Web Services Penetration Testing

Chapter 1

Introduction:

Web application security is quite popular among the pen testers. So, organizations, developers and pen testers treat web applications as a primary attack vector. As web services are relatively new as compared to web applications, it's considered as secondary attack vector. Due to lack of concern or knowledge it is generally found that security measures implemented in a web service is worse than what is implemented in web applications. Which makes the web service a favorite attack vector and easy to penetrate as per the attacker's point of view.

Another reason to write this article is that the use of web services increased in last couple of years in a major ratio and, also the data which flows in web services are very sensitive. This makes web services again an important attack vector.

The use of web services increased suddenly because of mobile applications. As we all know the growth of usage for mobile applications has increased rapidly, and most mobile applications use some sort of web service. Which has relatively increased the use of web services. Web services are also mostly used by enterprise level software which carries a lot of sensitive data. Due to the lack of security implementations and resources available, web services play a vital role making it a possible attacking vector.

In this article we will focus on details of web services, its testing approach, tools used for testing etc.

SOA:

Before starting to penetrate a web service we must know its basics. As a web service is the implementation of SOA. Let's start with SOA.

SOA stands for **S**ervice **O**riented **A**rchitecture. According to Wikipedia "**Service-oriented architecture** (**SOA**) is a software design and software architecture design pattern based on discrete pieces of software that provide application functionality as services, known as Service-orientation. A service is a self-contained logical representation of a repeatable function or activity. Services can be combined by other software applications that together, provide the complete functionality of a large software application".

In simple words it is quite similar, to client server architecture but here a client is a service consumer and server is a service provider. Service is a well-defined activity that does not depend on the state of other services. A service consumer requests a particular service in the format used by the service provider and the service provider returns with a service response as shown in Fig 1.

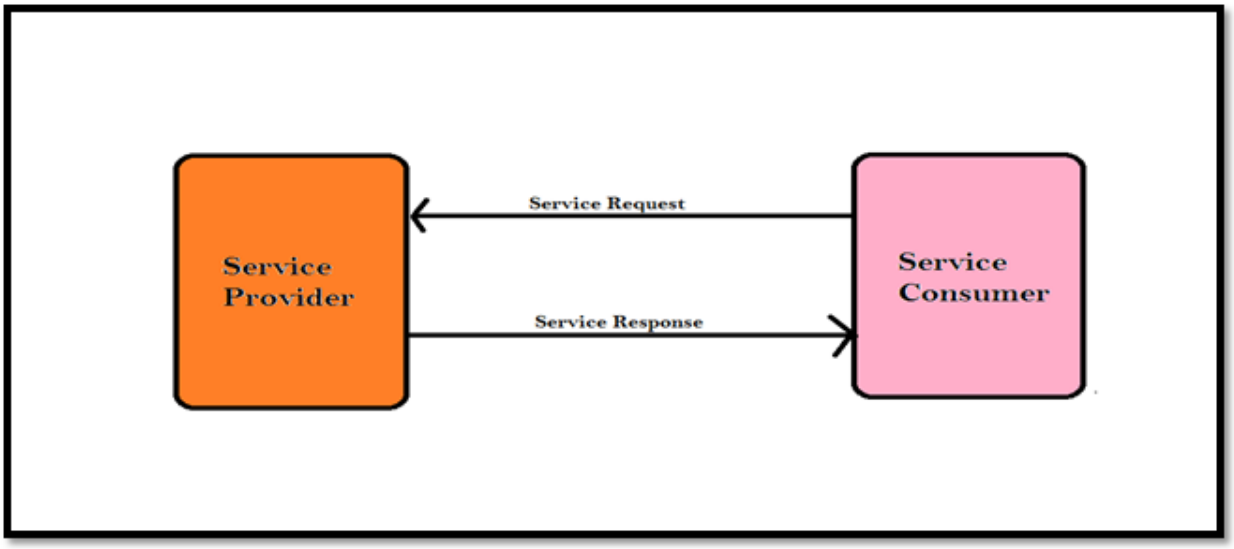


Fig 1: Service Oriented Architecture (SOA)

What is Web Service?

A Web service is a standardized way of establishing communication between two Web-based applications by using open standards over an internet protocol backbone. Generally, web applications work using HTTP and HTML, but web services work using HTTP and XML. Which as, added some advantages over web applications. HTTP is transfer independent and XML is data independent, the combination of both makes web services support a heterogeneous environment.

Why use Web Service?

Web services have some added advantages over web applications. Some are listed below:

1. Language Interoperability (Programming language independent)
2. Platform Independent (Hardware and OS independent)
3. Function Reusability
4. Firewall Friendly
5. Use of Standardized Protocols
6. Stateless Communication
7. Economic

Difference between Web Application and Web Services:

A web application is an application that is accessed through a web browser running on a client's machine whereas a web service is a system of software that allows different machines to interact with each other through a network. Most of the times, web services do not necessarily have a user interface since it's used as a component in an application, while a web application is a complete application with a GUI. Furthermore, web services will take a web application to the next level because it's used to communicate or transfer data between web applications that run on different platforms allowing it to support a heterogeneous environment.

Components of Web Services:

1. Service Consumer
2. Service Provider
3. XML (Extensible Markup Language)
4. SOAP (Simple Object Access Protocol)
5. WSDL (Web Services Description Language)
6. UDDI (Universal Description, Discovery and Integration)

Service Consumer and Service Provider:
are applications that can be written in any programming language. The work of both these components is already mentioned in SOA division.

Extensible Markup Language (XML):
is used to encode data and form the SOAP message.

Simple Object Access Protocol (SOAP):
is a XML-based protocol that lets applications exchange information over HTTP. Web services use a SOAP format to send XML requests. A SOAP client sends a SOAP message to the server. The server responds back again with a SOAP message along with the requested service. The entire SOAP message is packed in a SOAP Envelope as shown in Fig 2.

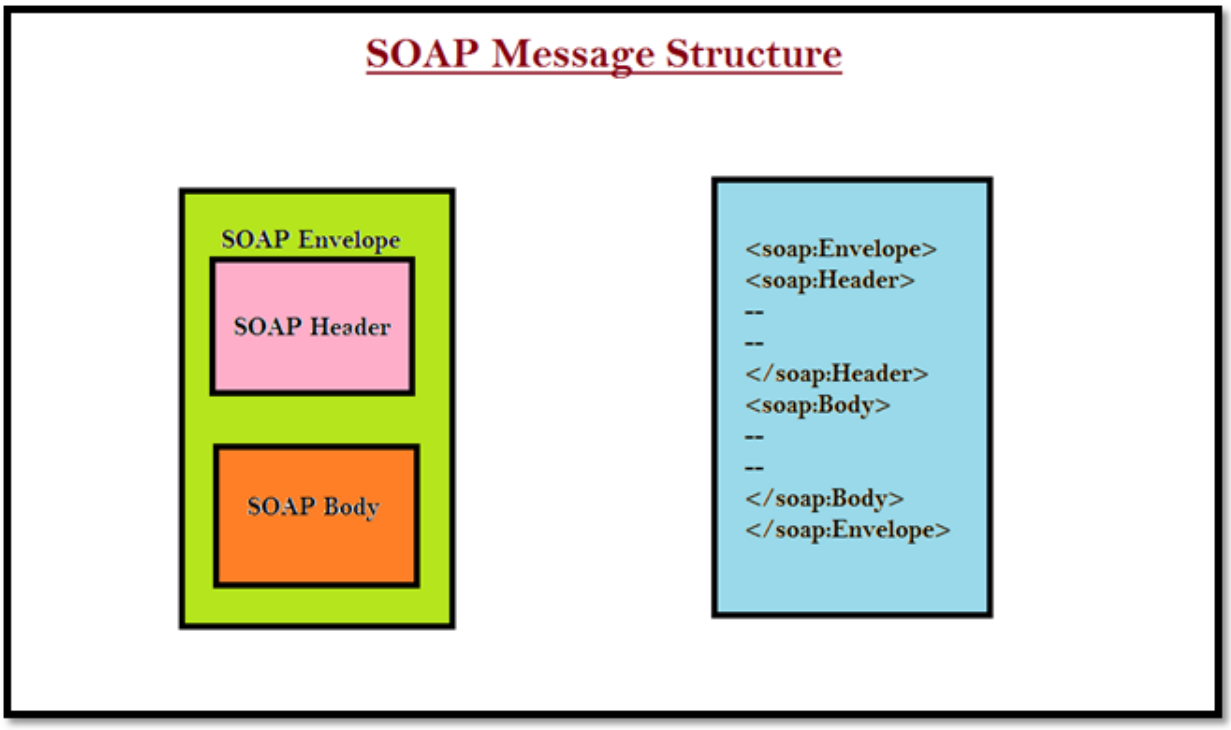


Fig 2: SOAP Message Structure
The actual data flows in the body block and the metadata is usually carried by the header block.

A typical SOAP request looks like Fig 3.

POST /ws/ws.asmx HTTP/1.1
Host: www.example.com
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://www.example.com/ws/IsValidUser"

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
<IsValidUser xmlns="http://www.example.com/ws/">
<UserId>string</UserId>
</IsValidUser>
</soap:Body>
</soap:Envelope>
```

Fig 3: SOAP Request

If the service consumer sends a proper SOAP request then the service provider will send an appropriate SOAP response. A typical SOAP response looks like Fig 4.

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
<IsValidUserResponse xmlns="http://www.example.com/ws/">
<IsValidUserResult>boolean</IsValidUserResult>
</IsValidUserResponse>
</soap:Body>
</soap:Envelope>
```

Fig 4:
SOAP Response

Web Services Description Language (WSDL): is really an XML formatted language used by UDDI. It describes the capabilities of the web service as, the collection of communication end points with the ability of exchanging messages. Or in simple words "Web Services Description Language is an XML-based language for describing Web services and how to access them".

As per pen testing web services are concerned, understanding of WSDL file helps a lot in manual pen testing. We can divide WSDL file structure in to two parts according to our definition. 1st part describes what the web service and the 2ndparts tells how to access them. Let's start with basic WSDL structure as shown in Fig 5.

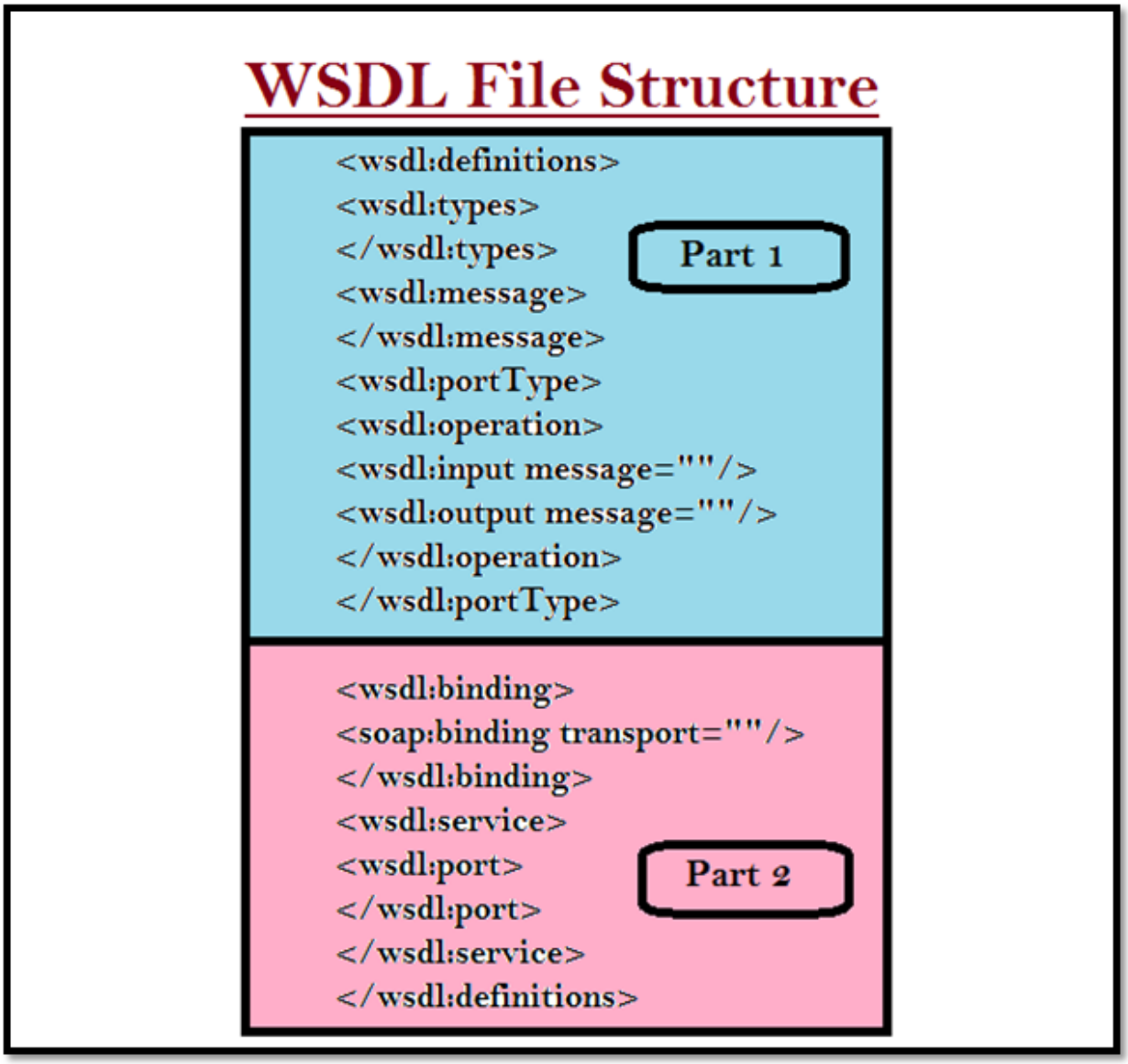


Fig 5: Basic WSDL File Structure

The Fig 5 image only focuses on some of the important elements of the WSDL file. What the element exactly contains is defined in Table 1.

