European Commission – Eurostat/B3

Specific Contract

QTM-4/2014

SRI.NET Vulnerability and Security Assessment Report   
(version 1.2)

Document Service Data

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Type of Document** |  | | | | |
| **Reference:** |  | | | | |
| **Issue:** |  | **Revision:** | 2 | **Status:** | Final |
| **Created by:** | Nikos Varotsis (NV),  Dragos Balan (DB) | | | **Submission Date:** | 31/03/2015 |
| **Distribution:** | EU-Eurostat, Intrasoft International S.A, Agilis S.A. | | | | |
| **Contract Full Title:** |  | | | | |
| **Service contract number:** | Specific Contract | | | | |
| For Internal Use Only | | | | | |
| **Reviewed by:** | Dragos Balan (DB) | | | | |
| **Approved by:** | Mario Fendler (MF) | | | | |

Document Change Record

|  |  |  |
| --- | --- | --- |
| Issue/Revision | Date | Change |
| 1.0 | 02/12/2013 | Initial version of the document. |
| 1.1 | 18/08/2014 | Improvements after running the security tests for SRI.Net synchronized to SdmxSource 1.1.4 |
| 1.2 | 23/03/2015 | Document updated to reflect the security assessment of SRI.NET having SdmxSource 1.1.4b |

Table of contents

1 Introduction 6

1.1 Purpose 6

1.2 Scope 6

1.3 Document Structure 6

1.4 References 6

1.5 Terms and abbreviations 7

2 General information on vulnerabilities in applications 8

2.1 Fuzz testing or fuzzing 8

2.2 SQL injection 8

2.3 XML bomb 8

2.4 XPath injection 9

2.5 Boundary scan / Invalid Types 9

2.6 Malformed XML 9

2.7 Cross Site Scripting 9

3 Security assessment of the .Net NSI Web Service 4.3.0 11

3.1 SDMX 2.0 Web Services security assessment 13

3.1.1 The first attack on ‘GetCompactData’ 13

3.1.2 The second attack on ‘GetCompactData’ request 15

3.1.3 The third sample of ‘GetCompactData’ request 17

3.1.4 The first sample of ‘QueryStructure’ request 20

3.1.5 The second sample of ‘QueryStructure’ request 23

3.1.6 Summary results for the SDMX 2.0 compliant Web Services 25

3.2 SRI Web Service SDMX 2.1 compliant 27

3.2.1 Security testing for ‘GetCodelist’ request 27

3.2.2 Checking the security for ‘GetStructureSpecificData’ request 30

3.2.3 Summary results for the SDMX 2.1 Compliant web services 34

3.3 SDMX 2.1 REST Web Services Security Assessment 35

3.3.1 Security testing for ‘GetCodelist’ REST request 35

3.3.1.1 Security testing for ‘GetData’ REST method 37

3.3.2 Summary results for the SRI Rest web services security test 40

3.4 SRI AuthModule 41

4 Security assessment of .Net NSI Web Client 3.4.0 44

4.1 OWASP 44

4.2 Samurai Web Testing Framework 45

4.3 Analysis of Web Client application risks using Samurai 46

4.3.1 A1 Injection 46

4.3.1.1 Testing SQL Injection on the POST “dataflow change” ajax request 47

4.3.1.2 Testing SQL Injection on the “update results” ajax POST request 48

4.3.2 A2 Broken Authentication and Session Management 49

4.3.2.1 Checking the Broken Authentication and Session management on “dataflow change” ajax request by fuzzing the requests 51

4.3.2.2 Checking the Broken Authentication and Session management on “results update” ajax request 52

4.3.3 A3 - Cross site scripting 53

4.3.3.1 XSS attack on “dataflow change” 53

4.3.3.2 XSS attack on “results update” 54

4.3.4 A4 - Insecure Direct Object References 54

4.3.5 A5 - Security miss-configuration 55

4.3.6 A6 – Sensitive Data Exposure 55

4.3.7 A7 - Missing Function Level Access Control 55

4.3.8 A8 - Cross Site Request Forgery 56

4.3.9 A9 - Using Components with Known Vulnerabilities 58

4.3.10 A10 - Invalidated Redirects and Forwards 59

4.3.11 Summary results for Web Client security testing 59

**List of figures Page**

[Figure 1 Security test results for GetCompactData WS method 14](#_Toc415496404)

[Figure 2 Security results for the second sample of GetCompactData 17](#_Toc415496405)

[Figure 3 Security test results for the third sample of GetCompactData method 20](#_Toc415496406)

[Figure 4 Security testing report for the first sample of QueryStructure method 22](#_Toc415496407)

[Figure 5 Security test report for the second sample of QueryStructure 25](#_Toc415496408)

[Figure 6 Results of the GetCodelist security testing 29](#_Toc415496409)

[Figure 7 Security report for GetStructureSpecificData 34](#_Toc415496410)

[Figure 8 Security report for GetCodelist attack 37](#_Toc415496411)

[Figure 9 Security Report for the attack on GetData method 39](#_Toc415496412)

[Figure 10 Attack the http basic authentication 42](#_Toc415496413)

[Figure 11 Basic Authentication mechanism with base64 encoded text 43](#_Toc415496414)

[Figure 12 ZAP Security tool performing an SQL Injection attack 47](#_Toc415496415)

[Figure 13 SQL Injection attack on “dataflow change” ajax request 48](#_Toc415496416)

[Figure 14 SQL Injection attack on “results update” ajax request 49](#_Toc415496417)

[Figure 15 List of plugins used during Broken Auth. and Session Management test 50](#_Toc415496418)

[Figure 16 Fuzzer performing a fuzz attack for Broken Auth. and Session Mgmt 51](#_Toc415496419)

[Figure 17 Fuzz attack sample on “dataflow change” ajax request 51](#_Toc415496420)

[Figure 18 Fuzz attack on “results update” ajax request 52](#_Toc415496421)

[Figure 19 List of plugins used for the XSS attack 53](#_Toc415496422)

[Figure 20 Sample XSS request on “dataflow change” ajax request 54](#_Toc415496423)

[Figure 21 List of scans planned for CSRF security testing 56](#_Toc415496424)

[Figure 22 ZAP Active Scan performing a CSRF attack on NSI Web Client 57](#_Toc415496425)

[Figure 23 A sample malicious active scan request for “criteria edit” ajax request 57](#_Toc415496426)

[Figure 24 Results of the Active Scan security test 58](#_Toc415496427)

**List of tables Page**

[Table 1 - Terms and Abbreviations 7](#_Toc415496428)

[Table 2 List of security scans performed on GetCompactData 14](#_Toc415496429)

[Table 3 List of scans performed on GetCompactData second sample 17](#_Toc415496430)

[Table 4 List of scans performed on GetCompactData third sample 19](#_Toc415496431)

[Table 5 List of scans performed on the first sample of QueryStructure 22](#_Toc415496432)

[Table 6 List of scans performed on the second sample of QueryStructure 24](#_Toc415496433)

[Table 7 Summary Results for SDMX 2.0 Web Services 26](#_Toc415496434)

[Table 8 List of performed scans on GetCodelist 29](#_Toc415496435)

[Table 9 List of performed scans on GetStructureSpecificData 33](#_Toc415496436)

[Table 10 Results of the security testing for SDMX 2.1 web services 34](#_Toc415496437)

[Table 11 List of performed scans on GetCodelist REST service 36](#_Toc415496438)

[Table 12 List of performed scans on GetData REST service 38](#_Toc415496439)

[Table 13 Results of security scans for REST services 40](#_Toc415496440)

[Table 14 List of external libraries used in .Net NSI Web Client 58](#_Toc415496441)

[Table 15 Results of the security tests for .Net NSI Web Client 60](#_Toc415496442)

# Introduction

## Purpose

This document stands for the deliverable “Vulnerability and Security Assessment Report” as a result of integration of SdmxSource 1.1.4b into SRI.NET (QTM-4 / 2014) (see [R1]).

The purpose of this document is to present a vulnerability and security assessment analysis on the SRI modules.

## Scope

The scope of this deliverable is to illustrate the potential vulnerabilities of the SDMX SRI by utilizing the knowledge of code injection along with software tools targeted at security checking. Solutions will be presented where needed.

## Document Structure

The structure of this document is as follows:

**Section** 1 is an introduction of the document.

**Section 2** presents general information on security vulnerabilities.

**Section 3** contains the vulnerability/security assessment on the SRI modules.

## References

This document references:

|  |  |
| --- | --- |
| Reference | Document/Resource Name |
| [R1] | [Technical annex of QTM-4/2014](https://circabc.europa.eu/d/a/workspace/SpacesStore/b57eade4-cc8f-4a89-81a8-7b1116d601bc/QTM-4_SDMX_2014-10_Subtask_4_IT-000042%20annex_iia_v5%20Final.pdf) |
| [R2] | <http://en.wikipedia.org/wiki/SQL_injection> |
| [R3] | <http://www.unixwiz.net/techtips/sql-injection.html> |
| [R4] | <http://www.acunetix.com/websitesecurity/sql-injection.htm> |
| [R5] | <http://en.wikipedia.org/wiki/Cross-site_scripting> |
| [R6] | <http://www.acunetix.com/websitesecurity/xss.htm> |
| [R7] | <http://searchsoftwarequality.techtarget.com/answer/XML-security-Preventing-XML-bombs> |
| [R8] | <https://www.owasp.org/index.php/XPATH_Injection> |

## Terms and abbreviations

| **Acronym** | **Definition** |
| --- | --- |
| SDMX | Statistical Data and Metadata eXchange |
| SDMX-ML | XML format used in SDMX messages |
| SQL | Structured Query Language |
| SQLi | SQL injection |
| SRI | SDMX Reference Infrastructure |
| WS | Web Service |
| XML | Extensible Markup Language |
| XSS | Cross-site scripting |

Table 1 - Terms and Abbreviations

# General information on vulnerabilities in applications

In computer security, vulnerability is a weakness, which allows an attacker to reduce a system's information assurance. Various code injection techniques exist that exploit security vulnerabilities. For the purposes of the current report the SQL injection, which is a code injection technique, the cross-site scripting (XSS), which is a specific type of security vulnerability, alongside with other security scans will be utilized.

## Fuzz testing or fuzzing

Fuzz testing is a software testing technique, often automated or semi-automated, that involves providing invalid, unexpected, or random data to the inputs of a computer program. The program is then monitored for exceptions such as crashes, or failing built-in code assertions or for finding potential memory leaks. Fuzzing is commonly used to test for security problems in software or computer systems.

## SQL injection

SQL Injection occurs at the database level of an application. It is said to be the most common hacking technique, which attempts to pass SQL statements through a web application for execution on the database. With such a type of attack, data can be stolen or deleted from a database. Please consult the following [R2], [R3] and [R4] for more elaborated information on the SQL injection.

