or no knowledge
of the unique characteristics of those units (source: ISO 19119)
```

- OAuth 2.0

```
It is a framework, specified by the IETF in RFCs 6749 and 6750 (published in 2012)
designed to support the development of authentication and authorization protocols.
It provides a variety of standardized message flows based on JSON and HTTP.
```

- OpenID Connect

```
OpenID Connect defines optional mechanisms for robust signing and encryption based
on OAuth 2.0 family of specifications. It allows clients of all types (APIs,
mobile, and JavaScript clients) to request and receive information about
authenticated sessions and end-users. OpenID Connect lets developers authenticate
their users across websites and apps without having to own and manage password
files.
```

- Proxy Server

```
A Proxy server is a server that acts as an intermediary for requests from clients
seeking resources from other servers.
```

- SAML 2

```
Security Assertion Markup Language 2.0 (SAML 2.0) is a version of the SAML standard
for exchanging authentication and authorization data between security domains. SAML
2.0 is an XML-based protocol that uses security tokens containing assertions to
pass information about a principal (usually an end user) between a SAML authority,
named an Identity Provider, and a SAML consumer called a Service Provider. SAML 2.0
enables web-based, cross-domain single sign-on (SSO), which helps reduce the
administrative overhead of distributing multiple authentication tokens to the user.
```

- SSL Client Certificates

```
Secure Sockets Layer (SSL), is a cryptographic protocol designed to secure all
communications between servers and web browsers. A client certificate is used to
authenticate clients (not servers) during an SSL handshake process to verify that
the client is who they claim to be.
```

3.1. Abbreviated Terms

- API Application Programming Interface
- AWS Amazon Web Services
- CDATA Character DATA
- CFP Call For Participation
- CITE Compliance Interoperability & Testing Evaluation
- CORS Cross Origin Resource Sharing
- CTL Compliance Test Language
- ECP Enhanced Client Proxy
- ETS Executable Test Suite
- HTTP Hypertext Transfer Protocol
- HTTPS Hypertext Transfer Protocol Secure
- IA Information Assurance
- IDE Integrated development environment
- IDP Identity Provider
- IETF Internet Engineering Task Force
- ISO International Organization for Standardization
- IT Information Technology
- JAX-RS Java API extension for RESTful Web services
- JAR Java ARchive
- OGC Open Geospatial Consortium
- OSGi Open Services Gateway initiative
- OWS OGC Web Service
- SAML Security Assertion Markup Language
- SSL Secure Sockets Layer
- SSO Single Sign-On
- SWG Standards Working Group
- TEAM Test, Evaluation, and Measurement
- TestNG Test Next Generation

- TLS Transport Layer Security
- URL Uniform Resource Locator
- WCS Web Coverage Service
- WFS Web Feature Service
- WMS Web Map Service
- WMTS Web Map Tile Service
- WPS Web Processing Service
- WWW World Wide Web
- XML eXtensible Markup Language

Chapter 4. Overview

This public engineering report was produced during the OGC Testbed-14 project, executed from April to December 2018. The document focuses on specific security topics that were worked on during OGC Testbed-14 for client-side security testing.

The document contains the following information:

- Section 5 introduces the OGC Web Services Security Standard and presents the design of secure client testing, implementation plan, use cases and issues.
- Section 6 discusses test scenarios, their issues and considerations.
- Section 7 explains the results of testing different scenarios.
- Section 8 provides a summary of the main findings.

Chapter 5. Background

The OGC Web Services Security Candidate Standard [https://portal.opengeospatial.org/files/?artifact_id=76516&version=1] specifies how conforming OGC Web Services shall advertise their Information Assurance (IA) controls, describes the governance process for IA Control registers, details examples of register contents, and describes how this information should be used. Next, this candidate standard defines conformance classes and requirements classes to be used for reaching compliance and their validation via conformance tests. Finally, the candidate standard defines client behavior to ensure interoperable processing of advertised security controls.

The candidate standard defines an annotation mechanism for Capabilities documents or responses to the *GetCapabilities* request, ensuring interoperability between a secured OGC Web Service instance and a client application. It further overrides existing HyperText Transfer Protocol (HTTP) protocol limitations and exception handling from existing OGC Web Services standards to achieve interoperability with mainstream IT security standards and their implementations.

5.1. Implementation Plan and Use Cases

To validate a secure client's conformance to the standard, an executable test suite that can act as a server must be developed. Guidelines on the requirements for such a test suite harness are covered in Annex B of OGC Web Services Security Candidate Standard [https://portal.opengeospatial.org/files/?artifact_id=76516&version=1], and can be summarized as follows:

- Test harness must be configurable for each Conformance Class defined by the standard
- Test harness must be configurable for producing annotated capabilities according to different combinations requirements classes
- Must include support for HTTPS requirements class and Partial Capabilities

These test harness requirements must be combined with the goal for OGC Testbed-14, testing a secure client's ability to adhere to Use Case 1 and Use Case 2 of the OGC Web Services Security Candidate Standard [https://portal.opengeospatial.org/files/?artifact_id=76516&version=1].

- **Use Case 1** is to verify that the secure client can anonymously connect to the secure server using HTTPS and issue valid requests.
- **Use Case 2** is similar to Case 1, but the secure client must provide authentication to the server.

The specific method of authentication is not defined, but it could be HTTP Basic or SSL Client Certificates.

Discussion in the CITE thread with other participants led to the selection of two client authentication methods for this implementation. The first is OpenID Connect, and the second is version 2 of the Security Assertion Markup Language (SAML) standard. Details on the implementation of these in the test suite will be explained below, and the testing procedure for these methods described in a later chapter.

Development of the test suite implementation was influenced by existing work from other CITE Executable Test Suites (ETS), particularly the WMS Client test suite. Nearly all other test suites are

based on testing a service's implementation, and those work by integrating a client in the test suite to examine interactions with a service or deployment. For the secure client test, this will have to be inverted, by integrating a server into the test suite to test a client's interactions.

The WMS Client test suite does this by integrating a proxy service to communicate with actual WMS implementations. By having the WMS client connect to the proxy, the test suite can act as a broker between the client and an actual WMS implementation, while collecting information on the client's behavior in its HTTP requests. These HTTP requests can then be analyzed by tests to verify their conformance to a standard (in this case, the WMS standard). It is useful to understand how the WMS Client test suite works, as it shows what one can and cannot do with the Secure Client test suite.