Elements	**What it contains**
definitions	All the XML elements are packed under definition element. It is also called as root or parent element of the WSDL file.
types	All the schema types or data types defined here.
message	This is a dependent element. Message is specified according to the data types defined in ***types*** element. And used in side ***operation*** element later.
portType	Element collects all the operations within a web service.
operation	Collection of input, output, fault and other message as specified in ***message*** element.
input message	It's nothing but the parameters of the method used in SOAP request.

output message	It's nothing but the parameters of the method used in SOAP response.
binding	This element connects part 2 of WSDL file with part1 associating itself to the ***portType*** element and allows to define the protocol you want to use.
soap:binding	It formulates the SOAP message at runtime.
service	Contains name of all the services provided by the service provider.
port	It provides the physical path or location of web server so that service consumer can connect with service provider.

Table 1: Defining Different Elements of WSDL File

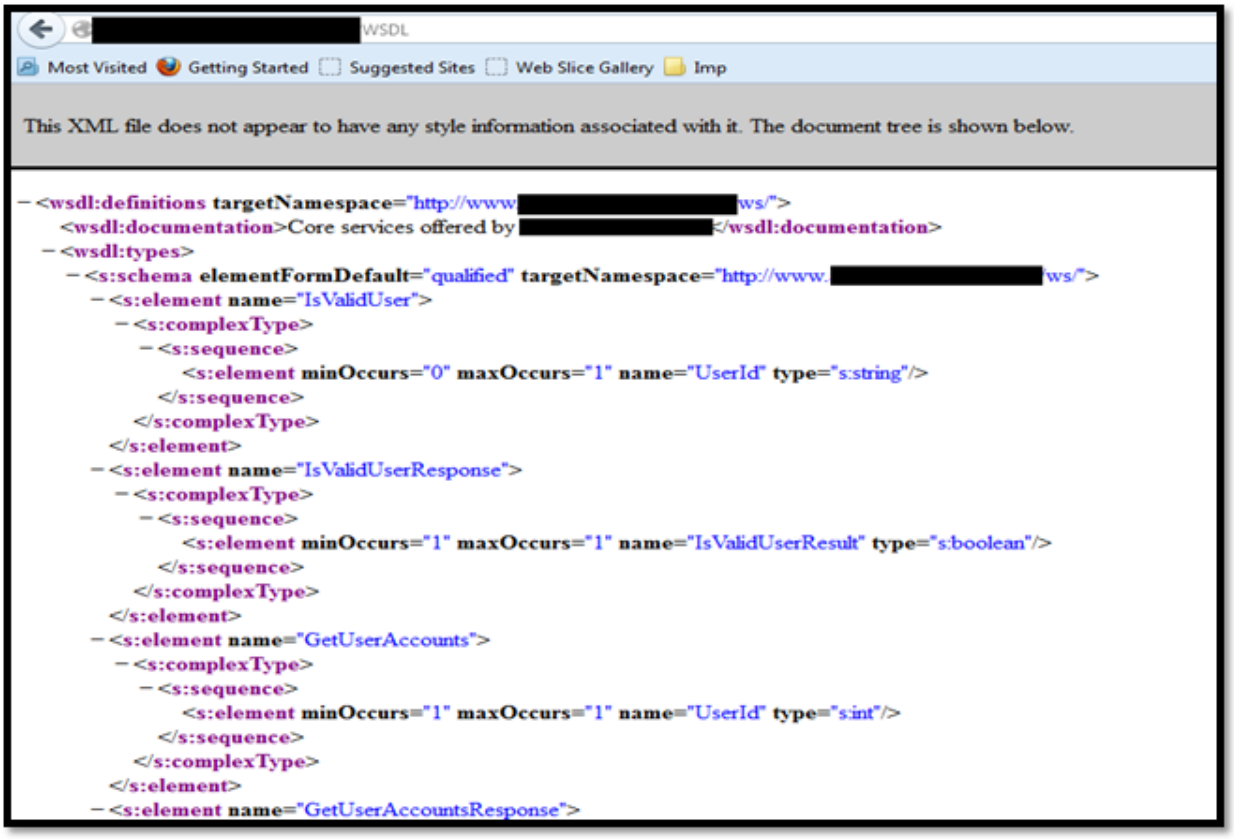


Fig 6: A WSDL file

Universal Description, Discovery and Integration (UDDI): is a distributive directory on the web, where every service provider who needs to issue registered web services using its WSDL. The service consumer will search for appropriate web services and UDDI will provide the list of service providers offering that particular service. The service consumer chooses one service provider and gets the WSDL.

A typical UDDI link looks like Fig 7.

http://anything.example.org/juddi/inquiry

Fig 7: UDDI Link

What are Web Services?

Let's redefine the web services from all the things what we've covered above. "Web services are a standardized way of establishing communication between two Web-based applications by using XML, SOAP, WSDL, UDDI and open standards over an internet protocol backbone. Where XML is used to encode the data in the form of a SOAP message. SOAP is used to exchange information over HTTP, WSDL and is used to describe the capabilities of web services and UDDI is used to provide the list of service provider details as shown in Fig 8."

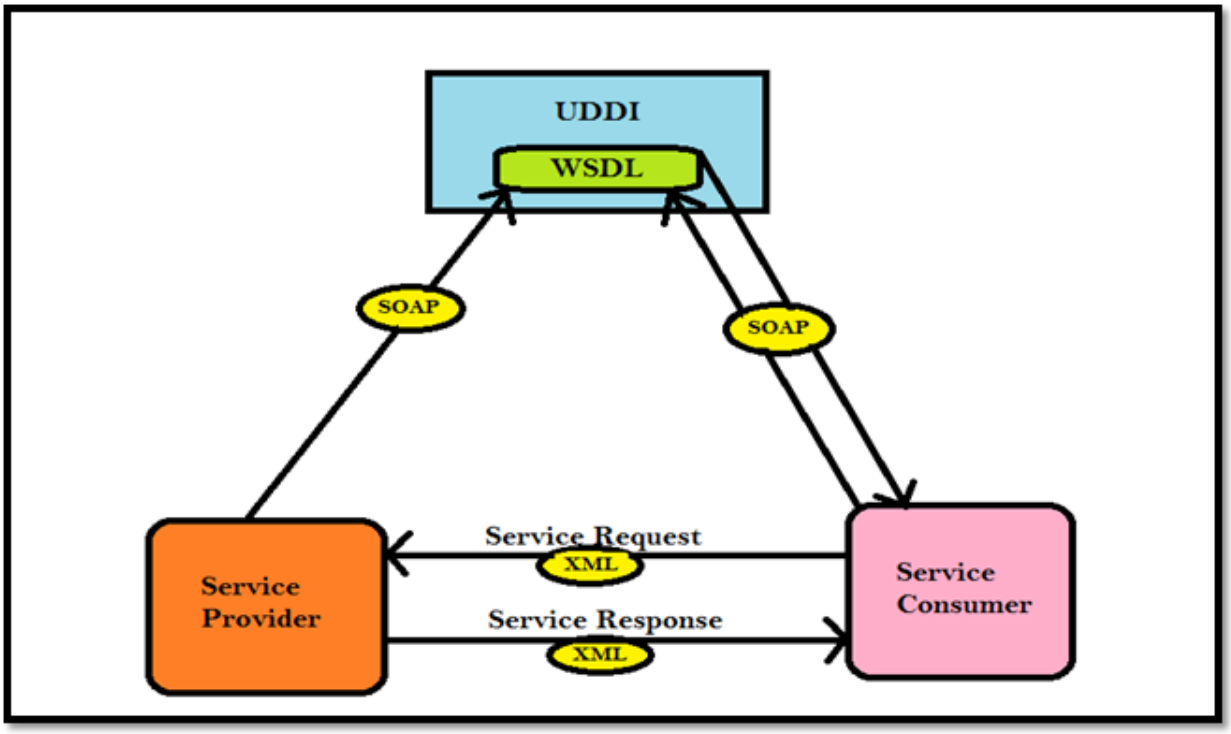


Fig 8: Web Service Description

In a real-time scenario if a service consumer wants to use some sort of web service, then it must know the service provider. If a service provider validates a service consumer it will provide the WSDL file directly and then the service consumer creates a XML message to request for a required service in the form of a SOAP message and the service provider returns a service response.

On other hand if a service consumer is not aware of the service provider, it will visit UDDI and search for the required service. The UDDI returns the list of service providers offering that particular service. Then by choosing one service provider again the service consumer generates a XML message to request for a required service in the form of a SOAP message, as specified in the WSDL file of that service provider. The service provider then returns a service response. Generally, in web service testing we assume the service consumer and the service provider know each other, so to start testing a web service we must ask for the WSDL file.

How to test Web Services?

The testing approach of web services is quite similar to the testing approach used in web applications. Though there are certain differences, but we will discuss those at a later time. Web services testing is categorized in 3 types:

1. Black Box Testing

2. Grey Box Testing
3. White Box Testing

In black box testing the tester has to focus more on authentication because he/she will be provided only with WSDL file. I prefer grey box most, because it's better to have some sample requests and responses to analyze the web services better. It will help to understand the user roles, authentication mechanism and data validations etc.

Depending upon the scope and scenario, our testing methodology will change. We will focus on all these testing approaches but to start with now we will use black box testing.

Where to Start?

Let's say that you want to test for web services associated with a web application http://www.example.com, it's a black box testing and you have no details of the web service associated. (Generally, if a client wants to test their web services they will provide you the WSDL file but for now we assume that we don't have the WSDL file).

Then you can start from web services fingerprinting. As we already covered that all the web services descriptions are present in WSDL so you can use google to fingerprint the WSDL file of that particular web application using special notations such as filetype shown in Fig 9.

www.example.com filetype: WSDL

Fig 9: Use of Google Dork to Find WSDL

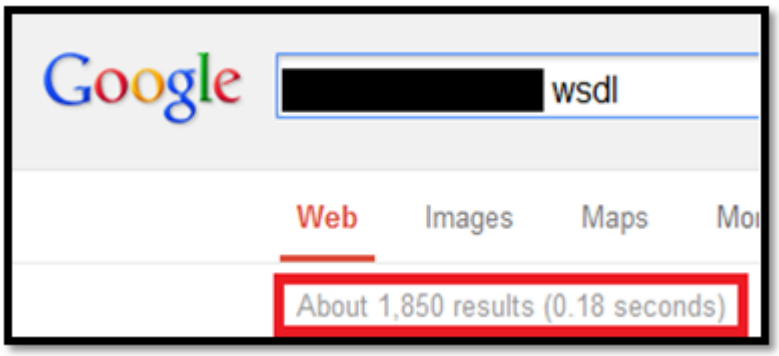


Fig: 10 (Search Result)

As shown in Fig 10, Google will provide you the link of the WSDL file associated with that particular web application. You can use your own dorks or there are dorks available on internet to search for different web services which you can apply also.
Now you have the WSDL file, what is next? As in any kind of penetration testing we need some tools, here also we will use some tools to test for web services. I will cover tools used in web services testing in the installment of this article.

Conclusion:

The sudden increase in the use of web services makes it an important attack vector and the lack of importance it is given makes it more vulnerable. Organizations, developers and testers need to give web services equivalent importance as web applications.

Chapter 2

We discussed how the sudden increase in the use of web services makes it an important attack vector. Also, we covered different components of web services, different elements of WSDL, their uses, where to start, and how to perform penetration testing.

In this article we will be focusing more on automated tools available for web service penetration testing.

Tools

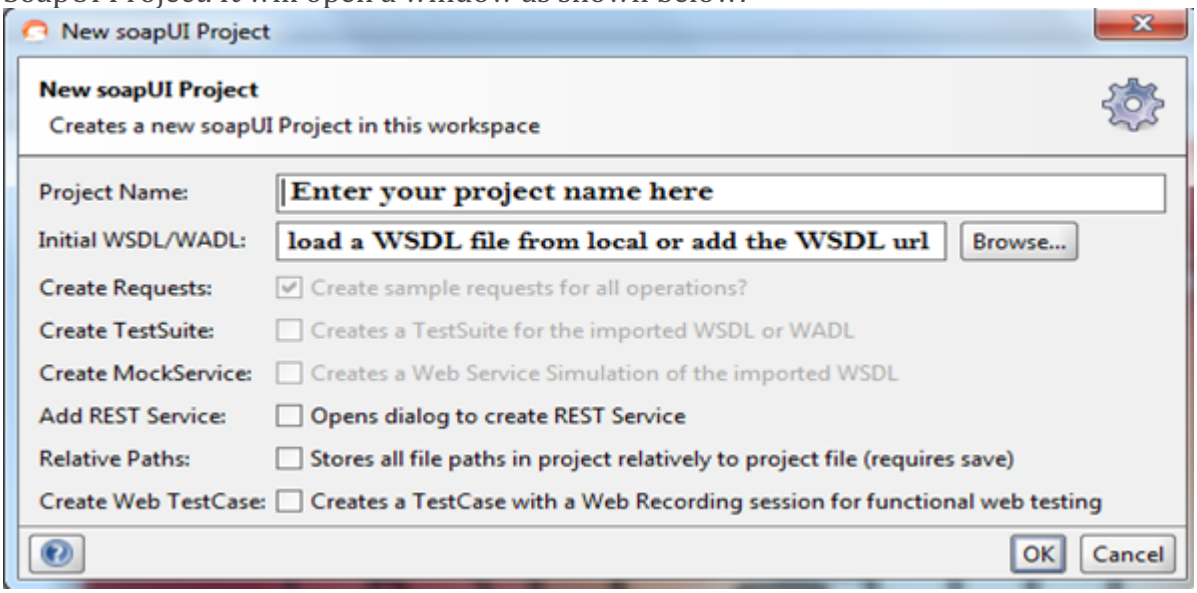
Tools play a very important role in any type of penetration test. But unfortunately, the availability of tools to test web services is limited, compared to web applications. The majority of web service testing tools are built for quality assurance and not for security testing. The approach used to conduct web service testing are mostly from developer's perspective, and precautions are taken such as XML firewalls, to reduce the risk false positives of web service-based attacks.