## XML bomb

An XML bomb is a dangerous XML message that can be crafted by a malicious user. It could be sent by a service consumer (a client) as a request message to a Web service provider (a server) in order to cause a denial-of-service DoS attack. An XML bomb exploits Document Type Definition (DTD) processing functionality in XML parsers -- the software that parses an XML message -- by causing them to process an exponentially growing amount of data. An XML bomb is actually a small XML message that is created in a way that makes the XML content grow only as it gets processed by the XML parser.

The XML Bomb attacks will be tested only against XML based protocols which in our case is SOAP.

## XPath injection

XPath injection issimilar to SQL Injection, XPath Injection attacks occur when a web site uses user-supplied information to construct an XPath query for XML data. By sending intentionally malformed information into the web site, an attacker can find out how the XML data is structured, or access data that he may not normally have access to. He may even be able to elevate his privileges on the web site if the XML data is being used for authentication (such as an XML based user file). Here’s how XPath injection works: Querying XML is done with XPath, a type of simple descriptive statement that allows the XML query to locate a piece of information. Like SQL, you can specify certain attributes to find, and patterns to match. When using XML for a web site it is common to accept some form of input on the query string to identify the content to locate and display on the page. This input must be sanitized to verify that it doesn't mess up the XPath query and return the wrong data**.**

## Boundary scan / Invalid Types

Boundary scan / Invalid types tests handling of invalid input data having values outside of the defined boundaries or wrong types. When implementing web services it's easy to forget handling of values that you don't expect, especially if input is restricted already on client side. The Invalid Types Security Scan is designed to help you to make sure that your server handles these kinds of situations gracefully.

## Malformed XML

Malformed XML tries to exploit bad handling of Invalid XML on your service. This security check will be applied only to the SOAP services for obvious reasons.

## Cross Site Scripting

Cross Site Scripting (also known as XSS or CSS) is generally believed to be one of the most common application layer hacking techniques. In general, cross-site scripting refers to a hacking technique that leverages vulnerabilities in the code of a web application to allow an attacker to send malicious content from an end-user and collect some type of data from the victim.

Cross Site Scripting allows an attacker to embed malicious JavaScript, VBScript, ActiveX, HTML, or Flash into a vulnerable dynamic page to fool the user, executing the script on his machine in order to gather data. The use of XSS might compromise private information, manipulate or steal cookies, create requests that can be mistaken for those of a valid user, or execute malicious code on the end-user systems. The data is usually formatted as a hyperlink containing malicious content and which is distributed over any possible means on the internet.

Any web request which passes parameters to a database can be vulnerable to this hacking technique. Usually these are present in Login forms, Forgot Password forms, etc.

# Security assessment of the .Net NSI Web Service 4.3.0

The Web Service Provider (also known as SRI WS module) is a re-usable building block, wrapped into a SOAP or REST web service responsible for coordinating the incoming calls to the dissemination environment.

When deploying the SRI WS the user can configure the connection settings to the Mapping Store database. However this depends on the security aspects of the server and not the service itself. The same situation applies for the authentication/authorization settings.

SRI WS SOAP receives as input a SOAP message including an SDMX-ML Query message for data requests. Also, SRI WS receives as input a SOAP message including an SDMX-ML Registry Interface message for structural metadata requests. In the SDMX-ML Queries that are part of the SOAP Requests, someone can insert malicious data.

SRI REST Web Services receive a HTTP GET request containing all parameters as part of the URL query. The following paragraphs will analyze what kind of malicious information the user can insert in both SOAP and REST web services.

With the help of SoapUI tool, the security assessment was conducted on several sample requests selected randomly: two SDMX 2.0 operations (QueryStructure, GetCompactData), two SDMX 2.1 operations (GetCodelists, GetStructureSpecificData) and another two REST methods (GetCodelist and GetData). For all the queries the SoapUI run the security tests by applying seven security scans: fuzzing scan, boundary scan, invalid types, malformed XML, SQL injection, XML bomb and XPath Injection.

For an overview of these security scans, please consult <http://www.soapui.org/Security/security-scans-overview.html>.

Each security scan completes by indicating zero or more alerts. According to the SoapUI tool, alerts are raised when a valid response has been returned from the service and denote possible security vulnerability in the target service. For each security scan a security log is written which entails all the requests sent, the responses and the results.

The SoapUI tool was used to automate the procedure of sending malformed requests to the service with several security scans applied. Thus said, it is vital for the conduction of the security assessment in the SRI Web service to analyze all the results for those requests that raised alerts as well as for those that did not raise an alert. Those requests that failed according to SoapUI must be analyzed as the failure denotes a possible security vulnerability of the service. Moreover, for those requests that did not raise alerts the responses must be analyzed because even if the responses are SOAP fault messages, malicious data might have injected in the SRI Web Service.

## SDMX 2.0 Web Services security assessment

### The first attack on ‘GetCompactData’

For the first sample of ‘GetCompactData’ request, a query for a specific dataflow was provided as a test query to SoapUI. A snapshot of the first query is:

<Query>

<query:DataWhere>

<query:Dataflow>SSTSCONS\_PROD\_A</query:Dataflow>

</query:DataWhere>

</Query>

**The testing strategy**

The main target of the attacks has been the <query:Dataflow> xml value but together with it some other xml elements or values have been tested: the message id, the message name, the sender id and the sender name. An example of a malicious request can be seen below:

<soapenv:Envelope>

<soapenv:Header/>

<soapenv:Body>

<nsis:GetCompactData>

<QueryMessage>

<Header>

<ID>**7CGNS2rDMb**</ID>

<Test>**gAslSmIZUK**</Test>

<Prepared>**VUnkij**</Prepared>

<Sender id=”**WgS4zV**”/>

<Receiver id=”**EPnEO6rt**”/>

</Header>

<Query>

<query:DataWhere>

<query:Dataflow>**FI7f0UTWfSs**</query:Dataflow>

</query:DataWhere>

</Query>

</QueryMessage>

</nsis:GetCompactData>

</soapenv:Body>

</soapenv:Envelope>

The number of requests per vulnerability has been established based on the number or parameters targeted and the complexity of the request.

|  |  |
| --- | --- |
| Vulnerability | Number of tests |
| Fuzzing Scan | 100 malicious requests |
| Invalid Types | 198 invalid xml types requests |
| SQL Injection | 98 potential sql injections |
| Malformed XML | 46 malformed xml requests |
| XML Bomb | 9 xml requests |
| XPath Injection | 70 invalid xpath requests |

Table 2 List of security scans performed on GetCompactData

**The expected result**

SoapUI was configured to report a failure in the following situations:

1. The response of the server exposes sensitive information to the attacker after a malicious request has been processed.
2. The Web Service fails to respond as a result of the attack meaning that the server is down.

The web service is deemed secure if SoapUI does not report a failure or a warning during the tests.

**The result of the attack**

SoapUI didn’t raise any warnings or failures during the attack.

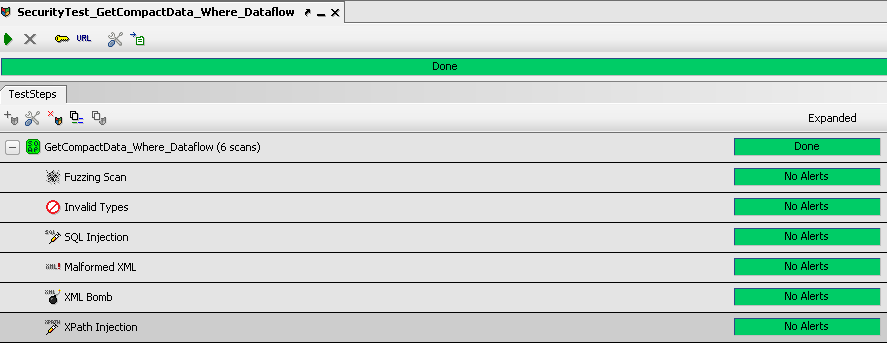


Figure 1 Security test results for GetCompactData WS method

A comprehensive listing of the parameters used for each attack as well as the result for each request can be found in the *[security\_reports]* folder under *sdmx2.0* subfolder as listing security\_scan\_SOAP\_GetCompactData\_Where\_Dataflow.txt

The GetCompactData SOAP method is deemed a secure Soap method for the tested combination of parameters.

### The second attack on ‘GetCompactData’ request

For the second sample of ‘GetCompactData’ request, a more complex query was provided to the tool containing references to a Dataflow and several dimensions as follows:

<Query>

<query:DataWhere>

<query:And>

<query:Dataflow>SSTSCONS\_PROD\_A</query:Dataflow>

<query:Or>

<query:Dimension id=*"STS\_ACTIVITY"*>NS0020</query:Dimension>

<query:Dimension id=*"STS\_ACTIVITY"*>NS0030</query:Dimension>

</query:Or>

</query:And>

</query:DataWhere>

</Query>

**The testing strategy**

The main target of the attacks has been the <query:Dataflow> value <query:Dimension> attributes and values. Of course together with these parameters all other xml elements have been tested: the message id, the message name, the sender id and the sender name. An example of a malicious request can be:

<soapenv:Envelope …>

<soapenv:Header/>

<soapenv:Body>

<GetCompactData>

<mes:QueryMessage>

<mes:Header>

<mes:ID>**MkcqJs1plwU**</mes:ID>

<mes:Test>**7l4RJm**</mes:Test>

<mes:Name lang="**tNyW4Wya**">**17q09yavuiHU1L**</mes:Name>

<mes:Prepared>**WvaKulCVlIAZBLu**</mes:Prepared>

<mes:Sender id="**vEkHOzrrGc4W**"/>

<mes:Receiver id="**Twnfyb9HIkq**"/>

</mes:Header>

<mes:Query>

<quer:DataWhere>

<quer:And>

<quer:Dataflow>**VHHcn09**</quer:Dataflow>

<quer:Or>

<quer:Dimension id="**A7L7XhrFGyd**">**PIxVoTAbjSV**</quer:Dimension>

<quer:Dimension id="**gcCYQJ**">

**17q09yavuiHU1L**

</quer:Dimension>

</quer:Or>

</quer:And>

</quer:DataWhere>

</mes:Query>

</mes:QueryMessage>

</ext:GetCompactData>

</soapenv:Body>

</soapenv:Envelope>

The number of requests per vulnerability has been established based on the number or parameters targeted and the complexity of this request.

|  |  |
| --- | --- |
| Vulnerability | Number of tests |
| Fuzzing Scan | 100 requests testing 10 xml attributes and values |
| Invalid Types | 246 invalid xml types requests |
| SQL Injection | 140 requests sql injections |
| Malformed XML | 40 malformed xml requests |
| XML Bomb | 30 xml bomb requests |
| XPath Injection | 100 xpath injection attempts |

Table 3 List of scans performed on GetCompactData second sample

**The expected result**

SoapUI was configured to report a failure in the following situations:

1. The response of the server exposes sensitive information to the attacker after a malicious request has been processed.
2. The Web Service fails to respond as a result of the attack meaning that the server is down.
3. Only for some requests: the response is not valid according to the SDMX 2.0 xsd schema. This is not possible in all requests because some values triggered Soap Faults Responses which of course is not valid according the the SDMX2.0 xsd.