The WMS Client test suite uses TEAM Engine's *MonitorServlet* class to allocate a new Uniform Resource Locator (URL) endpoint for tests, and assign a remote URL as the actual recipient of requests. This URL endpoint is then given to the test user, which they input to their client. The WMS client test suite uses a JavaScript-based program to set up the test session on what typically is the test session configuration page in TEAM Engine. As the client interacts with the proxy endpoint, details of the interactions are stored in the test session directory. The WMS Client test suite uses TEAM Engine's ability to create custom endpoints (using TEAM Engine SPI *JAX-RS (Java API extension for RESTful Web services)* ApplicationComponents, a separate Application Programming Interface (API) from TEAM Engine Web's *MonitorServlet*) to allow the JavaScript program on the test configuration page to query the status of the test, by retrieving the metadata that the client had sent to the proxy endpoint. This metadata is then analyzed by the JavaScript program and displayed to the user as a checklist of conformance and requirements according to the standard. When the test user is finished executing the checklist, they will submit the test form (what usually would be the "start test" action in TEAM Engine), and the JavaScript program will send the client test results to TEAM Engine for conversion into a test report.

There are some downsides to this testing method. Using TEAM Engine APIs in the test suite requires the test suite to run under TEAM Engine, and cannot run as a standalone Java ARchive (JAR) file. Running as a standalone JAR is useful for some test users who may want to automate testing using an interface-less automation framework, or for those who do not want to set up a Java Servlet Container such as Apache Tomcat. TEAM Engine does have a command-line console interface that does not require a servlet container to use, however, the WMS Client test suite must be run from a web browser and from a servlet container application for the JavaScript test program to run. The test suite also does not rely on TestNG (Test Next Generation) and server-side conformance classes, which means the JavaScript program is responsible for the conversion to TEAM Engine reports.

On the upside, the web-based checklist automatically updates as the test user runs their client. This provides excellent visual feedback to the test user that they are correctly using the client. As a test suite developer, updating and debugging a JavaScript-based program would have a faster development loop than Java, which requires recompilation between development tests; a JavaScript program can be updated by editing the Compliance Test Language (CTL) script directly or debugged in a web browser. Using a proxy to actual WMS implementations means the test suite does not need to include its own WMS implementations and can rely on those external implementations for providing data.

For the Secure Client test suite, the testbed sought to also provide dynamic URL endpoints for secure clients to use when testing their conformance. The testbed also tried to provide a clear user

interface that shows the test user what to expect and what to do with their client when executing the test suite. The test suite should also included server-side conformance classes implemented with TestNG, as these are easier to document, and the documentation is automatically generated for test users and test suite developers.

Using these lessons from the WMS Client test suite, as well as the requirements and use cases for the Secure Client test suite, the following workflow was proposed:

1. The Secure Client Test suite is compiled, providing a TEAM Engine module. A system administrator then installs this into TEAM Engine along with the CTL scripts and dependent Java libraries.
2. A test user who wants to validate their secure client then logs into the TEAM Engine instance in their web browser. They select the Secure Client Test from the installed test suites and create the test session.
3. In the test session configuration page, the test user selects the secure service type they want to emulate. The service type should be selected according to their secure client's implementation; for example, users choose "WMS 1.1.1" if they are testing a secure client that implements WMS 1.1.1.
4. The test user can also configure the security annotations that will be added to the Capabilities document of the emulated service. These correspond to the requirements classes specified in the standard. The test user also selects an authentication method according to the use case. For Use Case 1, they would select "No Client Authentication," for Use Case 2, they would select an option such as "OpenID Connect" or "SAML 2". Note that the selection of some authentication methods may require the test user to provide additional parameters such as Identity Provider details.
5. The configuration page lists the test session endpoint URL for the secure client before the session has been started. The endpoint is not active until the user creates the test session, at which point a separate web server is started to capture requests on that endpoint URL. This separate web server responds to the mandatory request endpoints as defined by the standard that is being emulated; however, the Capabilities are limited to the bare minimum required to test the secure client.
6. The test user then uses this URL in their secure client, which then opens an HTTP or HTTPS connection to the Secure Client test server. The server implements HTTP port unification, which handles clients that either open an insecure HTTP connection or secure HTTPS connection on the same port.
7. The Secure Client Test server then records the connection metadata from the secure client, passing that data on to the TestNG test methods. The test methods then inspect the validity of secure client interactions. TEAM Engine then converts these TestNG results to a report that is readable in the web browser.
8. The test user then reviews the test results to see any issues with the conformance of their secure client.

5.2. Implementation Details

The source code for the secure client test suite is divided into multiple packages. The main package

is for running TestNG, the test executor framework used for OGC executable test suites. This portion of the code is responsible for receiving the test run properties from either a file when executed by a tester via a JAR, an Integrated Development Environment (IDE) or from a dynamically generated file from TEAM Engine. This TestNG Controller class will then validate the properties for correctness and set up an embedded test HTTP web server.

The embedded web server module will start a server with the parameters from the test run properties. These include server details such as the address, port, and hostname, as well as what kind of server to emulate and what secure service annotations to apply to its capability document. This server runs as a unified HTTP/HTTPS service supporting both protocols on the same port, as the test suite must check if a secure client is using HTTPS or not to make requests.

Currently, the test suite supports emulating WMS 1.1.1, WMS 1.3.0, and Web Processing Service (WPS) 2.0 (for OWS Common); additional types could be emulated by updating the test suite with sub-classes that can respond to "Get Capabilities" requests. Once the embedded server is started, a URL for that HTTP server is sent to the test user, either through the command line (for JAR/IDE) or on the web interface (for TEAM Engine).

The test user then connects their secure client to the test endpoint URL. For a simple no-authentication test workflow the test server waits to receive one request from the secure client, sends a response, and shuts down the test server. For more complex workflows such as SAML 2.0, the test server will wait for a pre-determined number of requests from the secure client before shutting down. Each request from the client is logged, converted to eXtensible Markup Language (XML), and saved to a temporary directory.

Once the test server has shut down, the TestNG Controller will take over and send the test run properties as well as the temporary file containing the secure client requests to the TestNG conformance class test packages. If there is an issue with setting up the test server such as missing test run properties, inability to bind the server, timeout waiting for secure client requests, and other potential issues will cause the TestNG controller to abort the embedded test server and pass the partial test run properties and results to the test methods.

The TestNG conformance class test methods are grouped in packages according to conformance classes from the OGC Web Services Security Candidate Standard [https://portal.opengeospatial.org/files/?artifact_id=76516&version=1]. These methods can validate the secure client request encoding and correctness with passed, failed, and skipped tests being logged in a report. The report is presented to the user at the end of the test run — when ran via JAR/IDE, an XML report is produced, and when running via TEAM Engine, the report is transformed into an HTML report that can be read in the web browser.