Due to that, a pen tester has to face a lot of problems while conducting web service penetration testing. But there are still certain tools which help to automate web service penetration testing, and we will go through each of them in this article.

The first tool we're going to use specializes in web services. As I think most of you now have guessed, it's SoapUI by SMARTBEAR (http://www.soapui.org).
SoapUI is the only popular tool available to test for soap vulnerabilities. But to automate the test, we need to use SoapUI Pro. SoapUI comes in two versions. The first is SoapUI (open source), the second is SoapUI Pro (the commercial version). There is a huge difference between these two tools. The main advantage of SoapUI Pro over SoapUI is that it's able to automate a security test. SMARTBEAR also provides a fourteen day free evaluation of SoapUI Pro.

SoapUI Pro

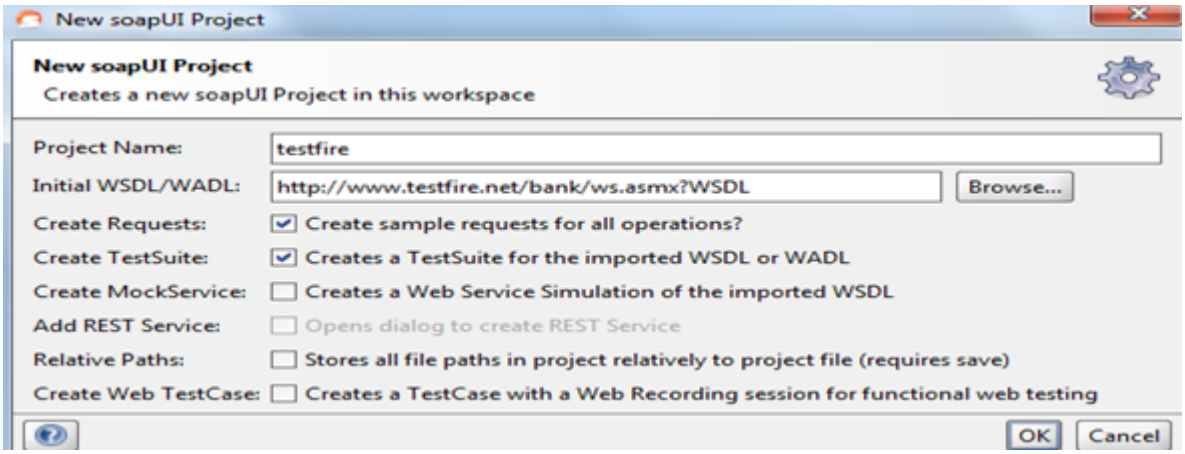
We will test the web services behind http://www.testfire.net/bank/ws.asmx?WSDL by using SoapUI Pro. To automate a test, first we need to open our SoapUI Pro tool. Then click on Files, New SoapUI Project. It will open a window as shown below.



Img1: New SoapUI Project window

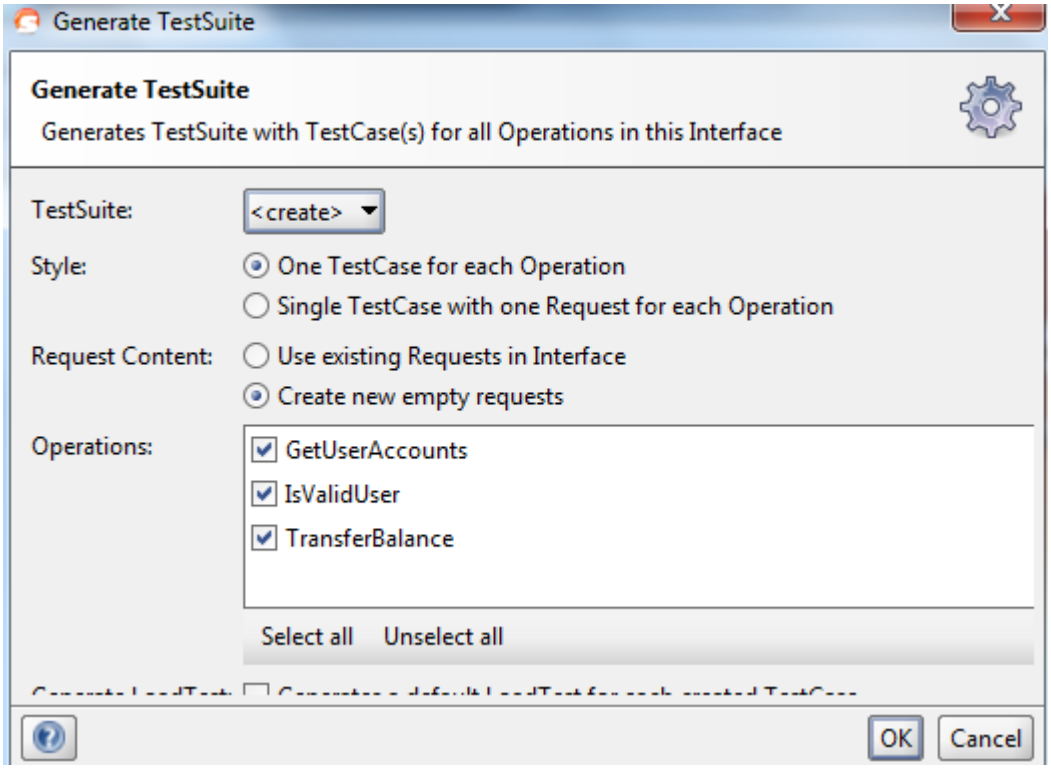
In this case, we're testing the web services of testfire. I will use testfire as the project name, and I will use a WSDL URL for testing. As you can see, the Create Requests option is enabled by default under New SoapUI project. That allows SoapUI Pro to extract all the functions and their requests individually from WSDL.

We can start a test with the default settings, but as we're focusing on automated testing, its better to enable Create TestSuite.



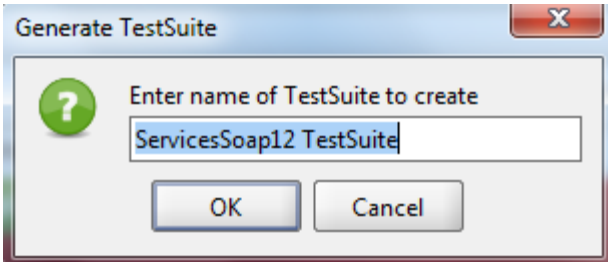
Img2: New soapUI project window with selected options

After that, click on OK to load all the definitions from WSDL. After loading the definitions, a new Generate TestSuite pop-up window will appear. That's because we've enabled the Create TestSuite option.



Img3: Generate TestSuite Window

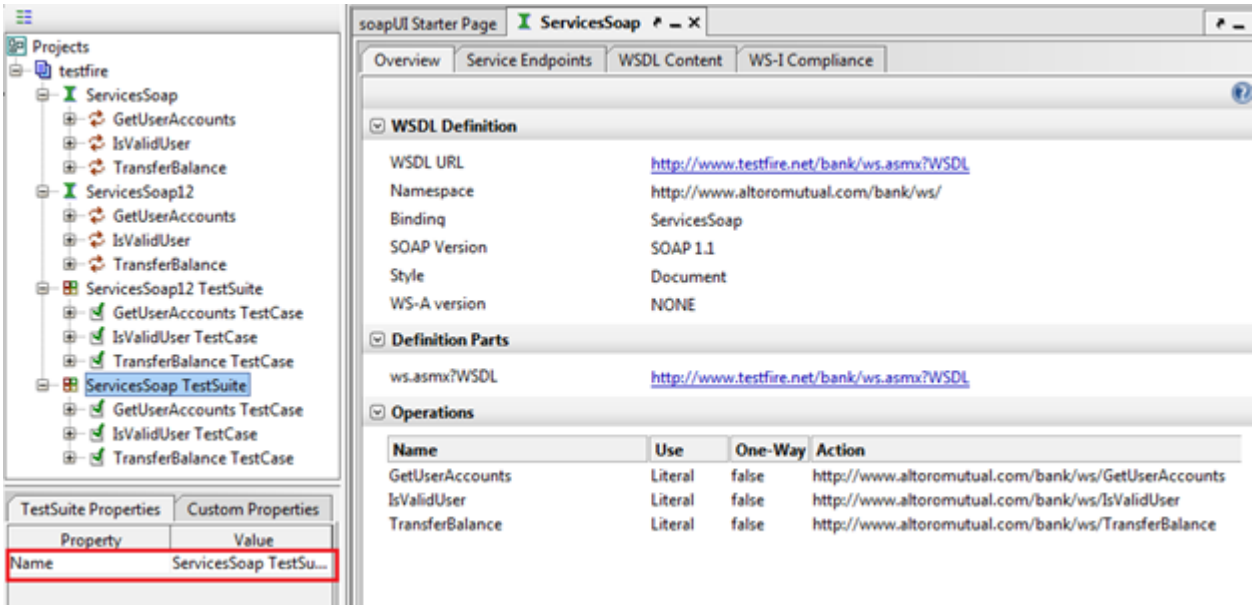
If you want to play with the options, do so. But, let's leave it just like that and click on OK. When you will click on OK, you'll be prompted by another window to enter name of the test suite.



Img4: Enter name for TestSuitewindowE

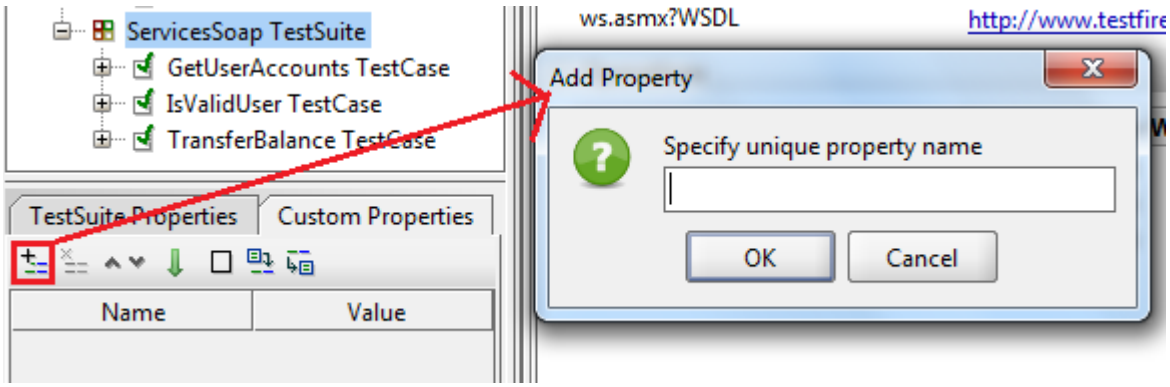
Give it a name, and click on OK. These two Generate TestSuite steps will continue according to the number of services present in WSDL. In our case, two services are present in WSDL. The first is "Services soap." The second is "Services soap12." After completing the process, you're ready to start security testing.

SoapUI Pro also shows you your test suite properties.



Img5: SoapUI Pro window

It also allows you to add a property in the property list, as shown below.

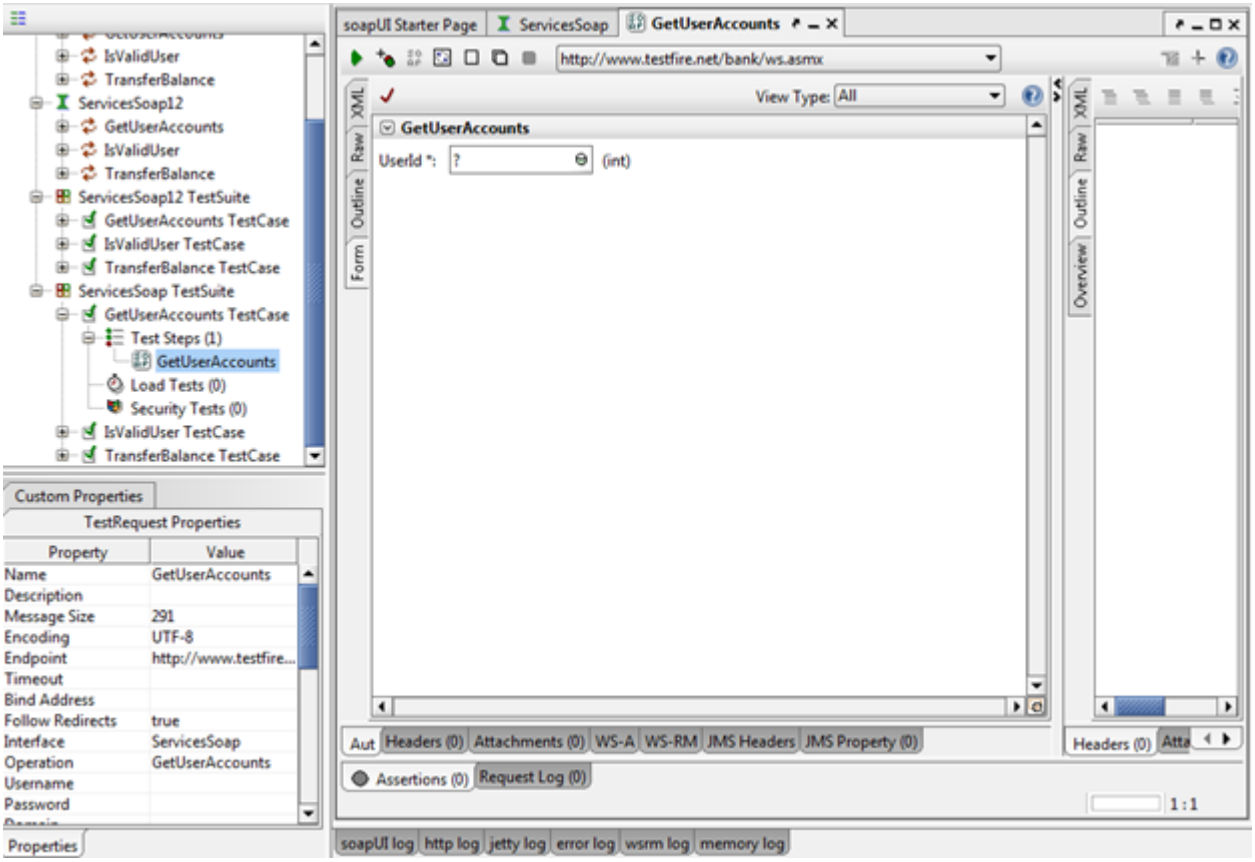


Img6: Add property window

SoapUI Pro allows us to see properties in each level, whether it's for test suite or any operation test case, or request level.