The web service is deemed secure if SoapUI does not report a failure or a warning during the tests.

**The result of the attack**

SoapUI didn’t raise any warnings or failures during the attack.



Figure 2 Security results for the second sample of GetCompactData

The full report of the attack and the values for each parameter can be found in the [security\_reports] folder, under sdmx2.0 subfolder as security\_scan\_SOAP\_GetCompactData\_Where\_DafaflowOrDimension.txt file.

### The third sample of ‘GetCompactData’ request

The third and final attack on GetCompactData targeted a more complex query containing Dimension, Start and End Time, a data provider and dataflow:

<Query>

<query:DataWhere>

<query:And>

<query:Dimension id=*"STS\_ACTIVITY"*>NS0020</query:Dimension>

<query:Time>

<query:StartTime>2000</query:StartTime>

<query:EndTime>2005</query:EndTime>

</query:Time>

<query:DataProvider>GR</query:DataProvider>

<query:Dataflow>SSTSCONS\_PROD\_A</query:Dataflow>

</query:And>

</query:DataWhere>

</Query>

**The testing strategy**

This time 5 query parameters have been thoroughly tested: dimension, data provider, dataflow, start time and end time. Of course together with these parameters all other xml elements have been tested: the message id, the message name, the sender id and the sender name. An example of a malicious request can be:

<soapenv:Envelope…>

<soapenv:Header/>

<soapenv:Body>

<ext:GetCompactData>

<mes:QueryMessage>

<mes:Header>

<mes:ID>**wyvjRf3**</mes:ID>

<mes:Test>**fugpTHWWvjrpWt6**</mes:Test>

<mes:Name lang="**Mptz0Bj**">

**CENSUSHUB\_Q\_XS1**

</mes:Name>

<mes:Prepared>**zHOKFgXLPlQcYth**</mes:Prepared>

<mes:Sender id="**n9b6G4i0lzPY**"/>

<mes:Receiver id="**VhaIzfDPAtmg39G**"/>

</mes:Header>

<mes:Query>

<quer:DataWhere>

<quer:And>

<quer:Dimensionid="**kFJd28ikR**">

**W7e24dq1XbgDtRarhs**

</quer:Dimension>

<quer:Time>

<quer:StartTime>**1R6iuCb8FQUZRGH**</quer:StartTime>

<quer:EndTime>**G2vD2GuG**</quer:EndTime>

</quer:Time>

<quer:DataProvider>**pnnQkqFl**</quer:DataProvider>

<quer:Dataflow>**t7e25dq1XQbgDt**</quer:Dataflow>

</quer:And>

</quer:DataWhere>

</mes:Query>

</mes:QueryMessage>

</ext:GetCompactData>

</soapenv:Body>

</soapenv:Envelope>

The number of requests per vulnerability has been established based on the number or parameters targeted and the complexity of this request.

|  |  |
| --- | --- |
| Vulnerability | Number of tests |
| Fuzzing Scan | 100 requests testing 13 xml attributes and values |
| Invalid Types | 296 invalid xml types requests |
| SQL Injection | 168 sql injections requests |
| Malformed XML | 70 malformed xml requests |
| XML Bomb | 36 xml bomb requests |
| XPath Injection | 120 xpath injection attempts |

Table 4 List of scans performed on GetCompactData third sample

**The expected result**

SoapUI was configured to report a failure in the following situations:

1. The response of the server exposes sensitive information to the attacker after a malicious request has been processed.
2. The Web Service fails to respond as a result of the attack meaning that the server is down.
3. Only for some requests: the response is not valid according to the SDMX 2.0 xsd schema. This is not possible in all requests because some values triggered Soap Faults Responses which of course is not valid according the the SDMX2.0 xsd.

The web service is deemed secure if SoapUI does not report a failure or a warning during the tests.

**The result of the attack**

SoapUI didn’t raise any warnings or failures during the attack.

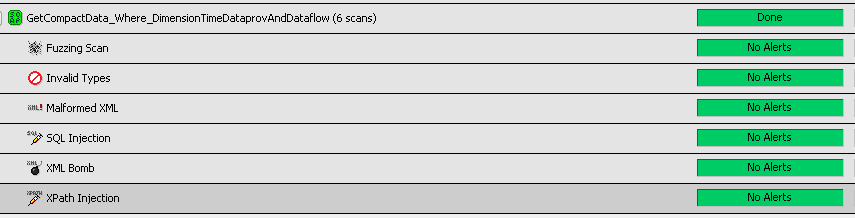


Figure 3 Security test results for the third sample of GetCompactData method

The full report of the attack and the values for each parameter can be found in the [security\_reports] folder, under sdmx2.0 subfolder as security\_scan\_SOAP\_GetCompactData\_Where\_DimensionTimeDataprovAndDataflow.txt

### The first sample of ‘QueryStructure’ request

In the first sample of ‘QueryStructure’ request, a query for a specific DSD was provided as a test query to SoapUI. A snapshot of the first query is as follows:

<QueryStructureRequest resolveReferences=*"false"*>

<registry:KeyFamilyRef>

<registry:AgencyID>ESTAT</registry:AgencyID>

<registry:KeyFamilyID>STS</registry:KeyFamilyID>

<registry:Version>2.0</registry:Version>

</registry:KeyFamilyRef>

</QueryStructureRequest>

**The testing strategy**

Testing the security for QueryStructure method focuses on KeyfamilyId, AgencyId and Version xml values. Together with these parameters all other xml elements have been tested: the message id, the message name and the sender id. An example of a malicious request can be:

<soapenv:Envelope …>

<soapenv:Header/>

<soapenv:Body>

<ext:QueryStructure>

<mes:RegistryInterface>

<mes:Header>

<mes:ID>**fsmF5d**</mes:ID>

<mes:Test>**a40iOej**</mes:Test>

<mes:Truncated>**HQARle4XGeaBZ**</mes:Truncated>

<mes:Name xml:lang="**p2KygvNA**">**Trans46302**</mes:Name>

<mes:Prepared>**kiK8tfkUYUTkq**</mes:Prepared>

<mes:Sender id="**jUyBHxHFMzW**"/>

</mes:Header>

<mes:QueryStructureRequest

resolveReferences="**YXSblAxehgm8**">

<reg:KeyFamilyRef>

<reg:AgencyID>**Qi7Sm5qVtSt0ok**</reg:AgencyID>

<reg:KeyFamilyID>**3g7vsBzmuO2jOj**</reg:KeyFamilyID>

<reg:Version>**nMFPnG**</reg:Version>

</reg:KeyFamilyRef>

</mes:QueryStructureRequest>

</mes:RegistryInterface>

</ext:QueryStructure>

</soapenv:Body>

</soapenv:Envelope>

The number of requests per vulnerability has been established based on the number or parameters targeted and the complexity of this request.

|  |  |
| --- | --- |
| Vulnerability | Number of tests |
| Fuzzing Scan | 100 requests testing 13 xml attributes and values |
| Invalid Types | 248 invalid xml types requests |
| SQL Injection | 168 sql injections requests |
| Malformed XML | 70 malformed xml requests |
| XML Bomb | 33 xml bomb requests |
| XPath Injection | 110 xpath injection attempts |

Table 5 List of scans performed on the first sample of QueryStructure

**The expected result**

SoapUI was configured to report a failure in the following situations:

1. The response of the server exposes sensitive information to the attacker after a malicious request has been processed.
2. The Web Service fails to respond as a result of the attack meaning that the server is down.
3. Only for some requests: the response is not valid according to the SDMX 2.0 xsd schema. This is not possible in all requests because some values triggered Soap Faults Responses which of course is not valid according the SDMX2.0 xsd.

The web service is deemed secure If SoapUI does not report a failure or a warning during the tests.

**The result of the attack**

During the attack SoapUI didn’t report any warnings or failures.



Figure 4 Security testing report for the first sample of QueryStructure method

The full report of the attack and the values for each parameter can be found in the [security\_reports] folder, under sdmx2.0 subfolder as security\_scan\_QueryStructure\_KeyFamily.txt file.

### The second sample of ‘QueryStructure’ request

For the second sample of ‘QueryStructure’ request, a query for a specific codelist was provided as a test query to SoapUI. A snapshot of the second query is as follows.

<QueryStructureRequest resolveReferences=*"false"*>

<registry:CodelistRef>

<registry:AgencyID>ESTAT</registry:AgencyID>

<registry:CodelistID>CL\_FREQ</registry:CodelistID>

<registry:Version>1.0</registry:Version>

</registry:CodelistRef>

</QueryStructureRequest>

**The testing strategy**

This time 3 parameters have been attacked: agency id, codelist id and version xml element. Along these 3 parameters, all other xml elements have been tested: the message id, the message name, the sender id and the sender name. An example of a malicious request can be:

<soapenv:Envelope …>

<soapenv:Header/>

<soapenv:Body>

<ext:QueryStructure>

<mes:RegistryInterface>

<mes:Header>

<mes:ID>**fsmF5d**</mes:ID>

<mes:Test>**a40iOej**</mes:Test>

<mes:Truncated>**HQARle4XGeaBZ**</mes:Truncated>

<mes:Name xml:lang="**p2KygvNA**">**Trans46302**</mes:Name>

<mes:Prepared>**kiK8tfkUYUTkq**</mes:Prepared>

<mes:Sender id="**jUyBHxHFMzW**"/>

</mes:Header>

<mes:QueryStructureRequest resolveReferences="**YXSblAxehgm8**">

<reg:CodelistRef>

<reg:AgencyID>**Qi7Sm5qVtSt0ok**</reg:AgencyID>

<reg:CodelistID>**3g7vsBzmuO2jOj**</reg:CodelistID>

<reg:Version>**nMFPnG**</reg:Version>

</reg:CodelistRef>

</mes:QueryStructureRequest>

</mes:RegistryInterface>

</ext:QueryStructure>

</soapenv:Body>

</soapenv:Envelope>

The number of requests per vulnerability has been established based on the number or parameters targeted and the complexity of this request.

|  |  |
| --- | --- |
| Vulnerability | Number of tests |
| Fuzzing Scan | 100 requests testing 13 xml attributes and values |
| Invalid Types | 98 invalid xml types requests |
| SQL Injection | 154 sql injections attempts |
| Malformed XML | 70 malformed xml requests |
| XML Bomb | 33 xml bomb requests |
| XPath Injection | 110 xpath injection attempts |

Table 6 List of scans performed on the second sample of QueryStructure

**The expected result**

SoapUI was configured to report a failure in the following situations:

1. The response of the server exposes sensitive information to the attacker after a malicious request has been processed.
2. The Web Service fails to respond as a result of the attack meaning that the server is no longer responding to requests.
3. Only for some requests: the response is not valid according to the SDMX 2.0 xsd schema. This is not possible in all requests because some values triggered Soap Faults Responses which of course is not valid according the SDMX2.0 xsd.