5.3. Implementation Issues

Implementation of the Secure Client test suite encountered multiple issues that complicated development. Understanding these issues, what caused them, and the potential solutions are useful for developers of not only the Secure Client test suite, but can be re-used in the development of other future client test suites.

The main development issues are as follows.

1. Embedding a web server causes Java package conflicts with Servlet Containers
2. Supporting non-TEAM Engine environments increases development complexity
3. Testing HTTPS requires additional work to deploy SSL certificates
4. TEAM Engine and Executable Test Suite documentation is almost entirely focused on testing services
5. Embedding a web server is not ideal for how TEAM Engine creates test sessions

5.3.1. Embedding a web server causes Java package conflicts with Servlet Containers

The embedded web server is Jetty, which uses the Java Servlet API. This has no issues when ran as a standalone JAR file, but when ran in TEAM Engine the Servlet API will conflict with the Servlet API used for Apache Tomcat which is hosting TEAM Engine. As Apache Tomcat is loaded before the TEAM Engine and the secure client test suite, only the first Java Servlet API is loaded. This version of the Servlet API is older than the one used in Jetty, causing Jetty to fail when calling methods or classes that do not exist in the older version.

One option is to downgrade Jetty such that it has the same Servlet API version as Apache Tomcat. This is not ideal as earlier versions of Jetty may have security vulnerabilities or known bugs that could interfere with the test workflow and may not support the same Transport Layer Security (TLS) ciphers as modern clients. It would also mean that the test suite would have to update in lock-step with the Apache Tomcat version used for TEAM Engine, making installation more complicated for administrators.

Upgrading Apache Tomcat is another option. This places more work on the system administrator and may raise compatibility issues with TEAM Engine. There is also an issue with the newest version of Apache Tomcat (version 9) supporting Java Servlet API 4.0, which is not yet supported by any version of Jetty.

Dynamic class loaders in Java could potentially be used to load the Servlet API for Jetty separately from the Servlet API for Apache Tomcat.

OSGi dynamic modules could potentially work, loading Jetty and its Servlet API separately to avoid the version conflict. This solution was not used as it would require significant development time to transition using Open Services Gateway initiative(OSGi).

Running the Test Server in a separate process from TEAM Engine under Apache Tomcat could potentially work. This would require setting up some inter-process communication system to initialize and configure the HTTP test server and return the request data to TestNG for analysis in the test methods.

The solution chosen was to use the Maven "Shade" plugin. This allows you to rename imported packages, and references in the package classes will be updated to use the new names. The Shade plugin was used to renamed `javax.servlet` to `org.opengis.cite.servlet`, and `org.eclipse.jetty` to `org.opengis.cite.jetty`. As the package name changed, Apache Tomcat would load the library in addition to its Servlet API. In the test server code, it would now use the "new" package names to import classes. This works seamlessly when deployed under TEAM Engine in Apache Tomcat.

5.3.2. Supporting non-TEAM Engine environments increases development complexity

The test suite supports execution via JAR/IDE or TEAM Engine. These have some subtle differences that increase the complexity of debugging. For example, running under TEAM Engine uses a different XML Document class which is immutable. Running as a JAR, the same XML document is mutable. Trying to modify the immutable version causes a crash in TEAM Engine.

A potential solution is to replace the JAR workflow with a command line workflow that runs from within the TEAM Engine console. This would eliminate some issues with different classes being used, and development in an IDE should adequately reference the TEAM Engine version of those classes.

5.3.3. Testing HTTPS requires additional work to deploy SSL certificates

With most executable test suites, a client is used to evaluate a service's implementation and conformance. In this test suite, the roles are reversed, and a server is used to assess a client's compliance. For this test suite, it must also evaluate HTTPS support. To do this, a certificate must be generated and supplied to the embedded test server. The certificate is then presented to clients connecting with HTTPS.

It is easy to generate a self-signed certificate and use that for testing; however, a self-signed certificate will fail a client's certificate validation as the certificate is not trusted by any certificate authority known to the client. The client may have an option to disable certificate validation, but not all clients may allow this. Instead, the generated certificate should be signed by an authority trusted by the client. This authority may be managed by an organization that keeps the certificates internal only, or a public certificate authority can be used.

The public and free "Let's Encrypt" certificate authority was successfully used in this testbed to generate certificates that work with Apache Tomcat and Jetty, on a cloud server set up by the University of Calgary development team to run pre-release and beta versions of the test suite.

Managing HTTPS certificates for servers will add additional complexity for system administrators, but the certificates are necessary to test the secure client's HTTPS support.

5.3.4. TEAM Engine and Executable Test Suite documentation is almost entirely focused on testing services

A challenge in the development of this test suite is that the majority of existing test suites are focused on service testing and not client testing. This means documentation on building a client testing suite are limited, and existing documentation is built on the service testing workflow. Some of the missing or sparse procedures include details such as how best to structure the test suite code and packages, how to support running from the command line and running from TEAM Engine, what metadata to collect from the client for testing, and similar.

This will change as more client test suites are developed and documented, and lead to a more re-usable approach to building a client testing suite.

5.3.5. Embedding a web server is not ideal for how TEAM Engine creates test sessions

The initial assumption during the development of the test suite would be that a single embedded test server instance would be created and re-used for multiple test sessions, each test session getting its sub-path URL that queries the test server. This would prevent different test sessions from mixing their requests on the same URL. When running under a JAR, it is not necessary to support multiple test sessions, as a single user is running the test suite locally. However, when running under TEAM Engine, a new TestNG Controller is created for each test session, which in turn creates a separate embedded test server.

Trying to launch a second test server fails as the port is still being held by the first test server (from the first test session). This is a problem for TEAM Engine users, as only a single secure client test suite session can be running at a single time.

A potential solution is not to use ports on the host, and instead, bind to Unix domain sockets. Then TEAM Engine could set up a proxy URL that redirects to that domain socket. By having a unique name for each domain socket, there would be no conflicts. This may only work under specific operating systems, and TEAM Engine may not support proxying to domain sockets.

A better solution would be to build support into TEAM Engine for setting up temporary test endpoint URLs, which are then proxied or routed to a class/method in the test suite. Then client test suites would not need to include their own embedded test server and instead could rely on TEAM Engine to provide that functionality.

Chapter 6. Test Scenario, Issues, and Concerns

The test scenario describes the TEAM Engine as a proxy server between the ArcGIS Desktop Client Application (client), an Identity Provider (IDP) and the Web Map Service (WMS) using a SAML Web Browser SSO(Single Sign-On) Profile (Figure 1).