Now, before we automate the security test, we must understand the request we are going to use for this. Let's say we'll test the ServiceSoapTestsuite service and in there we are interested testing GetUserAccountsTestCase. Click on GetUserAccountsTestCase. You will find test steps, load tests, and security tests.

Click on test steps to find the request used. Then, click on that request to open it in the request editor.



Img7: Form View of request editor

As you can see above, by clicking on the GetUserAccounts request, it opens in a request editor. By default it opens in form view. SoapUI Pro allows us to see a request in four different views; XML, raw, outline and form.

We can change the view any time, by clicking on any of the four tabs present in the top left corner of the request editor. Along with that, SoapUI Pro also shows the request property in the bottom-left corner of the window.

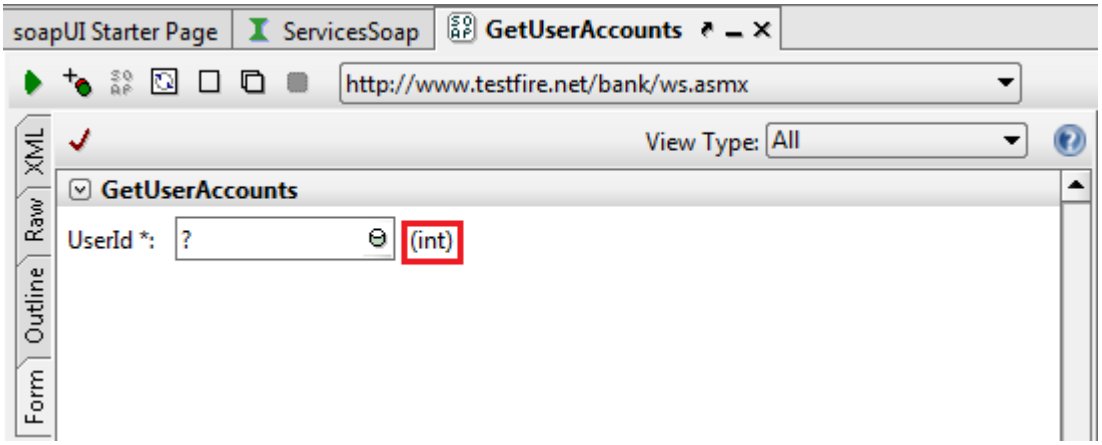
You can use any view you want for testing, but for better understanding, we will go for the xml view. By clicking on the XML view, you will get the XML of the GetUserAccounts request.

Img8: Xml view of request editor window

So, as we discussed in our previous article in the "SimpleObject Access Protocol (SOAP)" section, the entire SOAP message is packed in a SOAP envelope which contains a SOAP header and a SOAP body. The most important thing, from a security tester's point of view, is the value of the "UserId" parameter.

So to test the request, we must provide a value with the required data type in place of the "?" symbol. Generally, we must fuzz this parameter with different types of values of the required data type, to check the result. Also, we can look at other data type values to check the proper implementation of input validation. That's usually done in manual testing. We'll focus on that in the next part of the article.

It's simple to find the required data type of a parameter. You can find required data type information from the form view.



Img9: Form view

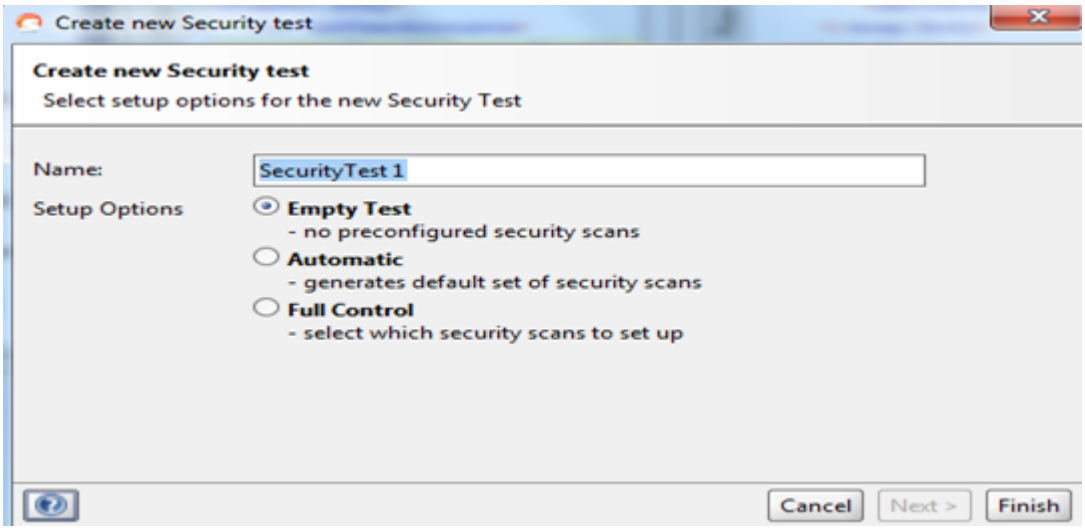
Enter any integer value in the XML view to replace the "?" symbol. Click on the green arrow mark on the top left corner of the request editor window to submit a request to the specified endpoint URL. You'll find a response in the next window.

Img10: Xml view

The XML response in the response window doesn't contain an error message. That means we executed the GetUserAccounts request properly.

Automation of a Security Test

To automate a security test, first we need to create a new security test. To create a new security test, right click on security test, under the Services Soap Testsuite. Click on new security test and a new window will open.



Img11: Create new Security test window

The new window comes with a name option. Type in any name you want. Under this setup, the three options are Empty Test (add a test with no preconfigurations), automatic (generates default set of security scans) and Full control (to customize your test options.)

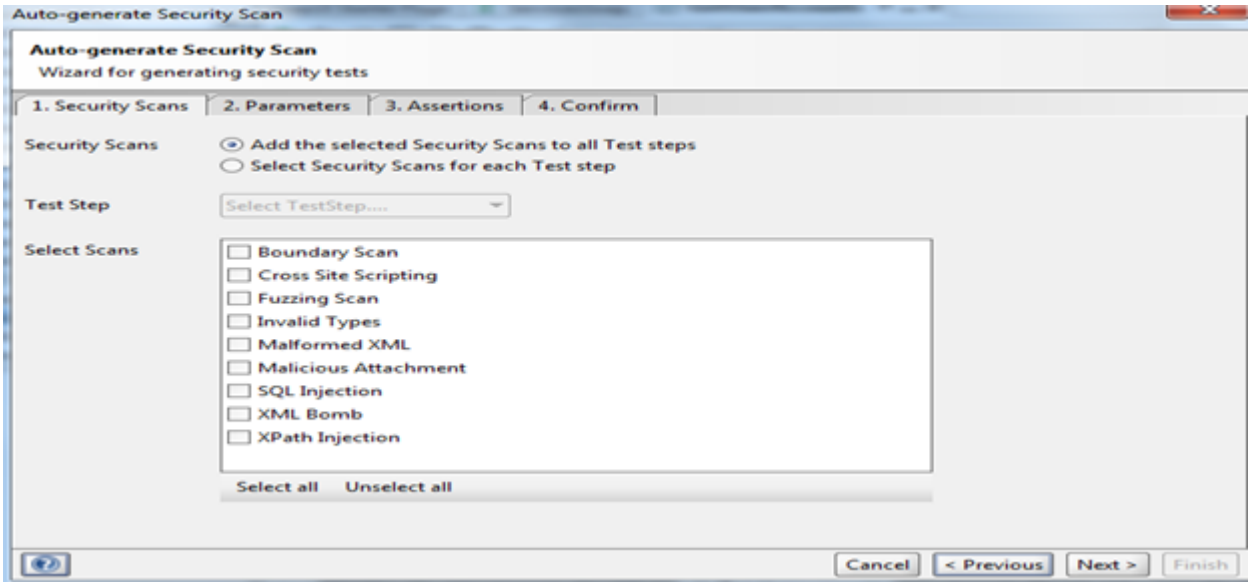
As we want to automate the security test, we can choose any option between automatic and full control. When choosing the automatic option, it will test for each and every vulnerability present in its checklist. Its checklist contains nine types of security scans.

1. Boundary scan
2. Cross site scripting
3. Fuzzing scan
4. Invalid Types
5. Malformed XML

6. Malicious Attachment
7. SQL Injection
8. XML Bomb
9. Xpath Injection

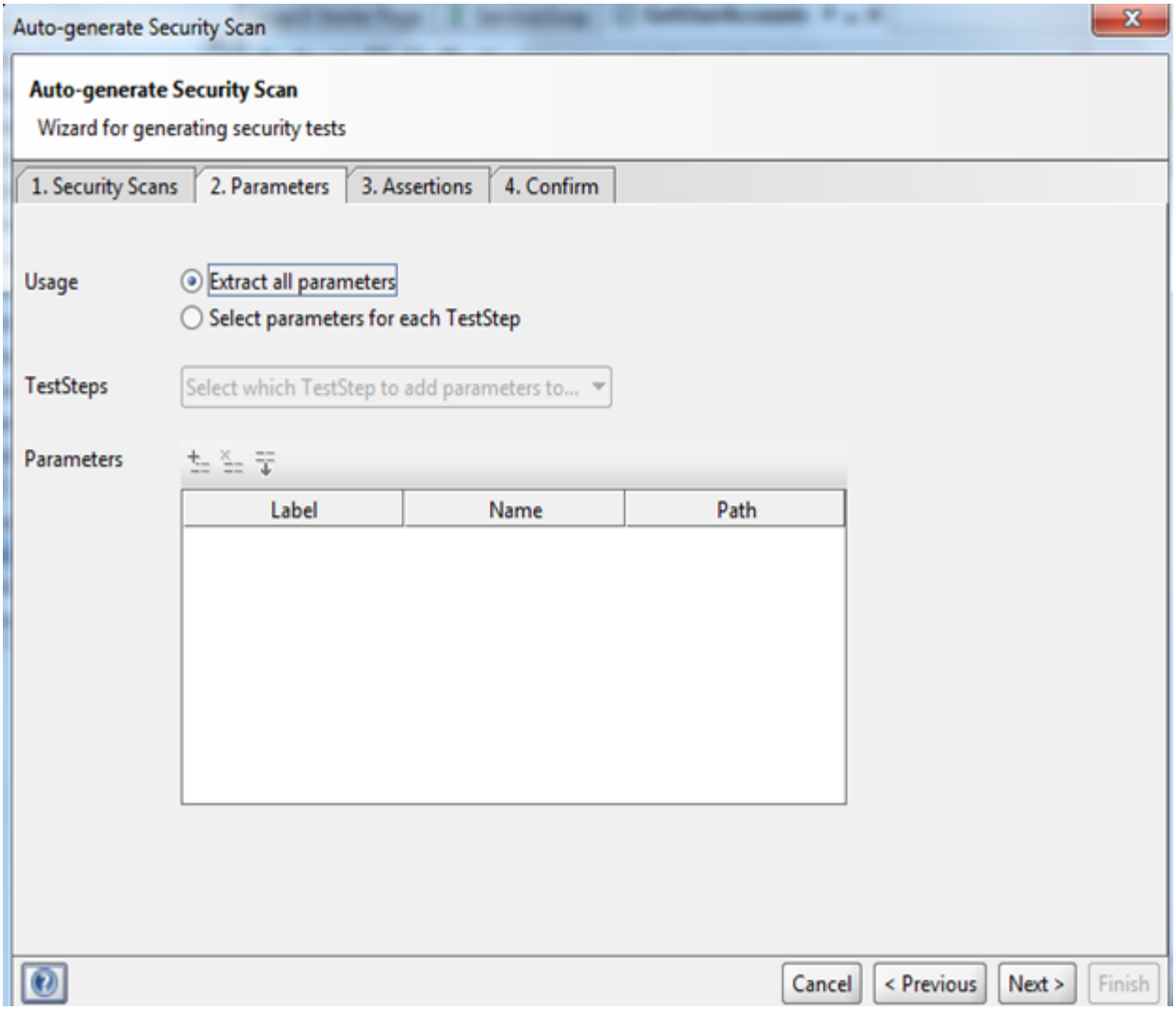
Usually, if you're doing black box testing, and you don't know the function of a web services request, then it's better to choose the automatic option and run all the scans blindly. But if you're doing grey box testing, or you have an understanding of the function used, then it's better to opt for full control to customize your test, which will save a lot of time.