The web service is deemed secure if SoapUI does not report a failure or a warning during the tests.

**The result of the attack**

SoapUI didn’t raise any warnings or failures during the attack.



Figure 5 Security test report for the second sample of QueryStructure

The full report of the attack and the values for each parameter can be found in the reports folder as security\_scan\_SOAP\_QueryStructureCodeList.txt file.

### Summary results for the SDMX 2.0 compliant Web Services

In the following table a summary of the experiments for the .NET SRI WS Intermediate Solution is presented:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Web Service Method | Security checks | Warnings | | Vulnerabilities |
| GetCompactData  Sample1 | 521 | | 0 | None |
| GetCompactData  Sample2 | 656 | | 0 | None |
| GetCompactData  Sample3 | 656 | | 0 | None |
| QueryStructure  Sample1 | 729 | | 0 | None |
| QueryStructure  Sample2 | 665 | | 0 | None |

Table 7 Summary Results for SDMX 2.0 Web Services

## SRI Web Service SDMX 2.1 compliant

With the help of SoapUI tool the security assessment was conducted on two sample requests. One concerned the ‘GetCodelist’ operation of the SRI Web Service Ultimate Solution and the other concerned the ‘GetStructureSpecificData’ operation. In those query messages were entailed queries for data of specific dataflow, of a specific value of dimension as well as specific time dimension value. For all the queries the SoapUI run the security tests by applying six security scans: fuzzing scan, invalid types, malformed XML, SQL injection, XML bomb and XPath Injection.

Each security scan completes by indicating zero or more alerts. According to the SoapUI tool, alerts are raised when a valid response has been returned from the service and denote possible security vulnerability in the target service. For each security scan a security log is written which entails all the requests sent, the responses and the results.

### Security testing for ‘GetCodelist’ request

For the sample of ‘GetCodelist’ request, a query for a specific Codelist was provided as a test query to SoapUI. A snapshot of the query is as follows.

<query:CodelistWhere>

<query:ID operator=*"equal"*>CL\_FREQ</query:ID>

<query:Version>1.0</query:Version>

<query:AgencyID>ESTAT</query:AgencyID>

<query:CodeWhere>

<ns10:ID>EU-15</query:ID>

</query:CodeWhere>

</query:CodelistWhere>

**The testing strategy**

The testing strategy consists on providing malicious values for the following xml values: query id, agency id and version. Along with these 3 parameters, all other xml elements have been tested: the message id, the message name, the sender id and the sender name. An example of a malicious request can be:

<soapenv:Envelope …>

<soapenv:Header/>

<soapenv:Body>

<web:GetCodelist>

<mes:CodelistQuery>

<mes:Header>

<mes:ID>**jl7ZDJ0**</mes:ID>

<mes:Test>**WjzH3AK**</mes:Test>

<mes:Prepared>**DNRKx**</mes:Prepared>

<mes:Sender id="**YWfEq1RvQ7cn**"/>

<mes:Receiver id="**XgHAlFovG**"/>

</mes:Header>

<mes:Query>

<quer:ReturnDetails

detail="**cKwEO3z**"

returnMatchedArtefact="**ddz4eYSaeMwHC2e**">

<quer:References detail="**v8y6E**">

<quer:Parents/>

</quer:References>

</quer:ReturnDetails>

<quer:CodelistWhere>

<quer:ID operator="**Yi1kZoP40Y0o**">**qco1yBDu**</quer:ID>

<quer:Version>**Z9tvj6cLQQfgEu**</quer:Version>

<quer:AgencyID>**zEC1Zo1BBDu**</quer:AgencyID>

<quer:CodeWhere>

<quer:ID>**AU7orn2VnI**</quer:ID>

</quer:CodeWhere>

</quer:CodelistWhere>

</mes:Query>

</mes:CodelistQuery>

</web:GetCodelist>

</soapenv:Body>

</soapenv:Envelope>

The number of requests per vulnerability has been established based on the number or parameters targeted and the complexity of this request.

|  |  |
| --- | --- |
| Vulnerability | Number of tests |
| Fuzzing Scan | 100 fuzz requests sent |
| Invalid Types | 323 invalid xml types requests |
| SQL Injection | 182 sql injections attempts |
| Malformed XML | 60 malformed xml requests |
| XML Bomb | 39 xml bomb requests |
| XPath Injection | 130 xpath injection attempts |

Table 8 List of performed scans on GetCodelist

**The expected result**

SoapUI was configured to report a failure in the following situations:

1. The response of the server exposes sensitive information to the attacker after a malicious request has been processed.
2. The Web Service fails to respond as a result of the attack meaning that the server is no longer responding to requests.
3. Only for some requests: the response is not valid according to the SDMX 2.1 xsd schema. This is not possible in all requests because some values triggered Soap Faults Responses which of course is not valid according the the SDMX2.1 xsd.

The web service is deemed secure if SoapUI does not report a failure or a warning during the tests.

**The result of the attack**

SoapUI didn’t raise any warnings or failures during the attack.



Figure 6 Results of the GetCodelist security testing

The full report of the attack and the values for each parameter can be found in the [security\_reports] folder, under sdmx2.1 subfolder as security\_scan\_GetCodeList.txt

### Checking the security for ‘GetStructureSpecificData’ request

For the sample of ‘GetStructureSpecificData’ request, a query for a specific StructureSpecificData was provided as a test query to SoapUI. A snapshot of the query is as follows.

<message:Query>

<query:ReturnDetails>

<query:Structure dimensionAtObservation=*"TIME\_PERIOD"* structureID=*"STS"*>

<common:Structure>

<Ref agencyID=*"ESTAT"* id=*"STS"* version=*"2.2"*/>

</common:Structure>

</query:Structure>

</query:ReturnDetails>

<query:DataWhere>

<query:Dataflow>

<Ref agencyID=*"ESTAT"* id=*"SSTSCONS\_PROD\_A"*/>

</query:Dataflow>

<query:DimensionValue>

<query:ID>REF\_AREA</query:ID>

<query:Value>IT</query:Value>

</query:DimensionValue>

<query:TimeDimensionValue>

<query:TimeValue operator=*"greaterThanOrEqual"*>

2000

</query:TimeValue>

<query:TimeValue operator=*"lessThanOrEqual"*>

2005

</query:TimeValue>

</query:TimeDimensionValue>

<query:Or>

<query:DimensionValue>

<query:ID>STS\_ACTIVITY</query:ID>

<query:Value>NS0020</query:Value>

</query:DimensionValue>

<query:DimensionValue>

<query:ID>STS\_ACTIVITY</query:ID>

<query:Value>NS0030</query:Value>

</query:DimensionValue>

</query:Or>

</query:DataWhere>

</message:Query>

**The testing strategy**

Most of the available parameters have been tested against malicious values. A sample malicious request can be seen below:

<soap:Envelope …>

<soap:Header/>

<soap:Body>

<web:GetStructureSpecificData>

<message:StructureSpecificDataQuery

xsi:schemaLocation="**tZ2tBLThAX**">

<message:Header>

<message:ID>**pZPsbVzx9RKWDL**</message:ID>

<message:Test>**OS21hM7BrCNf**</message:Test>

<message:Prepared>**flRsnhOih**</message:Prepared>

<message:Sender id="**Gmo8GaJ**"/>

<message:Receiver id="**ItfKFinT1Tcwz2h**"/>

</message:Header>

<message:Query>

<query:ReturnDetails

defaultLimit="**V8bqbTiyqOw8w**"

detail="**htsAnHmI7**">

<query:FirstNObservations>

**X0QOO1EVEZwndFS**

</query:FirstNObservations>

<query:LastNObservations>

**msOf43x9fl**

</query:LastNObservations>

<query:Structure

dimensionAtObservation="**dzC6dY6ij55**"

structureID="**sl72CL8Vn**">

<common:Structure>

<Ref agencyID="**d5lpEyeiKuvJ**"

id="**WlLKGG1**"

version="**fxaPmqE3gfj**"/>

</common:Structure>

</query:Structure>

</query:ReturnDetails>

<query:DataWhere>

<query:Dataflow>

<Ref agencyID="**NhyurVR**" id="**Hn2URSykF6**"/>

</query:Dataflow>

<query:DimensionValue>

<query:ID>**87SYOCw**</query:ID>

<query:Value>**0pjLjaEEj**</query:Value>

</query:DimensionValue>

<query:TimeDimensionValue>

<query:TimeValue operator="**cJLDKu8a4MpB**">

**thReiidhjdh2012**

</query:TimeValue>

</query:TimeDimensionValue>

<query:Or>

<query:PrimaryMeasureValue>

<query:ID>**BiybkqI**</query:ID>

<query:NumericValue operator="**kwmOy6JKFt**">

**Ryrhlidghi**

</query:NumericValue>

</query:PrimaryMeasureValue>

</query:Or>

<query:Or>

<query:DimensionValue>

<query:ID>**as9kx**</query:ID>

<query:Value>**0s6XW5GAo2eZxr**</query:Value>

</query:DimensionValue>

<query:DimensionValue>

<query:ID>**b4kgBzwmg0TNb**</query:ID>

<query:Value>**5scAo**</query:Value>

</query:DimensionValue>

</query:Or>

</query:DataWhere>

</message:Query>

</message:StructureSpecificDataQuery>

</web:GetStructureSpecificData>

</soap:Body>

</soap:Envelope>

The number of requests per vulnerability has been established based on the number or parameters targeted and the complexity of this request.

|  |  |
| --- | --- |
| Vulnerability | Number of tests |
| Fuzzing Scan | 100 requests testing 13 xml attributes and values |
| Invalid Types | 24 invalid xml types requests |
| SQL Injection | 378 sql injections attempts |
| Malformed XML | 120 malformed xml requests |
| XML Bomb | 81 xml bomb requests |
| XPath Injection | 270 xpath injection attempts |

Table 9 List of performed scans on GetStructureSpecificData

**The expected result**

SoapUI was configured to report a failure in the following situations:

1. The response of the server exposes sensitive information to the attacker after a malicious request has been processed.
2. The Web Service fails to respond as a result of the attack meaning that the server is no longer responding to requests.
3. Only for some requests: the response is not valid according to the SDMX 2.1 xsd schema. This is not possible in all requests because some values triggered Soap Faults Responses which of course is not valid according the the SDMX2.1 xsd.