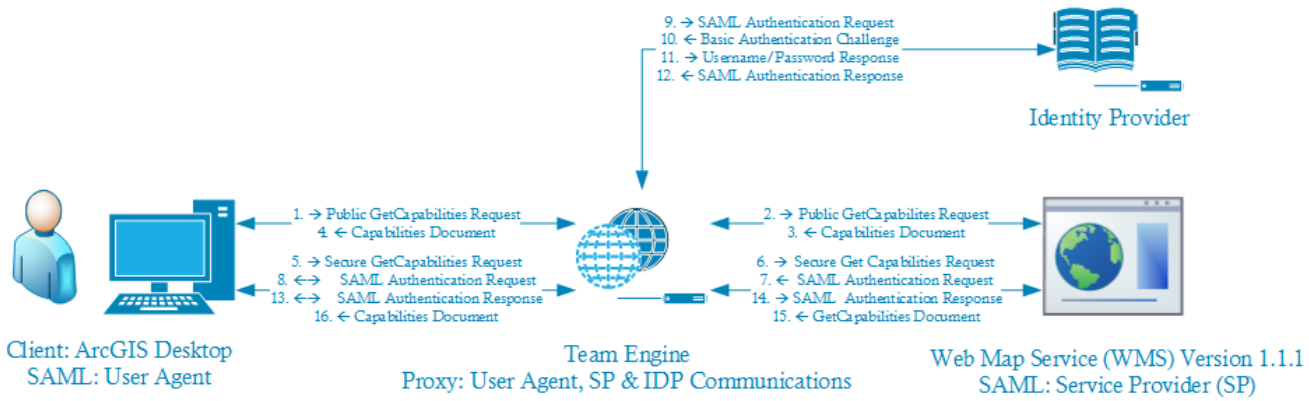

Figure 1. High-Level Mind Map of Testbed 14

However, SAML has many different profiles, and an implementer can mix and match the profile components to make up an implementation. The first big question is:

How to certify that a client understands SAML 2.0?

Is this even the right question for OGC? The purpose of compliance testing of the client is to see if the client can read the security extensions from the Capabilities document and understand it. The hope is that the client can make a judgment about whether they support the authentication method. This gives the client the ability to inform the user of the client when a particular service such as a WMS is supported by the client. The only real way to see if the client understands the markings is to observe its behavior with an actual implementation. To this end, the test engine may be implemented to act as a specific implementation that can proxy the authentication requests and responses to/from some particular reference implementation behind the test engine to monitor communications between the client, IDP, and WMS. The issue is that there are so many different ways to implement SAML. Thus, one has to consider what it is precisely that is being observed?

How much of the SAML standard needs to be implemented and validated in the test engine?

The OGC Web Services Security standard lists the SAML 2.0 Web Browser SSO Profile as an authentication option. The SAML 2.0 Standard documentation describes authentication types that may be requested from and implemented by an IDP; however, the implementation details of the authentication are left to the implementer of the IDP. To make matters more confusing for client implementers, a particular implementation may combine different profile elements to create a unique implementation. Thus, if developers say they understand SAML that means they understand a lot of profiles; not just the two general flavors of authentication communication such as Web Browser SSO Profile and Enhanced Client Proxy (ECP) Profile. For example, each profile may use sub-profiles such as HTTP Redirect, HTTP Post and Archival Document Profiles to name a few.

- If one implements the test engine to act as an IDP with a particular implementation or proxy to

a known reference implementation of an IDP; then, one can determine that the client and geospatial service provider such as WMS are compliant with that implementation. Likewise, does one have the test engine implemented to proxy to a known reference implementation of a geospatial service provider such as WMS with the SAML Service Provider Profile? This is necessary because the service provider needs to understand the implementation of the IDP; hence, the need for a particular implementation. If one has a specific implementation of the IDP and geospatial service provider, is that good enough to say that the client understands SAML markings in the Capabilities document? It is certainly not good enough to say that the client is SAML 2.0 compliant; only that it is compliant with a particular implementation.

- If one proxies to a vendor solution of an IDP and/or geospatial service provider, then the test engine needs to understand a wide variety of implementations. It would need to implement the SAML Metadata Profiles so that the test engine can be configured to a particular IDP and geospatial service acting as a SAML service provider. Again, this highlights that the client can handle a specific implementation. The other issues related to the IDP or geospatial service provider may not have a proper implementation of SAML, and this would complicate the certification process. The issue is that which one is being certified, just the client or the implementation of the IDP and geospatial service provider? One can see the need to certify both the client and the service provider but think that the IDP is probably out of scope. And, it may be better to separate the client tests from the geospatial service provider tests. This brings us back to the question of is it good enough to say that the client understands the security extension in the Capabilities document? Again, it isn't sufficient to say that the client is SAML 2.0 compliant; only that is compliant with a particular implementation.

6.1. Amazon Web Services (AWS) Considerations

When developing the Identity Provider in AWS, two issues surfaced. The first issue was that the API Gateway decodes any URL encoding before calling a Lambda. This provides a challenge with the XML Signature passed on the URL of the redirect to the Identity Provider. The Identity Provider needs to use URL encoding to encode the signed content before validating the signature. However, if the Service Provider encodes the content slightly differently, the Signature will not validate. You could calculate the signature on the base64 encoding before encoding for the URL in the Service Provider. This would allow the Identity Provider to verify the signature without re-encoding the content; however, this isn't how SAML is supposed to work.

The second issue encountered is that API Gateway remaps some common HTTP headers as Amazon does not want to implement anything around these headers. However, this prevents anyone else from implementing anything around these headers. The header that causes grief with our scenario is the `WWW-Authenticate` and Authorization headers for Basic Authentication support. To overcome this issue, the Identity provider needs to pass a custom `WWW-Authenticate` header (`X-WWW-Authenticate`) back to the browser. To prevent non-standard issues, the design uses a CloudFront Lambda@Edge [https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/lambda-at-the-edge.html] to remap the custom header back to the standard `WWW-Authenticate` header. The Identity provider looks for the response back from the browser in a header that amazon remapped it to (`x-amzn-remapped-Authentication`). From the consumer side, this appears to be a standards-based exchange. From the Identity providers viewpoint, this is a custom non-standard exchange.

One more consideration to keep in mind for testing and debugging, the Lambda@Edge writes log

messages to the data center closest to the edge server. Due to demand and load balancing, logs can go to different data centers. Adding an `X-Lambda-Region` header with the region where the Edge Lambda is run can help to troubleshoot efforts.