Here, I'll use the full control option to demonstrate how to choose options and why you should. Click on the full control option, and click on next. A new window will open.



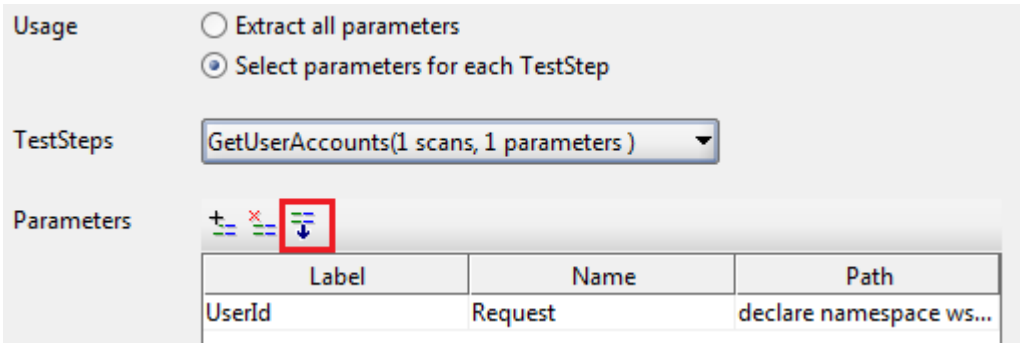
Img12: Security scans tab

You'll see two options. Choose the first one to select the same scan, or the second for different security scans for each test step. Use the default for the first option. In select scans, you can select the scans you think the request might vulnerable to. You can select all of them, but it'll be very time consuming. So select only SQL Injection, and click on next. You'll go to the parameters tab.



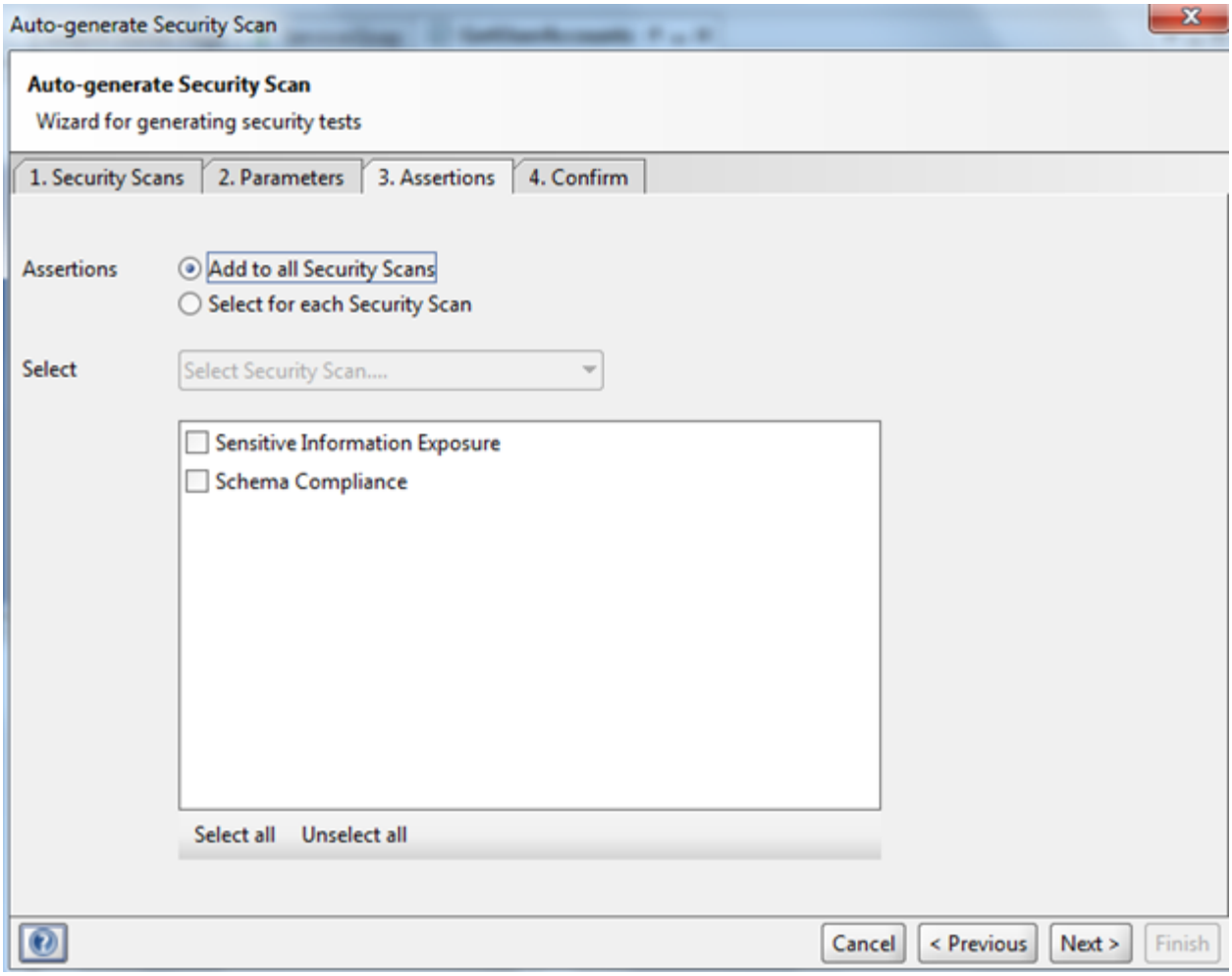
Img13: Parameters tab

By default, SoapUI Pro will extract all the parameters. You can extract parameters by selecting the second option and choosing them from the TestSteps dropdown window.



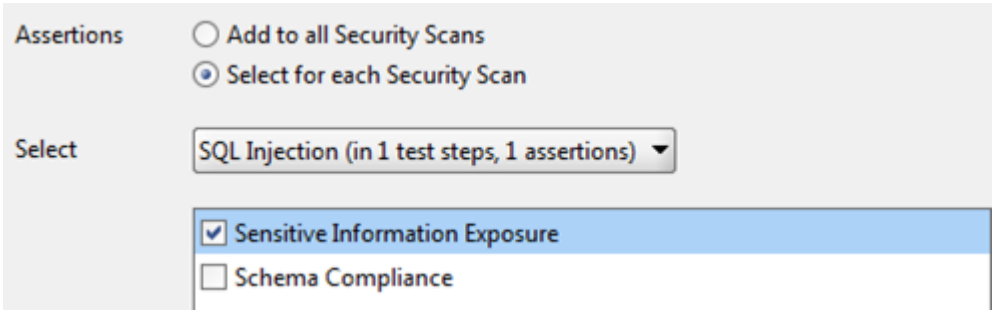
Img14: Parameters options

SoapUI Pro also allows you to add any parameter, and remove any parameter from the list. Click on next to open the Assertions tab.



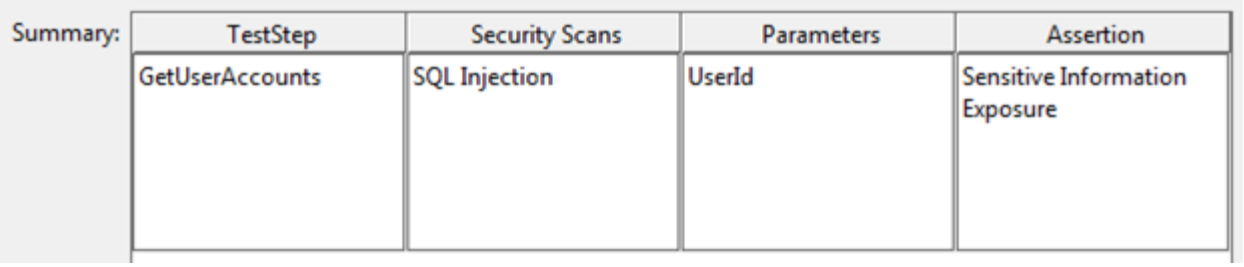
Img15: Assertions tab

That tab is most useful in grey box testing. By checking the sample requests, we can add some data or pattern which is sensitive. If in any test, SoapUI Pro finds the same value in response, it will generate an error to avoid the disclosure of sensitive information. In our case, choose select for each Security Scan option, and select Sensitive Information exposure.



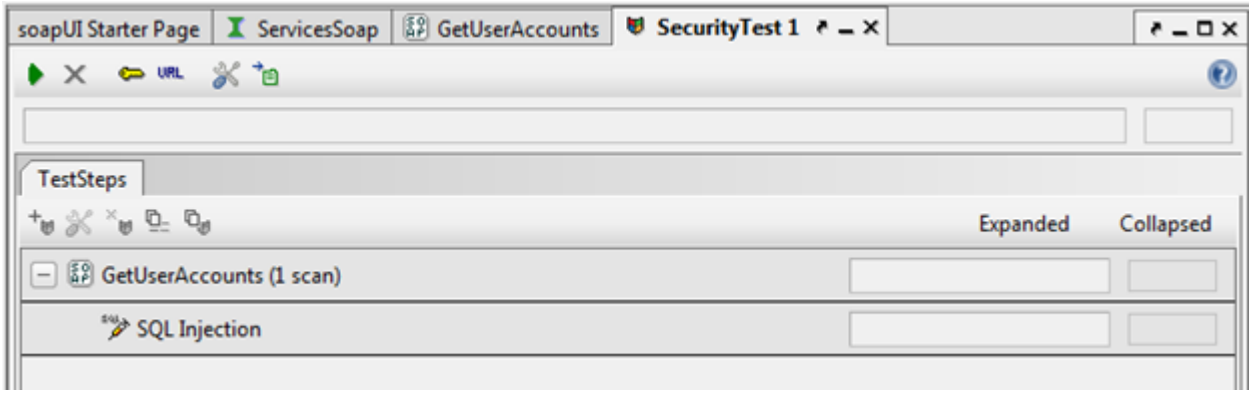
Img16: options

Click on next to move to the confirmation tab, where all the options we selected are shown in a table.



Img17: Summary

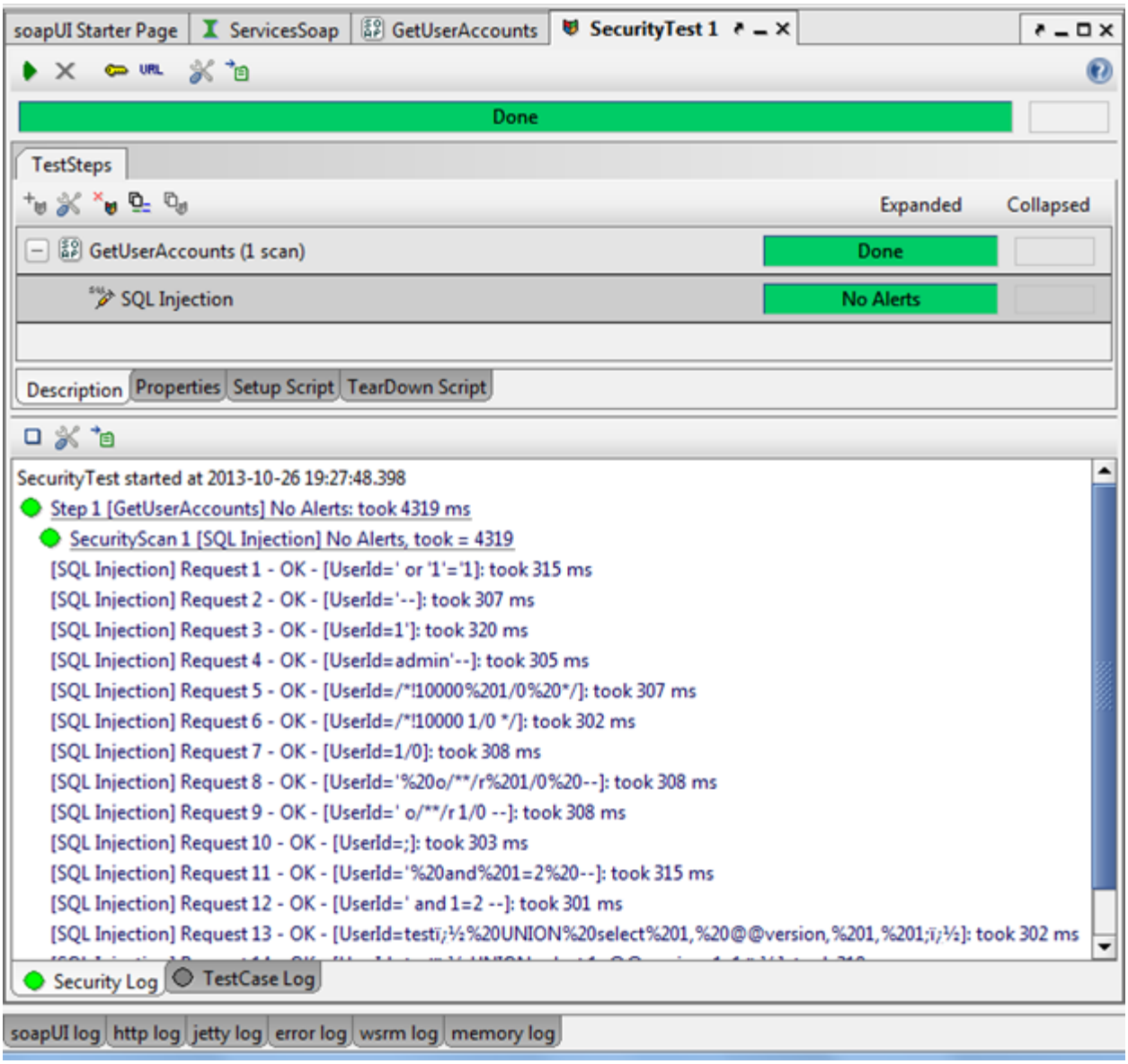
Then, click on finish to open a new window to start an automated test.