The web service is deemed secure if SoapUI does not report a failure or a warning during the tests.

**The result of the attack**

SoapUI didn’t raise any warnings or failures during the attack.



Figure 7 Security report for GetStructureSpecificData

The full report of the attack and the values for each parameter can be found in the [security\_reports] folder under sdmx2.1 subfolder, as security\_scan\_GetStructureSpecificData.txt

### Summary results for the SDMX 2.1 Compliant web services

In the following table a summary of the experiments for the .NET SRI WS Ultimate Solution is presented:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SDMX-ML query | Malformed Requests Sent | Alerts Raised | Vulnerabilities detected | Comments |
| GetCodelist | 824 | 0 | None | N/A |
| GetStructureSpecificData | 973 | 0 | None | N/A |

Table 10 Results of the security testing for SDMX 2.1 web services

## SDMX 2.1 REST Web Services Security Assessment

With the help of SoapUI tool the security assessment was conducted on two sample requests. One concerned the ‘GetCodelist’ operation of the SRI Web Service Ultimate Solution and the other concerned the ‘GetData’ operation. In those query messages were entailed queries for data of specific dataflow, of a specific value of dimension as well as specific time dimension value. For all the queries the SoapUI run the security tests by applying five security scans: fuzzing scan, invalid types, SQL injection, XPath Injection and Cross Site Scripting. The XML attacks done on SOAP services were replaced by the Cross Site Scripting check (specific to HTTP/HTML technology).

Each security scan completes by indicating zero or more alerts. According to the SoapUI tool, alerts are raised when a valid response has been returned from the service and denote possible security vulnerability in the target service. For each security scan a security log is written which entails all the requests sent, the responses and the results.

### Security testing for ‘GetCodelist’ REST request

For the sample of ‘GetCodelist’ request, a query for a specific Codelist was provided as a test query to SoapUI. A snapshot of the query is as follows.

GET http://sodi-test/nsiws/rest/codelist/ALL/CL\_FREQ/ALL/ HTTP/1.1

Accept-Encoding: gzip,deflate

Accept: \*/\*

Host: sodi-test

Connection: Keep-Alive

User-Agent: Apache-HttpClient/4.1.1 (java 1.5)

**The testing strategy**

The testing strategy consists on providing malicious values for the following xml values: agency, Id and version. An example of a malicious request can be:

GET http://sodi-test/nsiws/rest/codelist/**VUzEPcbI2ao**/**W8GdimUE**/**3F6RBNR**/ HTTP/1.1

Accept-Encoding: gzip,deflate

Accept: \*/\*

Host: sodi-test

Connection: Keep-Alive

User-Agent: Apache-HttpClient/4.1.1 (java 1.5)

The number of requests per vulnerability check has been established based on the number or parameters targeted and the complexity of this request.

|  |  |
| --- | --- |
| Vulnerability | Number of tests |
| Fuzzing Scan | 100 fuzz requests sent |
| Invalid Types | 75 invalid xml types requests |
| SQL Injection | 42 sql injections attempts |
| XPath Injection | 30 xpath injection attempts |
| Cross Site Scripting | 276 requests |

Table 11 List of performed scans on GetCodelist REST service

**The expected result**

SoapUI was configured to report a failure in the following situations:

1. The response of the server exposes sensitive information to the attacker after a malicious request has been processed.
2. The Web Service fails to respond as a result of the attack meaning that the server is no longer responding to requests.
3. No malicious scripts are passed back to the web service response ( XSS )

The web service is deemed secure if SoapUI does not report a failure during the tests.

**The result of the attack**

SoapUI raised 3 warnings during the attack but no failures.

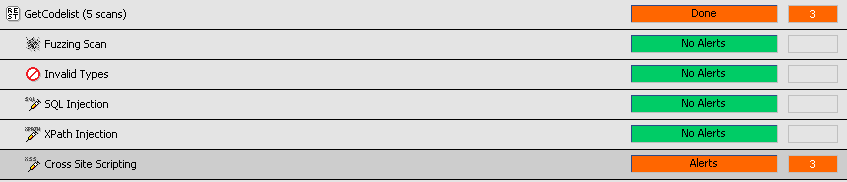


Figure 8 Security report for GetCodelist attack

The three warnings are linked to an empty response when the third parameter contains strings like / \ @ #.These characters are filtered by the IIS server and the webservice is not reached by the http request therefore the response is null:

[Cross Site Scripting] Request 208 - FAILED - [param3=<BODY onload!#$%&()\*~+-\_.,:;?@[/|\]^`=alert("XSS")>]: took 1 ms

-> null/empty response

[Cross Site Scripting] Request 209 - FAILED - [param2=<BODY onload!#$%&()\*~+-\_.,:;?@[/|\]^`=alert("XSS")>]: took 2 ms

-> null/empty response

[Cross Site Scripting] Request 210 - FAILED - [param1=<BODY onload!#$%&()\*~+-\_.,:;?@[/|\]^`=alert("XSS")>]: took 1 ms -> null/empty response

The full report of the attack and the values for each parameter can be found in the [security\_reports] folder, under rest subfolder as security\_scan\_GetCodelist.txt

#### Security testing for ‘GetData’ REST method

For the sample of ‘GetData’ request, a query for a specific Data request was provided as a test query to SoapUI. A snapshot of the query is as follows.

GET http://sodi-test/nsiws/rest/data/ESTAT%2CSSTSCONS\_PROD\_A/....NS0020+NS0030../ALL/?startPeriod=2000 HTTP/1.1

Accept-Encoding: gzip,deflate

Accept: \*/\*

Host: sodi-test

Connection: Keep-Alive

User-Agent: Apache-HttpClient/4.1.1 (java 1.5)

**The testing strategy**

The testing strategy consists on providing malicious values for the following xml values: agency id, dataflow id and dimensions. An example of a malicious request can be:

GET http://sodi-test/nsiws/rest/data/**8bPjgt**/**9fzB3Oh9cvBHBlN**/**ex9rJrppIK**/?startPeriod=**OO1V6y0TSR5MK** HTTP/1.1

Accept-Encoding: gzip,deflate

Accept: \*/\*

Host: sodi-test

Connection: Keep-Alive

User-Agent: Apache-HttpClient/4.1.1 (java 1.5)

The number of requests per vulnerability check has been established based on the number or parameters targeted and the complexity of this request.

|  |  |
| --- | --- |
| Vulnerability | Number of tests |
| Fuzzing Scan | 100 fuzz requests sent |
| Invalid Types | 100 invalid xml types requests |
| SQL Injection | 56 sql injections attempts |
| XPath Injection | 40 xpath injection attempts |
| Cross Site Scripting | 276 requests |

Table 12 List of performed scans on GetData REST service

**The expected result**

SoapUI was configured to report a failure in the following situations:

1. The response of the server exposes sensitive information to the attacker after a malicious request has been processed.
2. The Web Service fails to respond as a result of the attack meaning that the server is no longer responding to requests.
3. No malicious scripts are passed back to the web service response ( XSS )

The web service is deemed secure if SoapUI does not report a failure during the tests.

**The result of the attack**

SoapUI raised 3 warnings during the attack but no failures.

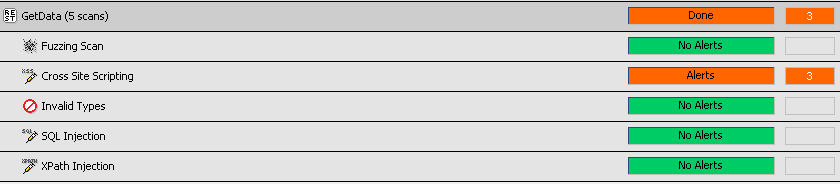


Figure 9 Security Report for the attack on GetData method

The three warnings are linked to an empty response when the third parameter contains special URL characters (like / \ $ % & @ ) which are filtered by the IIS server therefore no response provided:

[Cross Site Scripting] Request 277 - FAILED - [param3=<BODY onload!#%&()\*~+-\_.,:;?@[/|\]^`=alert("XSS")>]: took 1 ms

-> null/empty response

[Cross Site Scripting] Request 279 - FAILED - [param2=<BODY onload!#$%&()\*~+-\_.,:;?@[/|\]^`=alert("XSS")>]: took 2 ms

-> null/empty response

[Cross Site Scripting] Request 280 - FAILED - [param1=<BODY onload!#$%&()\*~+-\_.,:;?@[/|\]^`=alert("XSS")>]: took 1 ms -> null/empty response

The full report of the attack and the values for each parameter can be found in the [security\_reports] folder, under rest subfolder as security\_scan\_rest\_GetData.txt

### Summary results for the SRI Rest web services security test

In the following table a summary of the experiments for the .NET SRI WS Rest Web service is presented:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SDMX query | Malformed Requests Sent | Alerts Raised | Vulnerabilities detected | Comments |
| GetCodelist | 523 | 3 | None | The alerts raised by SoapUI are related to the null response received from the server. No real security threat has been found. |
| GetData | 572 | 3 | None | The alerts raised by SoapUI are related to the null response received from the server. No real security threat has been found. |

Table 13 Results of security scans for REST services

## SRI AuthModule

The SRI AuthModule is a .NET server module. It provides authentication and authorization support, using plug-able implementations for retrieving credentials, encrypting passwords, providing authentication and providing authorization.

Currently the SRI AuthModule uses HTTP Header Basic authentication to retrieve user credentials. HTTP Header Basic authentication relies on the fact that the HTTP connection between the server and client is secure and trusted.  When used without HTTPS it is not secure because it transmits the user credentials in plain text. When used with HTTPS (encrypted HTTP transport channel), then the user credentials are encrypted.

For the SRI Auth Module the security scans are related to the http basic authentication. A possible vulnerability is the parsing of the HTTP header containing the basic authentication.

|  |
| --- |
| GET /nsiws/NSIStdV20Service.asmx?wsdl HTTP/1.1  Host: sodi-test  Authorization: Basic T1IgMT0xIOKAkyA6YmxhaGJsYWg= |

The user is extracted from the base64 encoded string and used as input in a query against the authentication database and in a query the authorization database. In both cases, SQL parameters are used.