6.2. SAML Web Browser SSO Profile

The User Agent illustrated in Figure 2 is the ArcGIS Desktop Plugin. The plugin calls the *GetCapabilities* method on the Web Map Service (WMS). The capabilities document includes a security description that informs the plugin that it will be using a SAML Browser SSO process flow. This means the plugin must support HTTP redirection and HTML document processing as part of the workflow involves user interaction with a web page. The plugin also needs to support cookies as the assertion for subsequent calls will be stored in a cookie. The plugin then makes a request to the WMS for the full capabilities document by calling the *GetCapabilities* method listed in the public version. The WMS, acting as the Service Provider in Figure 2, then sends back a redirect to the plugin which then accesses the Discovery Service.

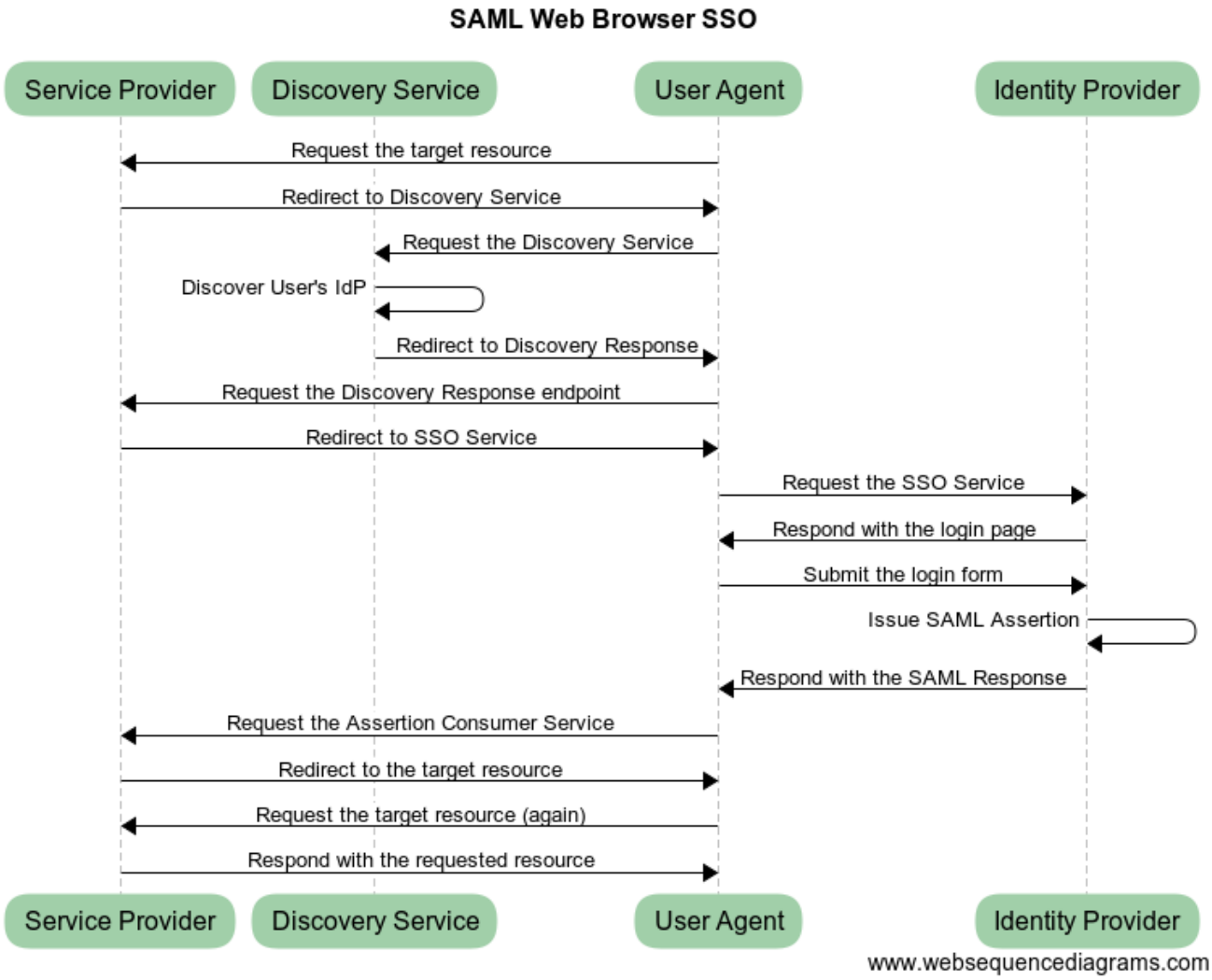

Figure 2. High-Level Mind Map of Testbed 14

The Discovery Service provides a choice to the plugin as an HTML document for how it wants to authenticate. The plugin allows the user of the plugin to make a choice and submits it to the Discovery Service. Then, the Discovery Service sends a redirect back to the Plugin for the Identity Provider associated with the choice. The plugin then goes to the Identity provider which provides a

means to elicit values for the authentication. The plugin allows the user to fill in the information and sends it to the Identity Provider. The Identity Provider responds back with a redirect to the Service Provider which then validates the assertion provided by the Identity Provider and sends a redirect back to the plugin for the target resource (*GetCapabilities* method). The plugin calls the *GetCapabilities* method and gets the full capabilities document back.

At this point the plugin follows the same flow for making a *GetMap* request; but this time, the plugin will be able to pass an assertion that the Service Provider (*e.g.* WMS) understands and will service the resource without doing the rest of the process flow.

NOTE

For the first version, the discovery profile may be skipped and go straight to a default Identity Provider. The first Identity Provider will follow the authentication challenge response process of IETF (Internet Engineering Task Force) RFC 2617 [https://www.ietf.org/rfc/rfc2617.txt] to elicit a user name and password from the plugin. The plugin would prompt the user for the information and pass it back to the Identity Provider. The rest of the flow would be as noted above.

6.3. Evaluation of Clients Lacking HTTPS Support

The secure client executable test suite must verify if a secure client uses HTTPS on the connection, as HTTPS is mandatory. To do this, the test suite must be able to mark a request from the secure client as having HTTPS or not. The test suite server does this by using port unification, which allows the test suite to act as an HTTP or HTTPS server depending on the client's opening message. As the client sends the first message, the server can "peek" at the headers to see if it is a TLS header for HTTPS. If the headers are missing, then the client can be assumed to be connecting with plain HTTP. The test suite will then respond with its capabilities document even if HTTPS is not available; however, the lack of HTTPS will be marked in the metadata that is sent to the test methods. By still accepting HTTP queries, the test suite can evaluate clients lacking HTTPS on other conformance test methods while marking the HTTPS test method as a failure.