Img18: Security test window

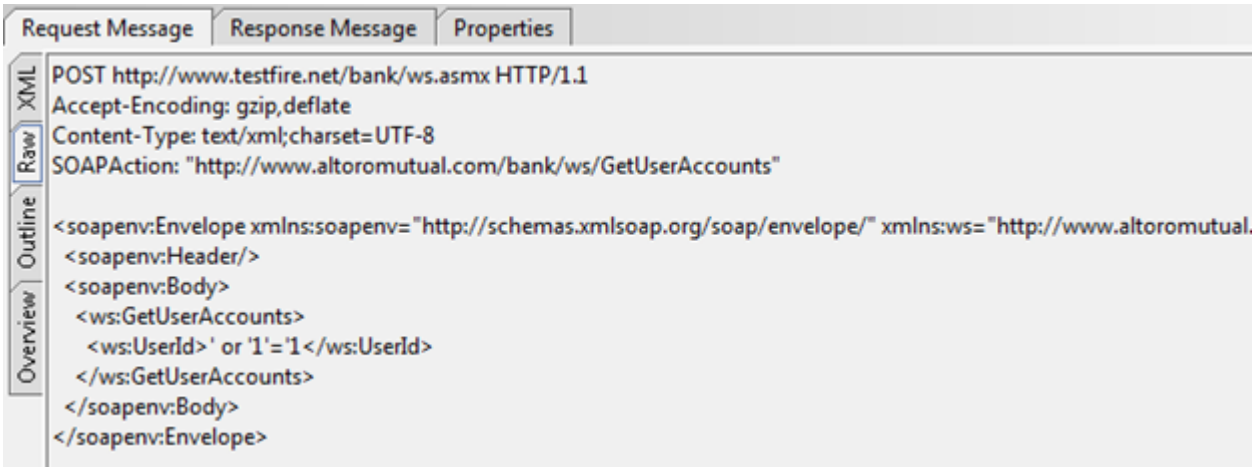
We've selected only the SQL Injection test for the GetUserDetails request as it appears in the security test screen. Click on the green arrow button in the top left corner to start an automated test.

After completing the test, the window will look like the image below.

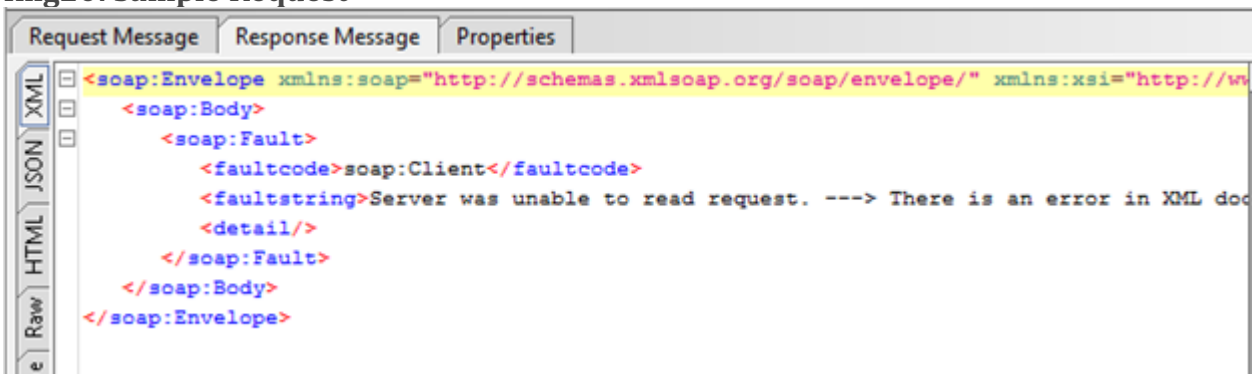


Img19: Security test results

As you can see from the above image, there's no alert triggered, as there's no sensitive information in any of the responses which come while using different payloads. Click any request to view the request and its response.

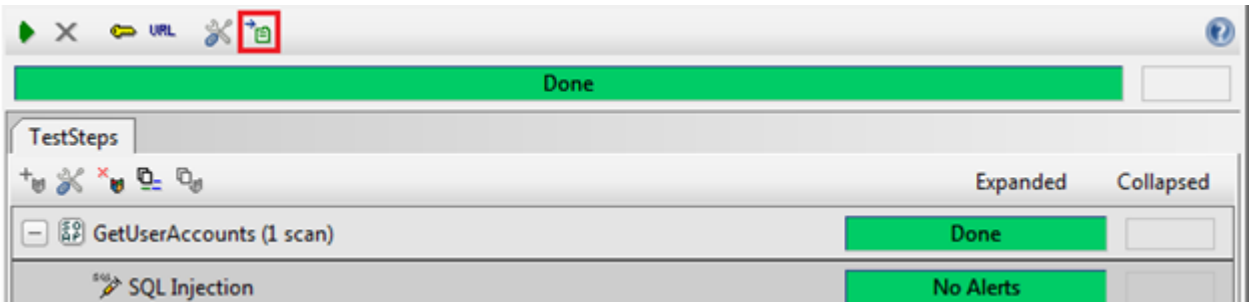


Img20: Sample Request



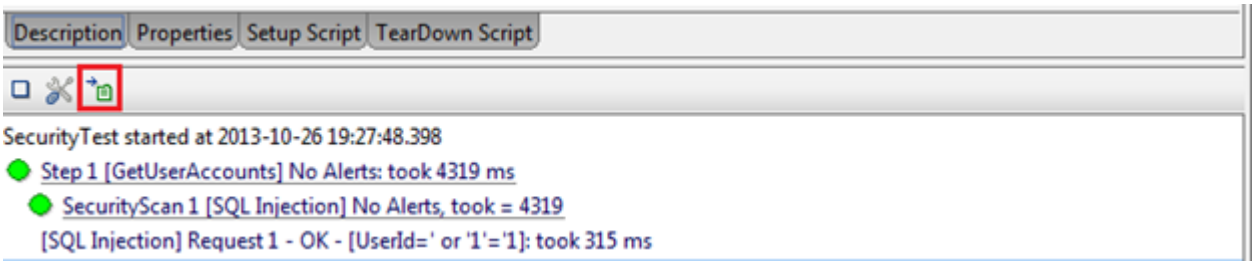
Img21: Sample Response

When the test is complete, you need a report. SoapUI provides a feature to create a report of the test, by clicking on the create a report for this item button, which is present at the top.



Img22: Report Generation option

Sometimes, you need the log to store as a proof of the test. SoapUI Pro also provides the option to store the log, by clicking on exports the log to a file button.



Img23: Save log option

Conclusion:

This is how SoapUI Pro allows us to automate a security test for different requests. There are other tools available in the market to provide automated web service testing. But SoapUI Pro is a specialized web service tool which can be used for functional testing, load testing, security testing and other different types of testing. This tool plays a vital role in testing web services.

Chapter 3

In the previous article, we discussed the importance of tools in penetration testing, how automation helps in reducing time and effort, and how to automate web services penetration testing using soapUI Pro.
In this article, we will be focusing on what other options are available to automate web services penetration testing.

Feasibility
To perform web services penetration testing, soapUI Pro is one of the best options, but in certain conditions you might search for other options: For example, you are not into regular web services penetration testing. or your budget is very low for a penetration testing that consists of web application penetration testing along with web services penetration testing, or you don't have much experience in performing web services penetration testing.

For these conditions, you need something that comes as a package. A tool for web application penetration testing as well as web services penetration testing. A tool where you just click next, next, next, and it will provide you the result of web services penetration testing. A tool where you can throw the WSDL and get the result. You might choose one of these very popular web application penetrations testing tools, IBM AppScan or HP WebInspect.

AppScan
IBM Security AppScan (http://www-03.ibm.com/software/products/us/en/appscan/) is one of the most popular and widely used automation tools in the arena of web application penetration testing. It allows penetration testers to automate their web application penetration testing to find out the vulnerabilities present in the application. Most penetration testers use it for only web application penetration testing but it can be also used to test web services to identify the vulnerabilities present. Now we will focus on how web services penetration testing is done by IBM Security AppScan.
Testing Web Services Using AppScan
Testing a Web Service using AppScan differs slightly from testing a normal web application because AppScan uses a separate client to explore the web services. That separate client is called the Generic Services Client (GSC).

Generic Service Client (GSC)
It uses the WSDL file of a web service to display the individual methods available in a tree format, and it creates a user-friendly GUI for sending requests to the service. You can use this interface to select methods, one by one, and to input the required values of the parameters. Simultaneously you can also send the request to the server to view the results in the form of the response. These processes are recorded by AppScan and later used to create test cases based on the number of requests made by the GSC for the service.

Configuration
You need to configure AppScan properly with the required options to perform a web services penetration test. As we learned from "Web Services Penetration Testing Parts 1 and 2," we need a WSDL file or URL to perform web services penetration testing properly. We will test the web services of http://www.testfire.net/bank/ws.asmx?WSDL. It's always better to have sample test data (SOAP requests and responses) to test web services properly but, since we are performing black box testing, we will provide the format of data needed to perform the testing.

Open AppScan to start the web services penetration testing. AppScan will start with the window shown in Figure 1.

Figure 1: New Window

This window will show the recent scans and the option to "Create New Scan." Click on that option. The "New Scan" window will open, as shown in Figure 2.

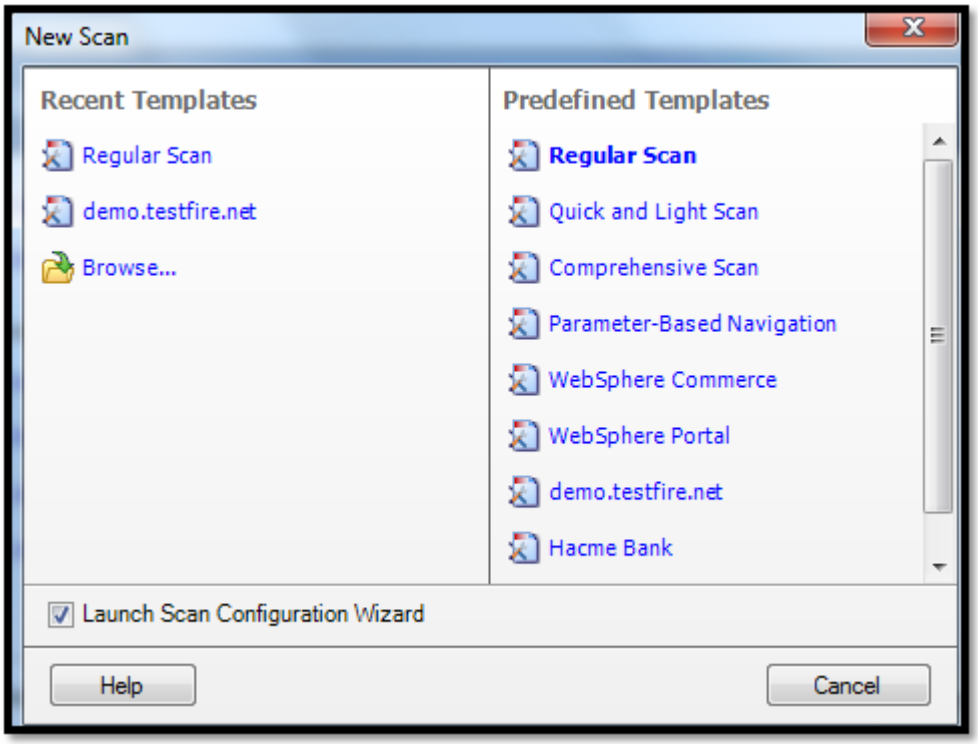


Figure 2: New Scan Window

This “New Scan” window will show the “Recent Templates” used and also option to select one of the “Predefined Templates.” Select “Regular Scan” from the “Predefined Templates.” By clicking on the “Regular Scan” template, the “Scan Configuration Wizard” window will open, as shown in Figure 3.

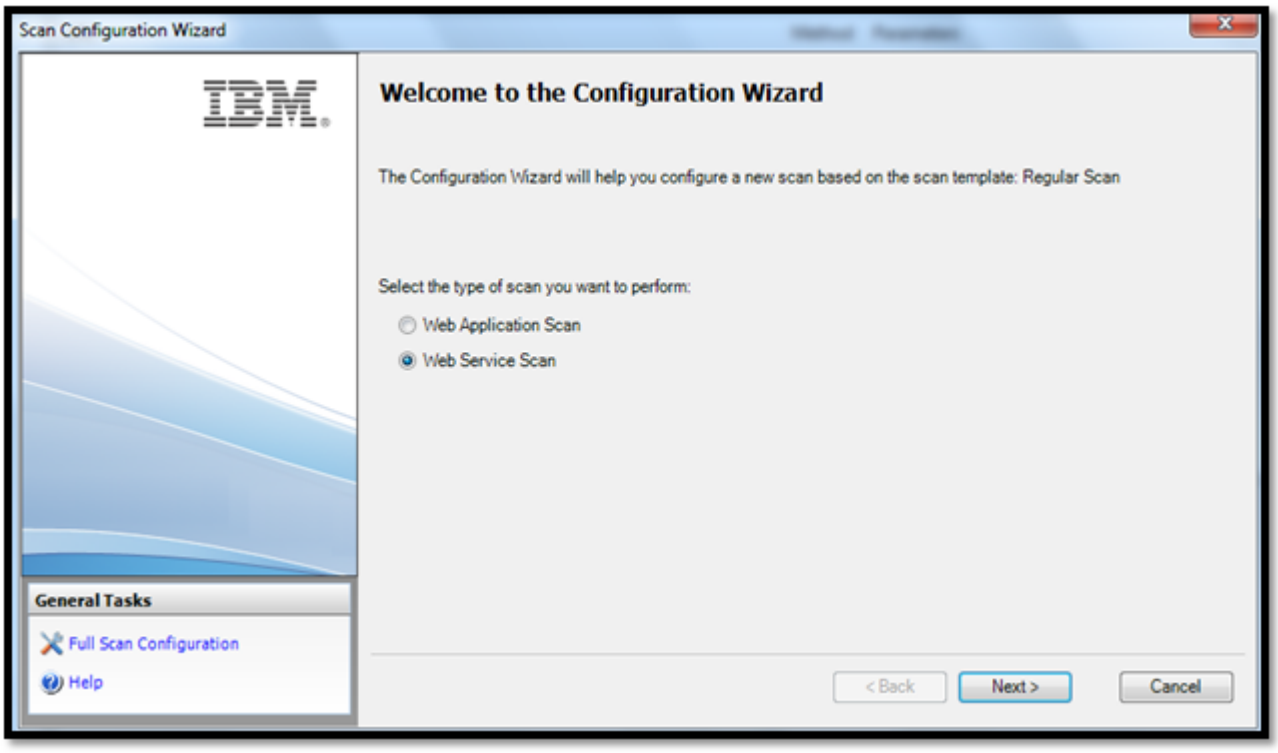


Figure 3: Scan Configuration Window

In the “Scan Configuration Wizard,” select “Web Services Scan” and click on “Next” to open a window where you need to provide the WSDL file or WSDL URL, as shown in Figure 4.