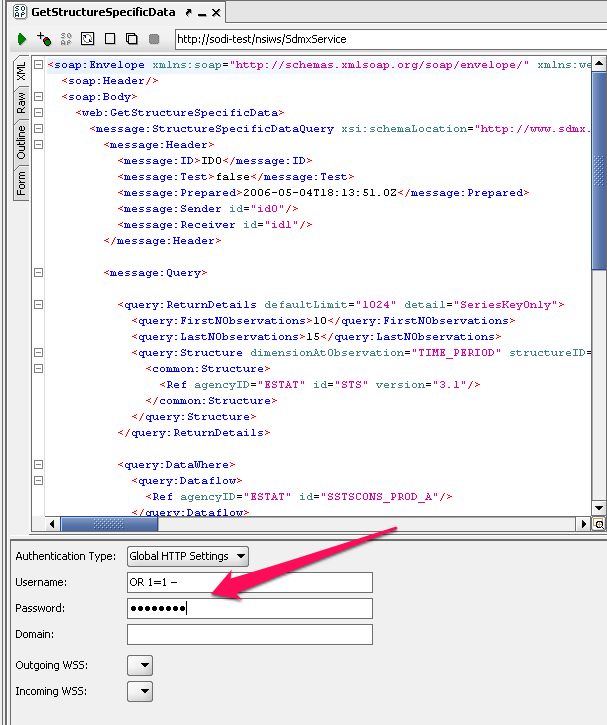


Figure 10 Attack the http basic authentication

When sending the request, the password can be seen as a Base64 encoded text:

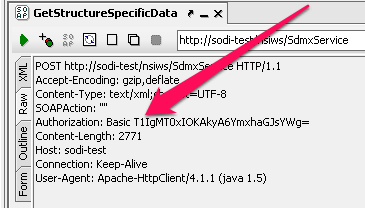


Figure 11 Basic Authentication mechanism with base64 encoded text

# Security assessment of .Net NSI Web Client 3.4.0

The Web Client application is the more susceptible to an external attack. However, since it does not access the database directly, it is less exposed to attacks.

## OWASP[[1]](#footnote-1)

Reports of vulnerabilities in web applications have risen exponentially. Exploits are easy to develop and targets are easy to find. Web applications are commonly used from a web browser and they cover a range of activities, such as e-banking, webmail, online shopping, etc. In recent years the development of such applications has been considerable and today rich internet applications offer complex, real-time interactions with users. While web applications have become omnipresent, they also present new security risks. It is important to identify and understand these risks when developing, hosting or simply using these applications.

The Open Web Application Security Project (OWASP) is an open community dedicated to enabling organizations to develop, purchase, and maintain applications that can be trusted.

OWASP has created a list of web applications vulnerabilities widely known as OWASP Top 10.

The OWASP Top 10 is a set of classes of vulnerabilities that are very high risk. Application developers can judge whether their applications meet best practices based on whether or not they has facilities to protect against these vulnerabilities. The OWASP Top 10 represents a broad consensus regarding the most critical vulnerabilities for web application security. A variety of security experts from around the world contribute their expertise to produce the OWASP Top 10. The last version of the list is the Top 10 for 2013 (see <https://www.owasp.org/index.php/OWASP_Top_Ten>).

The following vulnerabilities, in descending order of severity, comprise the OWASP Top 10 for 2010:

* A1 – Injection
* A2 – Broken Authentication and Session Management
* A3 - Cross Site Scripting (XSS)
* A4 – Insecure Direct Object References
* A5 – Security Misconfiguration
* A6 – Sensitive Data Exposure
* A7 – Missing Function Level Access Control
* A8 – Cross Site Request Forgery (CSRF)
* A9 – Using Components with Known Vulnerabilities
* A10 – Unvalidated Redirects and Forwards

## Samurai Web Testing Framework

In order to assess the vulnerability of this application, the Samurai Web Testing Framework has been used. The Samurai Web Testing Framework is a web penetration testing Live CD[[2]](#footnote-2) built on open source software.

Live CD's for penetration testing are becoming more prevalent these days, with a wider diversity of offerings. Live CD's allow testers the ability to run pre configured tools from operating systems they might not otherwise have easy access to. A live CD comes with a full operating system and several tools already installed on them. Utilizing live CD's to either directly boot your machine or from a virtual environment, gives penetration testers maximum flexibility in operating systems and tools at their disposal.

Samurai focuses on tools needed by web application testers to look for common vulnerabilities, such as misconfigurations, cross site scripting (XSS), SQL injection, remote file inclusion and other common vulnerabilities. The CD includes several tools to obtain information about web applications and servers, enumerate files and directories, and test scripts. A summary of the tools available in Samurai Web Testing Framework is presented on <http://www.madirish.net/node/218>.

## Analysis of Web Client application risks using Samurai

The list of risks presented in section 3.2.1 has been analysed in the context of the Web Client application and Samurai Web Testing Framework.

### A1 Injection

Injection flaws, such as SQL, OS, and LDAP injection, occur when un-trusted data is sent to an interpreter as part of a command or query. The attacker’s hostile data can trick the interpreter into executing unintended commands or accessing unauthorized data.

For NSI Client the SQL Injection attack was done with the help of OWASP ZAP security tool (part of Samurai Live CD). The test strategy consists of providing the usual malicious parameters as replacement for normal business data.

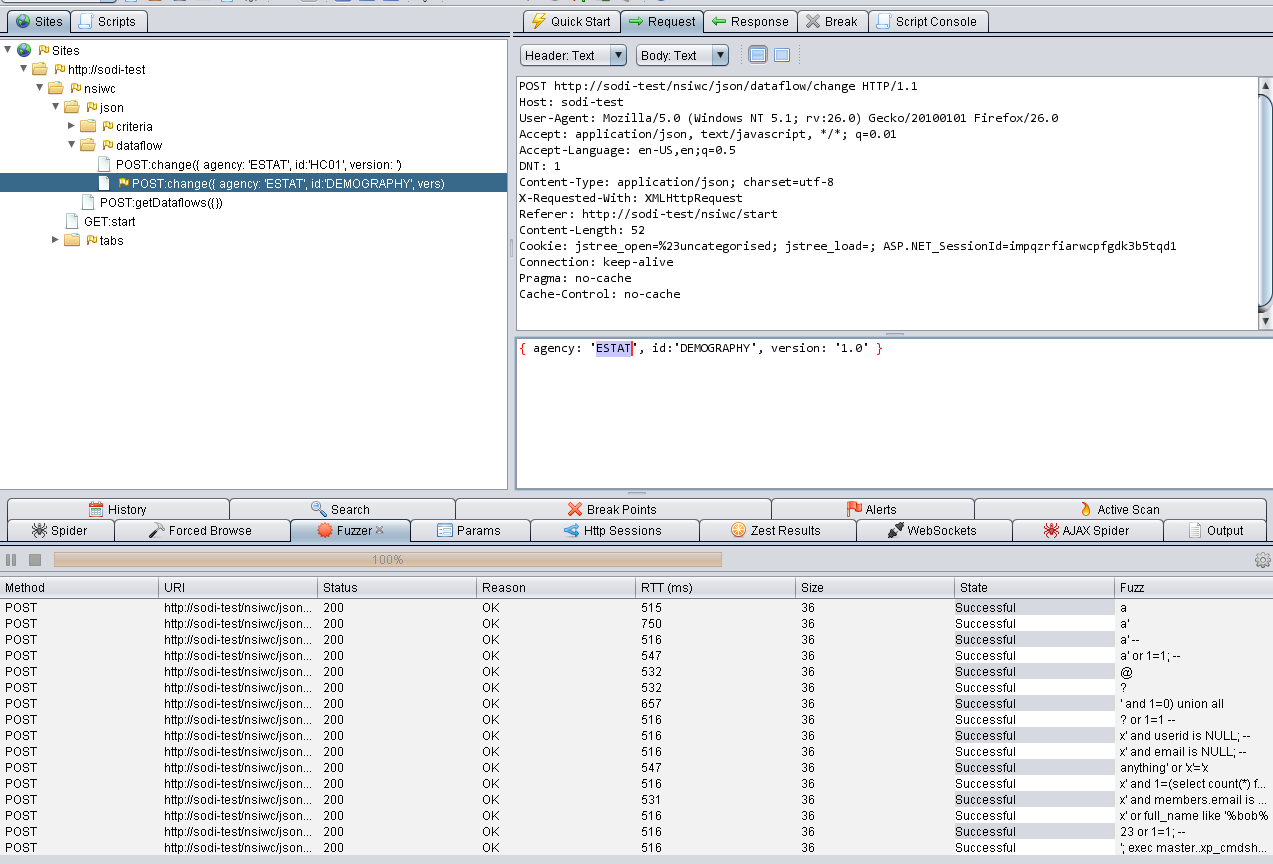


Figure 12 ZAP Security tool performing an SQL Injection attack

#### Testing SQL Injection on the POST “dataflow change” ajax request

An example of a malicious Sql Injection on the dataflow change ajax request can be seen below:

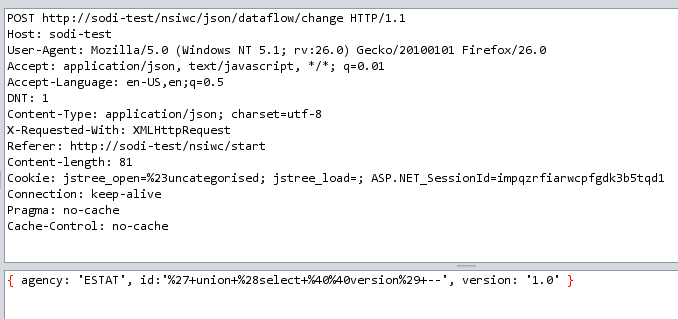


Figure 13 SQL Injection attack on “dataflow change” ajax request

The test consisted of around 20 SQL Injection attacks out of which no failures and no warnings have been raised. Unless we consider the error message HTTP 404 as vulnerability (which means that no data was found for the given parameters), there is no reason to consider the “dataflow change” as vulnerable.

The full report of the attack can be found in the [security\_reports] folder, under “Web Client security assessment” subfolder as sql-injection-on-dataflow-change.txt.

#### Testing SQL Injection on the “update results” ajax POST request

An example of an SQL injection request for the “update results” POST request is:

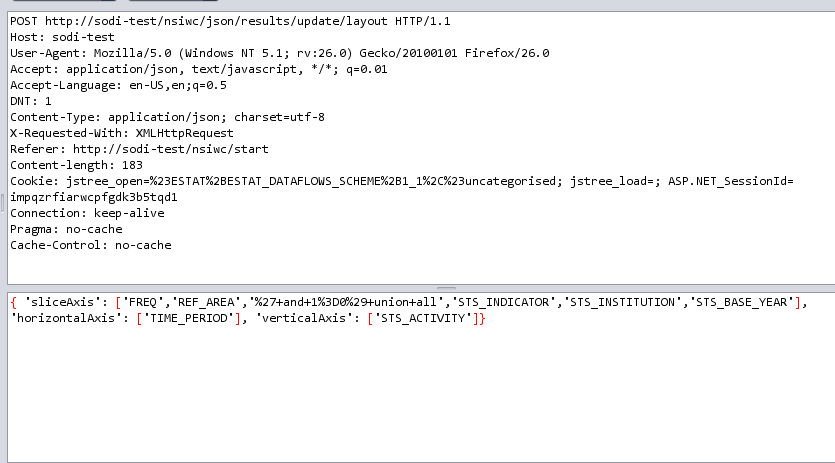


Figure 14 SQL Injection attack on “results update” ajax request

The full attack consisted of 20 SQL Injection attempts which resulted in no failures and no warnings. Unless we consider the error message HTTP 404 as vulnerability (which means that no data was found for the given parameters), we can consider the “results update” ajax call a secure method from the SQL Injection point of view.