6.4. Evaluation of Client Support for Partial Capabilities

For a secure client, they must also support partial capabilities as defined in the standard. This, however, cannot be evaluated from the test suite (*i.e.* server side), as the parsing and interpretation of the partial capabilities occur entirely in the client. The test user must evaluate that their secure client was able to parse the partial capabilities document for the full capabilities document URL, get the full capabilities and pass any application security controls, and parse the "Content" section from the full capabilities document.

The report generated by the test suite contains a note for the test user that they must evaluate the partial capabilities support manually, as well as provide a checklist and reference to the standard. As the test suite code does not assess it, then categorizes it as a "Skipped" test instead of a "Passed" or "Failed" test. This is provided as a convenience to the test user, reminding them of additional steps necessary to evaluate their client's conformance.

Chapter 7. Test Results

This public engineering report was produced by OGC Testbed-14, executed from April to December 2018. Even though the title is broad, this document focuses on specific security topics that were worked on during the testbed.

7.1. Requirement Classes in the Test Suite

There are three conformance classes (WMS 1.1.1, WMS 1.3.0, and OWS Common) that each has their own requirements class. In addition to these classes, there are requirement classes for different security annotations adding more constraints to the capabilities presented to secure clients. Some of these classes are related to protocol behavior and supporting HTTP features, and the rest of the classes are for specification of authentication and authorization.

For OGC Testbed 14, the secure client test suite implemented the main HTTP requirement classes and one of the authorization requirement classes. "HTTPS" was provided using TLS on the test suite embedded server. This embedded test server only supports "HTTP/1.1", one of the classes; neither "HTTP/1.0" or "HTTP/2" are enabled. Advertising for "W3C CORS (Cross-origin resource sharing)" is also enabled, as clients that use CORS (particularly web browsers) require that it is enabled to query the service. "HTTP Exceptions" will use standard HTTP status codes and messages with the exceptions returned by the service, linking the exceptions to the category of error (HTTP 4xx — client error, HTTP 5xx — server error).

For authentication, only "SAML2" is supported. This provides the SAML2 Web Browser SSO Profile for secure clients, as that profile had been built for the secure client as part of this testbed. Activating this requirement class in a test session will affect the annotations and the behavior of the test suite. The secure client will be expected to make multiple requests instead of only one.

The other authentication requirement classes were not added to the test suite as they require significantly more development and testing than the other classes. This includes coding for building additional responses, communication with external identity services, and supporting the three main conformance classes. Testing will also require secure clients that support these authentication methods, and no secure clients with support were available for this testbed. Some of the authentication requirement classes share some functionality with the SAML2 workflow, meaning code and procedures can be re-used to build support in the future.

7.2. Test Suite Demo

The details for the test suite (`ets-security-client10`) are accessible from its GitHub Repository [https://github.com/opengeospatial/ets-security-client10] and a demonstration video is available on OGC portal [https://portal.opengeospatial.org/files/?artifact_id=81760] and its YouTube channel.

This executable test suite (ETS) verifies the conformance of the secure client behavior as the Implementation Under Test (IUT) with respect to the following specification(s):

Conformance testing for the OGC Web Services Security 1.0 standard is a kind of "black box" testing that examines the externally visible characteristics or behaviors of the IUT while disregarding any implementation details.

Several conformance classes are defined in the principal specification; the ones listed below are covered by this test suite:

- Conformance Class "Common Security" (abstract)
- Conformance Class "OWS Common"
  - Only affects Capabilities document presented to secure client
- Conformance Class "WMS 1.1.1"
  - Only affects Capabilities document presented to secure client
- Conformance Class "WMS 1.3.0"
  - Only affects Capabilities document presented to secure client

The following snapshots (Figure 3) show two snapshots of the TEAM Engin while running the test.

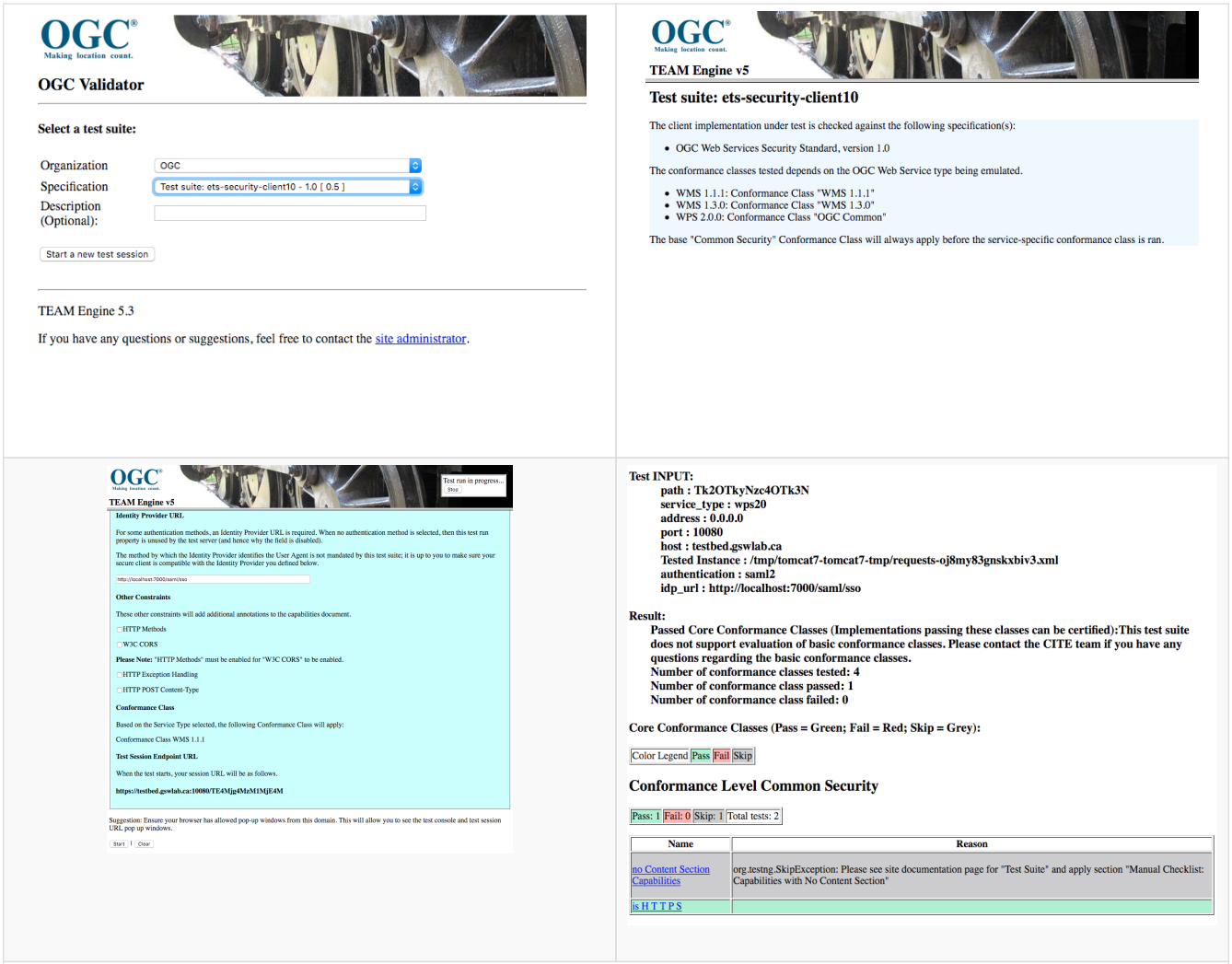

Figure 3. Screen Shots of TEAM Engine running the Client Test and its results

Visit the project documentation website [http://opengeospatial.github.io/ets-security-client10/] for more information, including the API documentation.