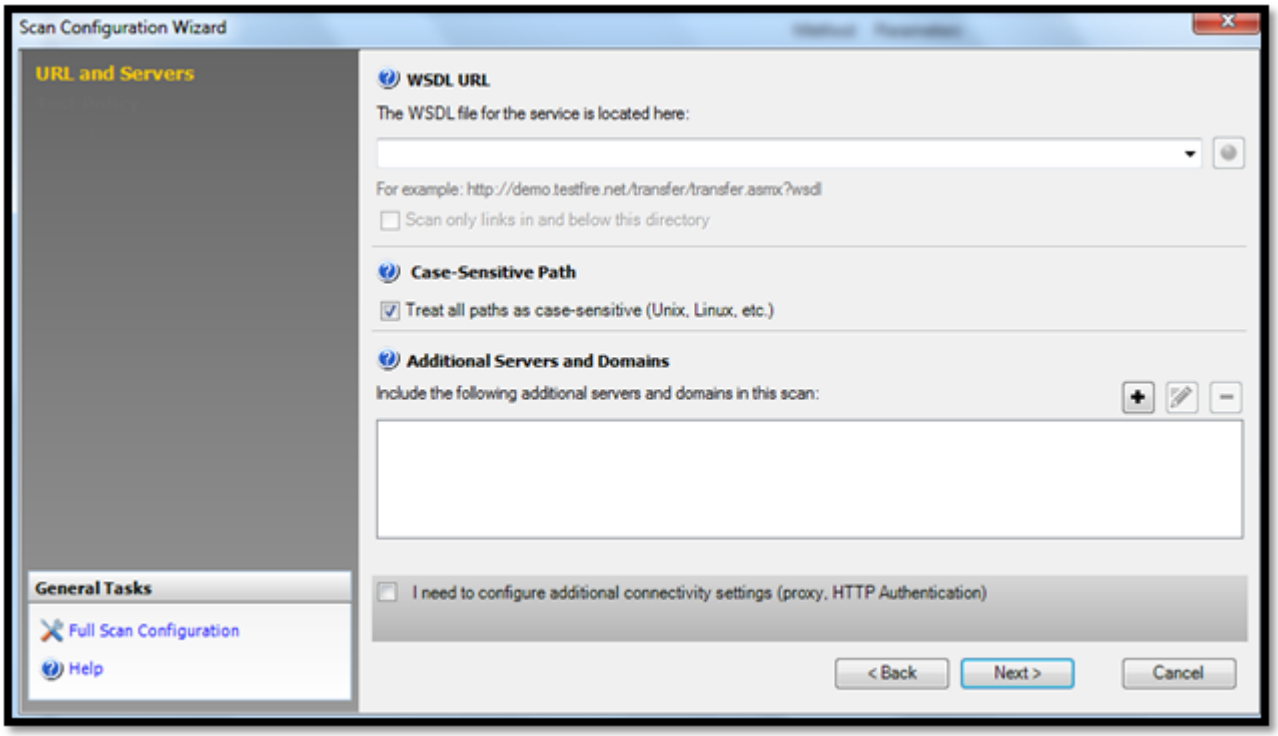


Figure 4: URL and Servers Window

If you need to configure any additional settings for proxy or HTTP authentication, you can configure them here, but to test the web services, I will continue with the default settings, as shown in Figure 5.

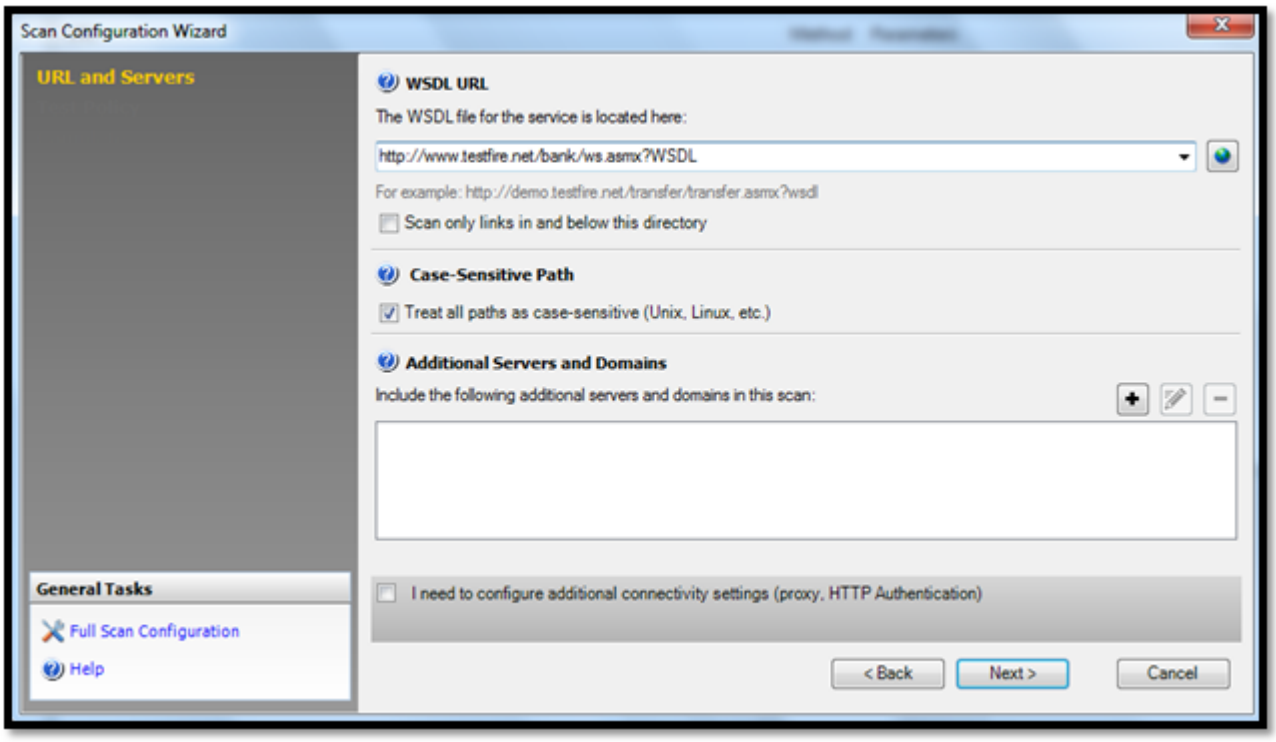


Figure 5: URL and Servers Window

Click on "Next" to open the "Test Policy" window, as shown in Figure 6.

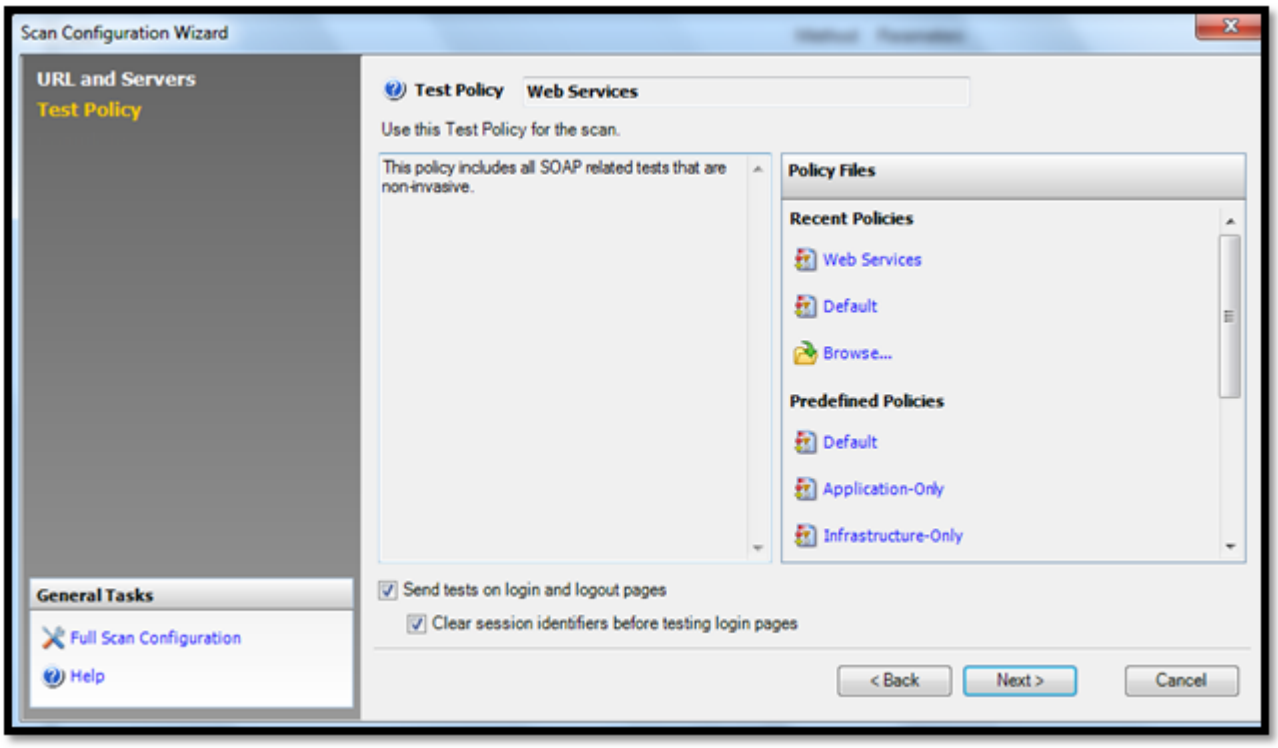


Figure 6: Test Policy Window

Here you will find a predefined policy present to test SOAP-related tests, i.e., "Web Services." Select "Web Services." If you want to check what are the test cases associated with this policy, just click on the "Full Scan Configuration" link, which is in the left bottom corner of this window, under "General Tasks." Clicking on the "Full Scan Configuration" link will open a new "Full Scan Configuration" window, as shown in Figure 7.

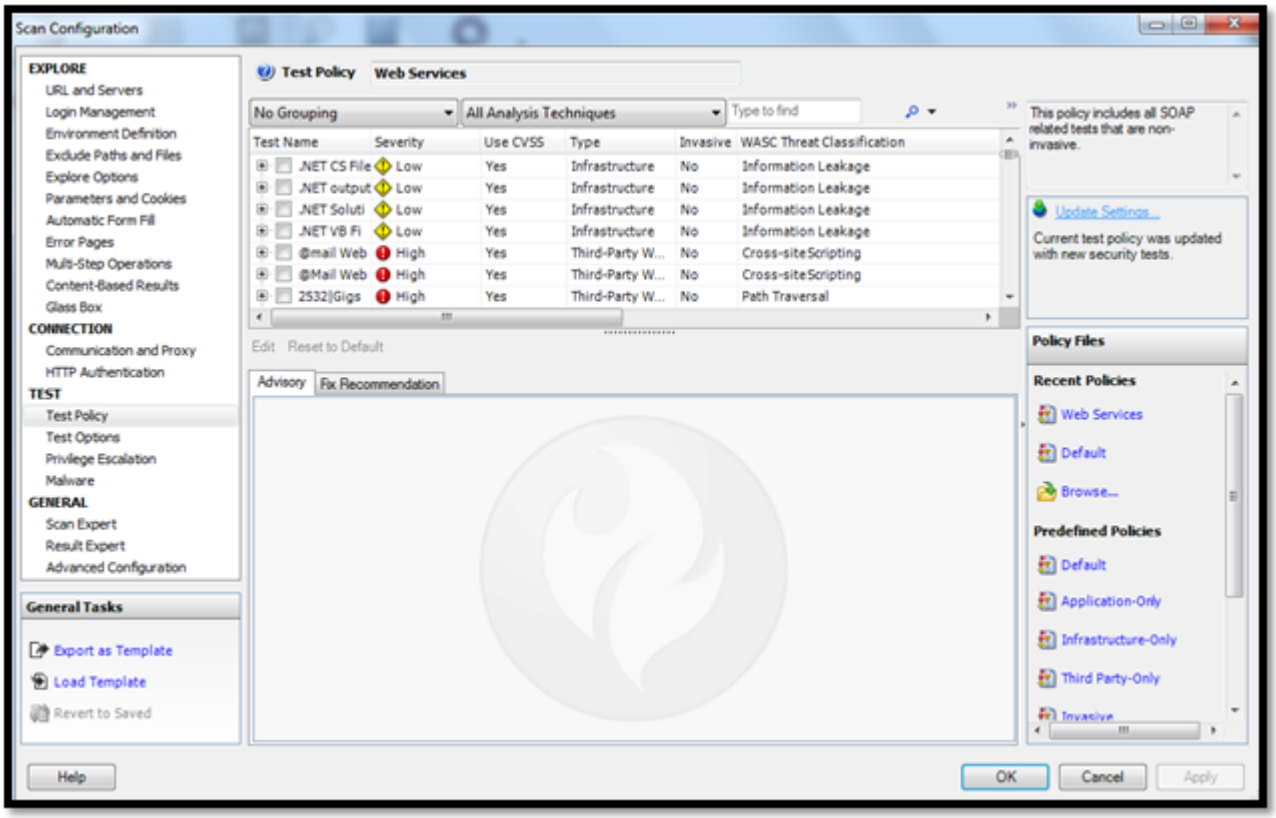


Figure 7: Full Scan Configuration Window

Select the "Test Policy" tab, which is on the left side of the window under "TEST," to view the test cases included in this Web Services policy. Under the "No Grouping" option when you start exploring the test cases, you will see three types of buttons:

1. Disabled
2. Enabled
3. Partially Enabled

Below mentioned are some of the classes of test cases included in this policy.