The full report of the attack can be found in the [security\_reports] folder, under “Web Client security assessment” subfolder as sql-injection-on-update-results.txt.

### A2 Broken Authentication and Session Management

Application functions related to authentication and session management are often not implemented correctly, allowing attackers to compromise passwords, keys, session tokens, or exploit other implementation flaws to assume other users’ identities.

For NSI Web Client the authentication and session management was tested with the help of the following plug-ins of the OWASP ZAP Attack Proxy:

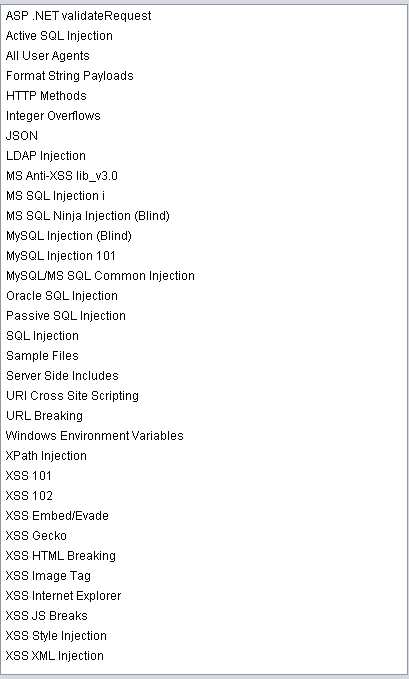


Figure 15 List of plugins used during Broken Auth. and Session Management test

Here’s a screenshot of Zap Fuzzer while performing an attack:

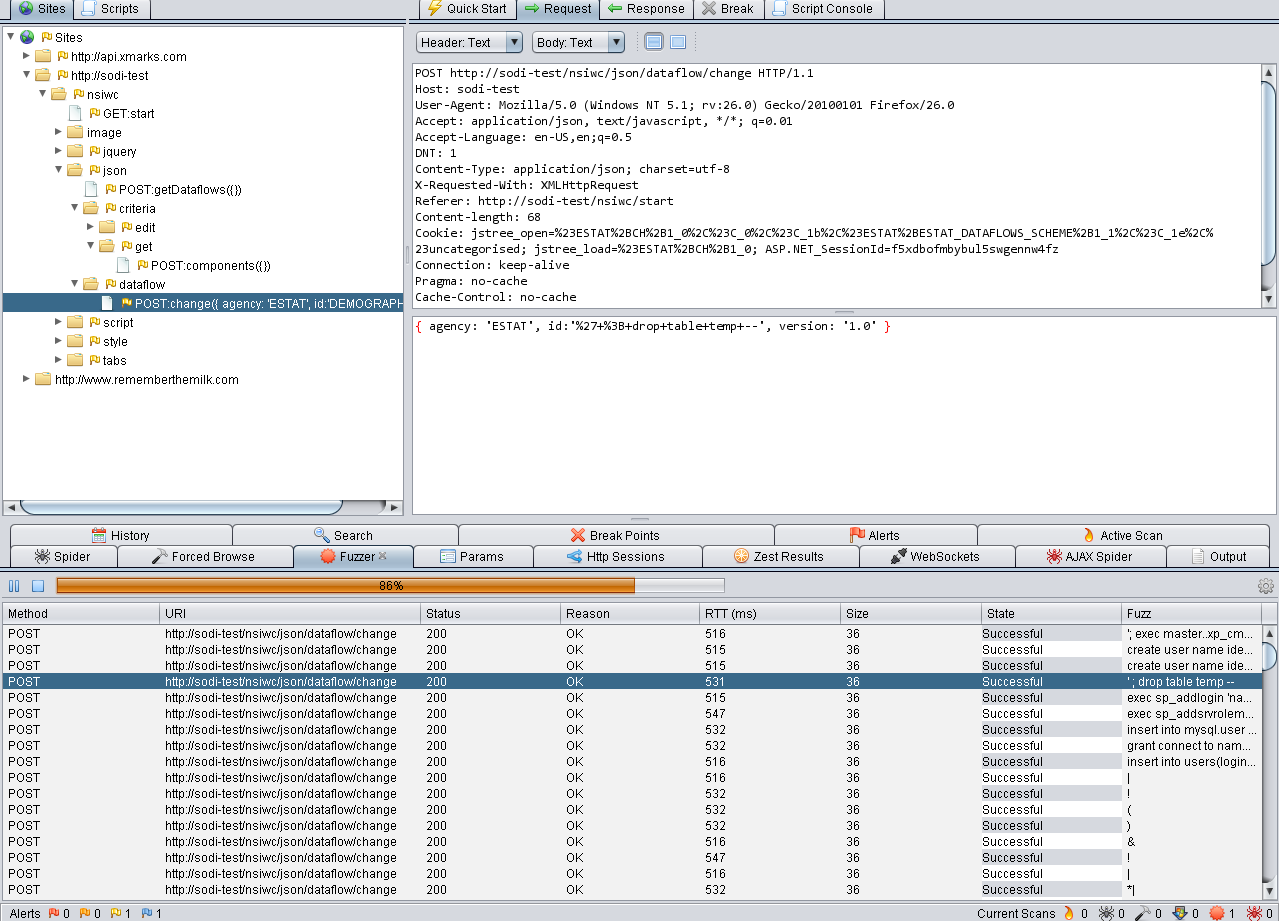


Figure 16 Fuzzer performing a fuzz attack for Broken Auth. and Session Mgmt

#### Checking the Broken Authentication and Session management on “dataflow change” ajax request by fuzzing the requests

A sample malicious fuzz attack on the “dataflow change” ajax request is:

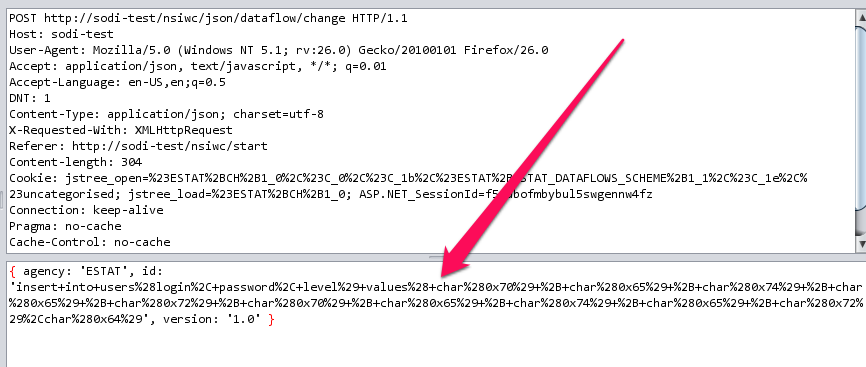


Figure 17 Fuzz attack sample on “dataflow change” ajax request

199 fuzz requests were tested and completed with no failures. ZAP Fuzzer produced two warnings of low priority concerning session management. In particular, the vulnerability identified concerns the accessibility of session id to malicious users. Nevertheless, this session id’s purpose is to keep information concerning the client and the dataflow he has access to. This is not secure information so this vulnerability is not considered as a serious security issue.

The full listing of the attacks can be found in the [security\_reports] folder, under “Web Client Security Assessment” as fuzz-scan-on-dataflow-change.txt file.

#### Checking the Broken Authentication and Session management on “results update” ajax request

A sample fuzz request for the “results update” ajax call can be:

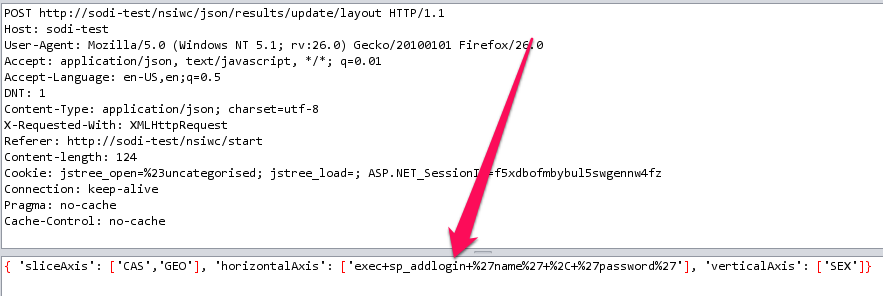


Figure 18 Fuzz attack on “results update” ajax request

The same 199 malicious requests have been tested against “results update” and completed with no failures. ZAP Fuzzer produced only two warnings of low priority concerning session management. In particular, the vulnerability identified concerns the accessibility of session id to malicious users. Nevertheless, this session id’s purpose is to keep information concerning the client and the dataflow he has access to. This is not secure information so this vulnerability is not considered as a security issue.

The full list of requests and results of the scan can be found under the [security\_reports] folder, under “Web Client Security Assessment” as fuzz-scan-on-results-update.txt file.

### A3 - Cross site scripting

XSS flaws occur whenever an application takes un-trusted data and sends it to a web browser without proper validation and escaping. XSS allows attackers to execute scripts in the victim’s browser that can hijack user sessions, deface web sites, or redirect the user to malicious sites.

In order to test the XSS vulnerabilities of the NSI WebClient we’ve used the following plug-ins found in Samurai - Zed Attack Proxy:

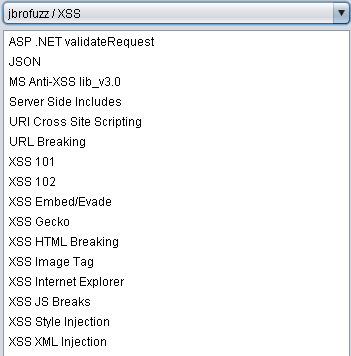


Figure 19 List of plugins used for the XSS attack

#### XSS attack on “dataflow change”

A sample XSS request extracted from an XSS 101 attack from ZAP could be:



Figure 20 Sample XSS request on “dataflow change” ajax request

The full security test on “dataflow change” consisted of 280 attacks. There were 0 failures, 0 warnings and 280 successfully rejected attacks.

The full listing of the scanned URLs and the parameters used can be found in the [security\_reports] folder, under “Web Client Security Assessment” as xss-scan-on-dataflow-change.txt file.