NOTE The project documentation site will be available when the test suite has been published to the OGC TEAM Engine site.

7.2.1. Test Suite Structure

The test suite definition file (testng.xml) is located in the root package, `org.opengis.cite.securityclient10`. A conformance class corresponds to a <test> element, each of which includes a set of test classes that contain the actual test methods. The general structure of the test suite is shown in Table 1.

Table 1. Test suite structure

Conformance class	Test classes
`Conformance Level Common Security`	`org.opengis.cite.securityclient10.levelCommonSecurity.`
`Conformance Level OWS Common`	`org.opengis.cite.securityclient10.levelOwsCommon.`
`Conformance Level WMS 1.1.1`	`org.opengis.cite.securityclient10.levelWms111.`
`Conformance Level WMS 1.3.0`	`org.opengis.cite.securityclient10.levelWms13.`

7.2.2. Test Run Arguments

The test run arguments are summarized in Table 2. The Obligation descriptor can have the following values: M (mandatory), O (optional), or C (conditional).

Table 2. Test run arguments

Name	Value domain	Obligation	Description
service_type	String	M	A string representing the service type to emulate for secure clients. Will affect the query parameters required to get capabilities. Valid values are: - `wms111` for WMS 1.1.1 Conformance Class - `wms13` for WMS 1.3.0 Conformance Class - `wps20` for WPS 2.0.0 on the OWS Common Conformance Class
address	String	M	Host interface on which to bind test server. When using IPv6, do not use square brackets, *e.g.* ::1
port	Integer	M	Port on which to bind test server.

Name	Value domain	Obligation	Description
host	String	M	Host name that will be advertised to clients. Maybe an IP address or domain name. Clients must be able to resolve this IP or name to the machine running the test server. If you are using a certificate from a Certificate Authority, this parameter must match the common name on that certificate. When using IPv6, use square brackets around the address, *e.g.* [::1]
path	String	O	URL Path at which the Test Server will listen. For example, test-session would result in the Test Server creating a servlet for https://host:port/test-session. If left blank, a random string will be generated. When used with TEAM Engine, the CTL script will automatically fill in a path such that the test session web page can inform the tester of the URL before starting the test session.

Name	Value domain	Obligation	Description
jks_path	String	M	A Java KeyStore containing the X.509 certificates for the host address must be located at *jks_path* for the embedded Jetty server to provide HTTPS. Self-signed certificates are permitted, although the test client will have to trust that certificate manually.
jks_password	String	M	The password required to unlock the Java KeyStore. Using a CDATA (character data) section is recommended to wrap passwords that have character data that may interfere with XML character entities.

Name	Value domain	Obligation	Description
authentication	String	O	Specify the authentication method that will be presented to Secure Clients via the capabilities document. Only one authentication method may be specified. Values are not case sensitive. Enabling this will include an `ows:Constraint` for requiring authentication from the secure client. For some authentication methods, additional test run properties may be mandatory. If this element is omitted or is empty, then authentication will not be enabled. For SAML2 authentication, specify saml2 and specify the *idp_url* test run property: `<entry name="authentication"> saml2</entry>`

Name	Value domain	Obligation	Description
idp_url	String	O	As part of the annotated capabilities document presented to the secure client, include Requirements Class "SAML2" (https://www.opengis.net/def/security/1.0/rc/authentication/saml2). Enabling this will include an `ows:Constraint` for requiring SAML2 authentication from the secure client. The authentication test run property must be set to saml2; otherwise, this test run property is ignored. The value must be a URL that resolves to a SAML2 SSO resource. This value will then be passed to the secure client in the `ows:Constraint`. Note that it is up to the test user to validate the URL and the service at that URL, the ETS will do no verification and pass the URL "as-is" to the secure client in the capabilities document. If this element is omitted or is empty, then SAML2 authentication will not be enabled. A guide has been included for running SAML2 tests.

Name	Value domain	Obligation	Description
http_methods	String	O	As part of the annotated capabilities document presented to the secure client, include Requirements Class "HTTP Methods" (https://www.opengis.net/def/security/1.0/rc/http-methods). Enabling this will include an `ows:Constraint` for listing all supported HTTP methods. Only a value of "true" will enable this property, and any other value will be evaluated as false. If the w3c_cors property is set to true, then this property will be overridden to true as well.

Name	Value domain	Obligation	Description
w3c_cors	String	O	As part of the annotated capabilities document presented to the secure client, include Requirements Class "W3C CORS" (https://www.opengis.net/def/security/1.0/rc/cors). Enabling this will include an `ows:Constraint` for the W3C recommendation "Cross Origin Resource Sharing". Only a value of true will enable this property, and any other value will be evaluated as false. If set to true, then the *http_methods* test run property will also be configured to true regardless of your configuration; "HTTP Methods" is required for this Requirements Class.

Name	Value domain	Obligation	Description
http_exception_handling	String	O	As part of the annotated capabilities document presented to the secure client, include Requirements Class "HTTP Exception Handling" (https://www.opengis.net/def/security/1.0/rc/http-exception-handling). Enabling this will include an `ows:Constraint` for enabling HTTP error code mapping to OWS Common exception codes. Only a value of true will enable this property, and any other value will be evaluated as false.

Name	Value domain	Obligation	Description
http_post_content_type	String	O	As part of the annotated capabilities document presented to the secure client, include Requirements Class "HTTP POST Content-Type" (https://www.opengis.net/def/security/1.0/rc/content-type). Enabling this will include an `ows:Constraint` for listing the mime-types permitted to be submitted by HTTP POST. At the minimum, the `application/x-www-form-urlencoded` MIME type must (and will) be supported and advertised by the service. Only a value of true will enable this property, and any other value will be evaluated as false.

Additional test run properties may be added in the future to support other authentication methods such as SAML 2.0 and OpenID Connect.

7.2.3. How to Run the Tests

The test suite is built using Apache Maven v3 [https://maven.apache.org/]. Maven is necessary for self-hosting the test suite in your test environment.

The test suite can be running in the following environments:

- Under an IDE such as Eclipse
- As a self-contained JAR command-line application
- As a module under TEAM Engine

7.2.4. Test Run Properties

For IDE and JAR testing, an XML file is used to pass the test run properties into the test suite. TEAM Engine users will instead use the web form interface to specify these properties, and TEAM Engine

will automatically convert the form into an XML file for the test suite.

More details about the mandatory and optional test run properties can be located in the GitHub Repository documentation [https://github.com/opengeospatial/ets-security-client10/blob/master/src/site/markdown/index.md].

1. **Integrated development environment (IDE)**

Use a Java IDE such as Eclipse, NetBeans, or IntelliJ. Clone the repository and build the project.

Set the main class to run: `org.opengis.cite.security-client10.TestNGController`.

Arguments: The first argument must refer to an XML properties file containing the required test run arguments. If not specified, the default location at `${user.home}/test-run-props.xml` will be used.

The TestNG results file (`testng-results.xml`) will be written to a subdirectory in `${user.home}/testng/` having a UUID value as its name.

2. **Command shell (console)**

One of the build artifacts is an "all-in-one" JAR file that includes the test suite and all of its dependencies; this makes it very easy to execute the test suite in a command shell:

```
java -jar ets-security-client10-0.1-SNAPSHOT-aio.jar [-o|--outputDir $TMPDIR] [test-run-props.xml]
```

This will require you first to compile and build the test suite as a Java application.

3. **OGC test harness**

Use TEAM Engine [https://github.com/opengeospatial/teamengine], the official OGC test harness. The latest test suite releases are usually available at the beta testing facility [http://cite.opengeospatial.org/te2/]. One can also build and deploy [https://github.com/opengeospatial/teamengine] the test harness for use as a local installation.

A detailed guide on setting up TEAM Engine is included in the GitHub Repository [https://github.com/opengeospatial/ets-security-client10].

7.2.5. Debugging the ETS

If you need to debug a secure client connection and inspect the HTTP or HTTPS details, add these to your Java VM arguments:

```
-DDEBUG=true -Dorg.eclipse.jetty.LEVEL=DEBUG -Djavax.net.debug=ssl,handshake,data
```

This will print very detailed information about cipher suites and any TLS extensions available from the client.

7.2.6. About the included sample Java KeyStore

This repository contains a sample Java KeyStore with a self-signed certificate for testing purposes. The keystore can be created with the following command.

```
$ keytool -keystore src/main/resources/security.jks -storepass "ets-security-client"
-genkey -alias dummy-key -keyalg RSA -sigalg SHA256withRSA -dname "cn=ETS Test
Operator, ou=None, o=None, c=us"
```

This creates `src/main/resources/security.jks` with a single key (`dummy-key`) and protects the file with a password (`ets-security-client`). As this is a self-signed certificate, secure clients must allow insecure server certificates or install the certificate to their keystore.

When running the test suite from an IDE, this KeyStore will be used as it is specified in `src/main/config/test-run-props.xml`.

To create a PEM version of the file for secure clients to use:

```
$ keytool -exportcert -alias dummy-key -keystore src/main/resources/security.jks
-storepass "ets-security-client" -rfc -file src/main/resources/security.pem
```

Chapter 8. Summary of Findings

In the course of building and releasing the deliverables and engineering report for OGC Testbed-14, a number of highlights and recommendations have been identified.

The following deliverables have been implemented:

- Embedded HTTP/HTTPS testing server.
- Emulated WMS 1.1.1, WMS 1.3.0, and WPS 2.0 for the conformance classes.
- SAML2 Authentication.
- Optional Requirement Classes "HTTP Methods", "W3C CORS", "HTTP Exception Handling", HTTP POST Content Type".
- Sample command-line interface secure clients for testing the ETS workflow.

The following items could not be implemented and are recommended for future Testbed efforts:

- OpenID Connect Authentication (this one was in the CFP, but nobody developed a secure client using it — in the Testbed 14 CITE thread meetings, we decided to implement SAML2 instead of OpenID Connect).
- OpenAPI Authentication.
- Authorization Requirement Class.
- WS-Policy Requirement Class.
- General Authentication Requirement Class.

The difficulties and lessons learned from the implementations and deliverables are listed as follows:

- ETS/TEAM Engine are good at building testing clients, but lack features for building testing servers.
- Some suggestions for improving ETS/TEAM Engine test server support:
  - Instead of embedding an HTTP server, there should be an API for creating/destroying servlets and specifying custom code in the ETS for handling requests to those servlets;
  - When running outside of TEAM Engine Web, the ETS should use TEAM Engine console instead of running the ETS TestNG Controller as a JAR;
  - TEAM Engine console should support the servlet create/destroy/bind API and run its own server if necessary (Jetty or similar);
  - Instructions for setting up HTTPS certificates will be necessary for TEAM Engine, as an ETS with a test server may need HTTPS;
  - Supporting HTTP/HTTPS Port Unification would be very helpful for an ETS to determine if clients are using HTTPS — this may be difficult to implement as the Java application container (*e.g.* Tomcat) would have to be configured specifically for this.

Appendix A: Revision History

NOTE *Example History (Delete this note).*
replace below entries as needed

Table 3. Revision History

Date	Editor	Release	Primary clauses modified	Descriptions
Oct 1, 2018	Sara Saeedi	.1	all	comments integrate
Nov 21, 2018	Sara Saeedi	.8	all	comments integrate
Nov 22, 2018	Gobe Hobona	.9	all	comments integrate
Feb 27, 2019	Sara Saeedi	1.0	various	preparation for publication

Appendix B: Bibliography

1. Matheus, A.: OGC Testbed 11 Engineering Report: Implementing Common Security Across the OGC Suite of Service Standards. OGC 15-022,Open Geospatial Consortium, https://portal.opengeospatial.org/files/?artifact_id=63312 (2015).
2. Matheus, A.: OGC Testbed 12 Engineering Report: OWS Common Security Extension. OGC 16-048r1,Open Geospatial Consortium, http://docs.opengeospatial.org/per/16-048r1.html (2017).
3. Matheus, A.: OGC Testbed 13 Engineering Report: Security. OGC 17-021,Open Geospatial Consortium, http://docs.opengeospatial.org/per/17-021.html (2018).