#### XSS attack on “results update”

A similar attack has been performed for the “results update” ajax method. In this case a set of 223 requests have been sent to the web server. No failures or warnings have been raised during the attack.

The full listing of the scanned URLs and the parameters used can be found in the [security\_reports] folder, under “Web Client Security Assessment” as xss-scan-on-results-update.txt file.

### A4 - Insecure Direct Object References

A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key. Without an access control check or other protection, attackers can manipulate these references to access unauthorized data.

In NSI Client there are no direct object references in the application therefore there is no risk for this vulnerability.

### A5 - Security miss-configuration

Good security requires having a secure configuration defined and deployed for the application, frameworks, application server, web server, database server, and platform. All these settings should be defined, implemented, and maintained and the application should be shipped with secure defaults. This includes keeping all software up to date, including all code libraries used by the application.

In this case, the security of the configuration depends on the environment where the application server and the database server are available. NSI Client, as it is shipped, provides secure settings, up-to-date libraries so we can deem the package as well configured.

### A6 – Sensitive Data Exposure

Many web applications do not properly protect sensitive data such as IDs, and authentication credentials. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data deserves extra protection such as encryption at rest or in transit, as well as special precautions when exchanged with the browser.

This is not applicable for the SRI.Net Web Client, as the main purpose of our stored data is public dissemination and no sensitive data is stored in the system.

### A7 - Missing Function Level Access Control

Most web applications verify function level access rights before making that functionality visible in the UI. However, applications need to perform the same access control checks on the server when each function is accessed. If requests are not verified, attackers will be able to forge requests in order to access functionality without proper.

There is no access control performed in the front end (i.e. NSI Web Client) that is not done also on the server side, given the Web Client has been developed independently and in front of the Web Service. Thus, the data presented depend exclusively on what the Web Service serves to the Web Client and there is no extra authorization required.

### A8 - Cross Site Request Forgery

A CSRF attack forces a logged-on victim’s browser to send a forged HTTP request, including the victim’s session cookie and any other automatically included authentication information, to a vulnerable web application. This allows the attacker to force the victim’s browser to generate requests the vulnerable application thinks are legitimate requests from the victim.

The OWASP ZAP was configured to look for CRSF vulnerability by testing the whole site against the Active Scan tool by using the following plug-ins:

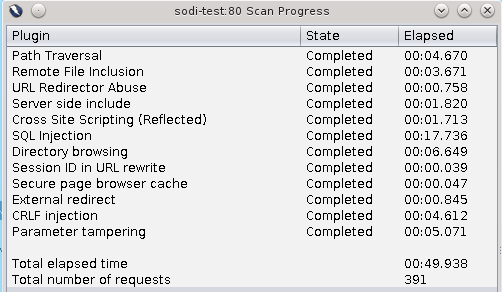


Figure 21 List of scans planned for CSRF security testing

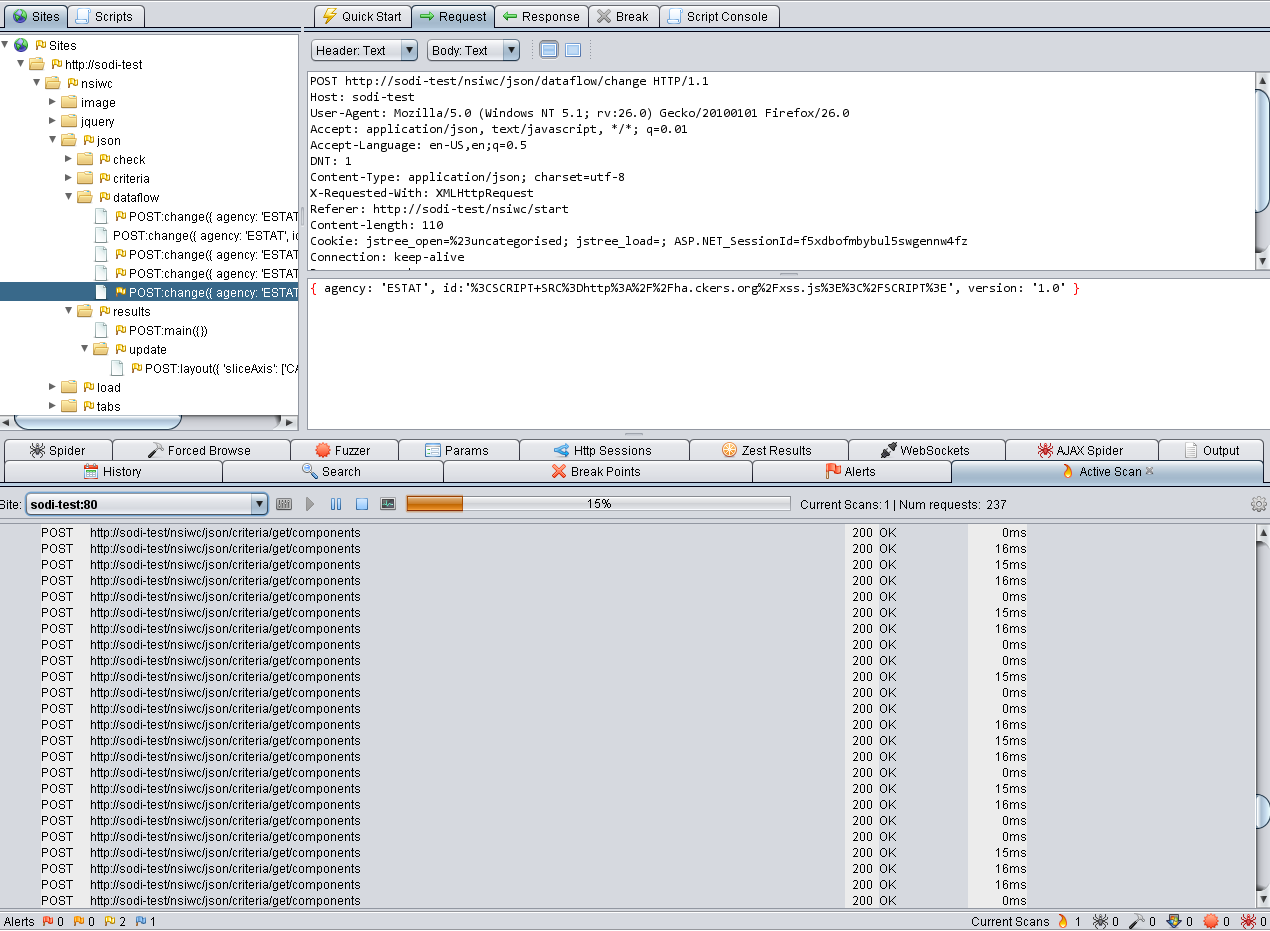


Figure 22 ZAP Active Scan performing a CSRF attack on NSI Web Client

An example of an active-scan malicious request can be:

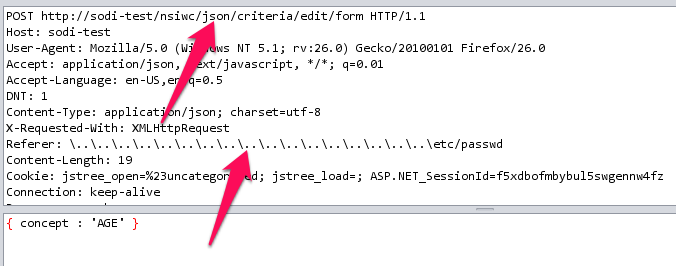


Figure 23 A sample malicious active scan request for “criteria edit” ajax request

391 malicious requests have been sent to NSI Web Client, none of them successful as reported by Active Scan:

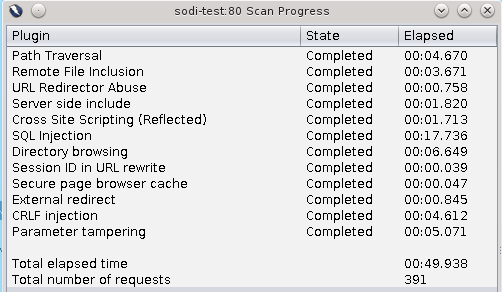


Figure 24 Results of the Active Scan security test

The full list of requests has been provided in the [security\_reports] folder, under “Web Client Security Assessment” as active-scan.txt.

### A9 - Using Components with Known Vulnerabilities

Components, such as libraries, frameworks, and other software modules, almost always run with full privileges. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications using components with known vulnerabilities may undermine application defences and enable a range of possible attacks and impacts.

NSI Web Client .NET uses the following external libraries:

|  |  |  |
| --- | --- | --- |
| Library | Version | Known security issues |
| SQLite Interop Library | 1.0.65 | None |
| log4net | 2.0.3 | None |

Table 14 List of external libraries used in .Net NSI Web Client

No libraries or frameworks with known vulnerabilities have been identified.

### A10 - Invalidated Redirects and Forwards

Web applications frequently redirect and forward users to other pages and websites, and use un-trusted data to determine the destination pages. Without proper validation, attackers can redirect victims to phishing or malware sites, or use forwards to access unauthorized pages.

In the case of NSI WebClient, there are no external links in the application therefore there is no risk for this vulnerability.

### Summary results for Web Client security testing

Below is presented the list of risks that might be considered for the Web Client application and the tools from Samurai that will be used to study the behaviour:

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Vulnerability | Tool used to analyse | Number of warnings or vulnerabilities found |
| A1 | Injection | OWASP ZAP - Injection plugins | None |
| A2 | Broken Authentication and Session Management | ZAP Fuzzer | None |
| A3 | Cross Site Scripting (XSS) | XSS Fuzz plugins of the OWASP Zap Attack Proxy | None |
| A4 | Insecure Direct Object References | No tool used | N/A |
| A5 | Security Miss configuration | No tool used | N/A |
| A6 | Sensitive Data Exposure | N/A | None |
| A7 | Missing Function Level Access Control | N/A | None |
| A8 | Cross Site Request Forgery (CSRF) | ZAP Active Scan | None |
| A9 | Using Components with Known Vulnerabilities | N/A | None |
| A10 | Un-validated Redirects and Forwards | No tool used | N/A |

Table 15 Results of the security tests for .Net NSI Web Client

1. <https://www.owasp.org/index.php/Main_Page> [↑](#footnote-ref-1)
2. A live CD, live DVD, or live disc is a [CD](http://en.wikipedia.org/wiki/CD) or [DVD](http://en.wikipedia.org/wiki/DVD) containing a [bootable](http://en.wikipedia.org/wiki/Booting) computer operating system. Live CDs are unique in that they have the ability to run a complete, modern operating system on a computer lacking [mutable secondary storage](http://en.wikipedia.org/wiki/Computer_storage), such as a hard disk drive. [↑](#footnote-ref-2)