1. XML External Entities
2. Information Leakage
3. Insufficient Authentication
4. SQL Injection
5. Cross Site Scripting
6. Directory Indexing
7. Abuse of functionality
8. Session Fixation
9. OS Commanding
10. Format String

11. Brute Force
12. Insecure Indexing
13. LDAP Injection
14. Content Spoofing
15. Remote File Inclusion
16. Null Byte Injection
17. SSI Injection
18. Insufficient Session Expiration
19. Insufficient Transport Layer Protection
20. HTTP Response Splitting
21. Path Traversal
22. XPath Injection

After exploring the test cases included, click on "OK" to close the full scan configuration window, then click on "Next" to complete the configuration wizard and it will open a new window, as shown in Figure 9.

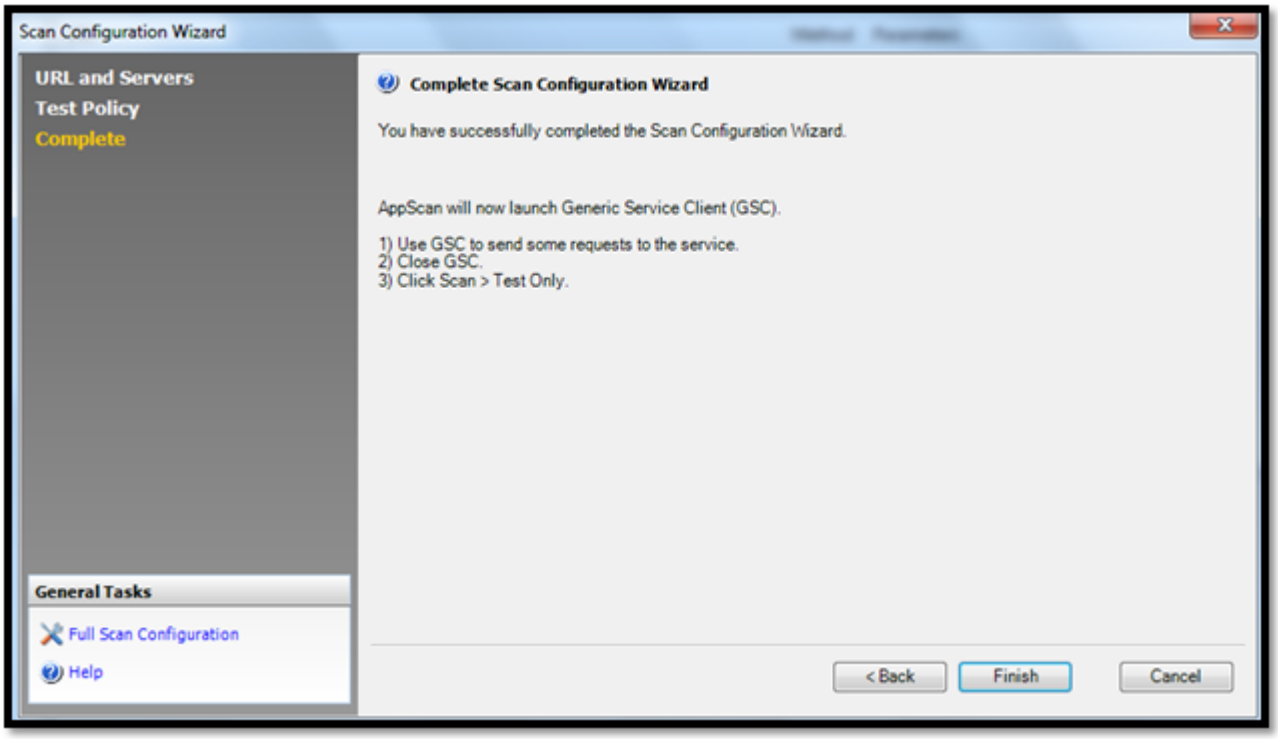


Figure 9: Complete Scan Configuration Window

This window shows that you have successfully completed the scan configuration wizard and also provides information how to start the test by exploring web services methods using GSC. Click on "Finish" to launch GSC, where GSC will import all the methods available in the provided WSDL file as shown in Figure 10.

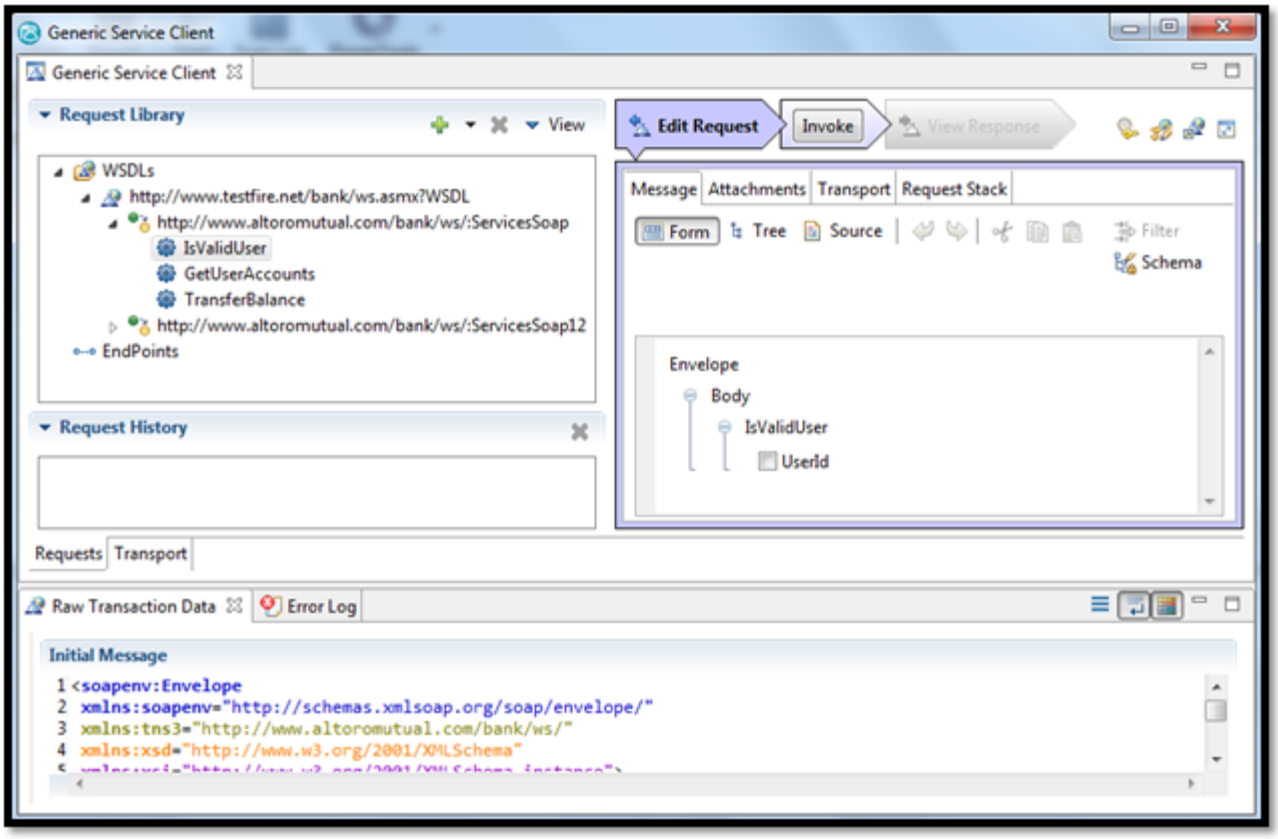


Figure 10: GSC Window

This GSC shows all the imported methods under "Request Library." Now you need to edit each method request and provide a value for the required parameter with the required data type in the edit request option, as shown in Figure 11.

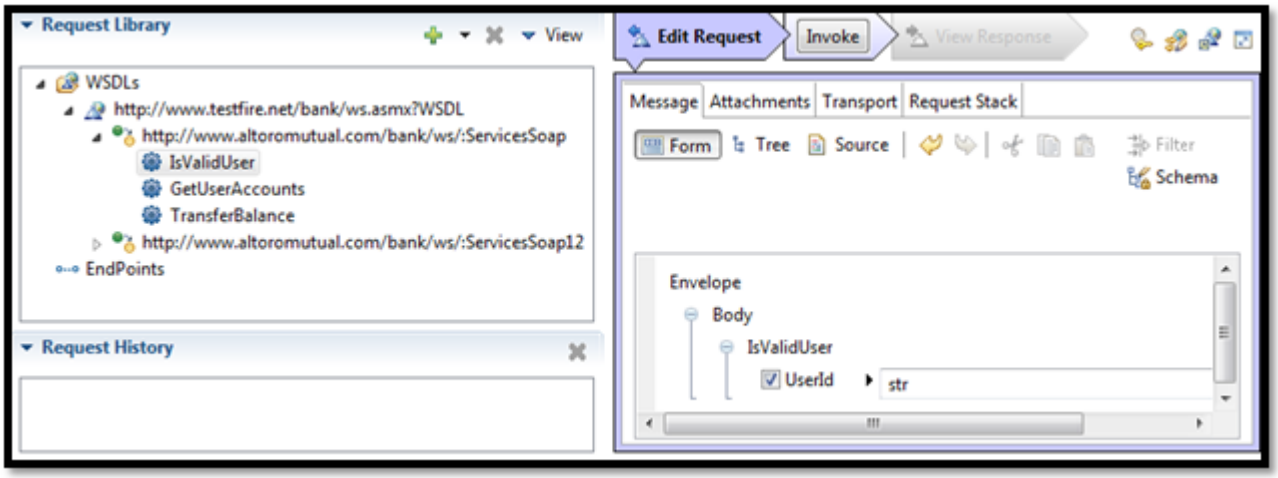


Figure 11: GSC

I selected the "IsValidUSer" method and clicked on the "UserId" parameter. It requires a string datatype value. Now provide a string datatype value and invoke the request. I provided the value 1 and clicked on the "Invoke" button; the response I got is shown in Figure 12.

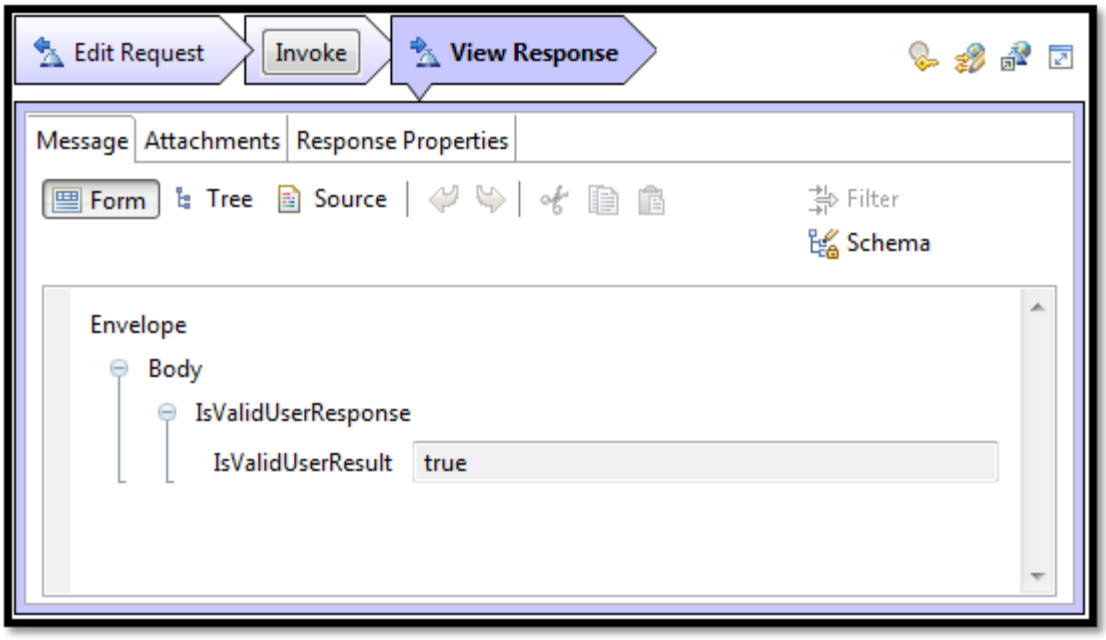


Figure 12: GSC Request Editor

Similarly select all the methods, put the required data type value in the parameters, and invoke the requests one by one. After completion of all the invocation of requests, close the GSC window. Now AppScan will record all the requests and generate the test cases to start web services penetration testing, as shown in Figure 13.

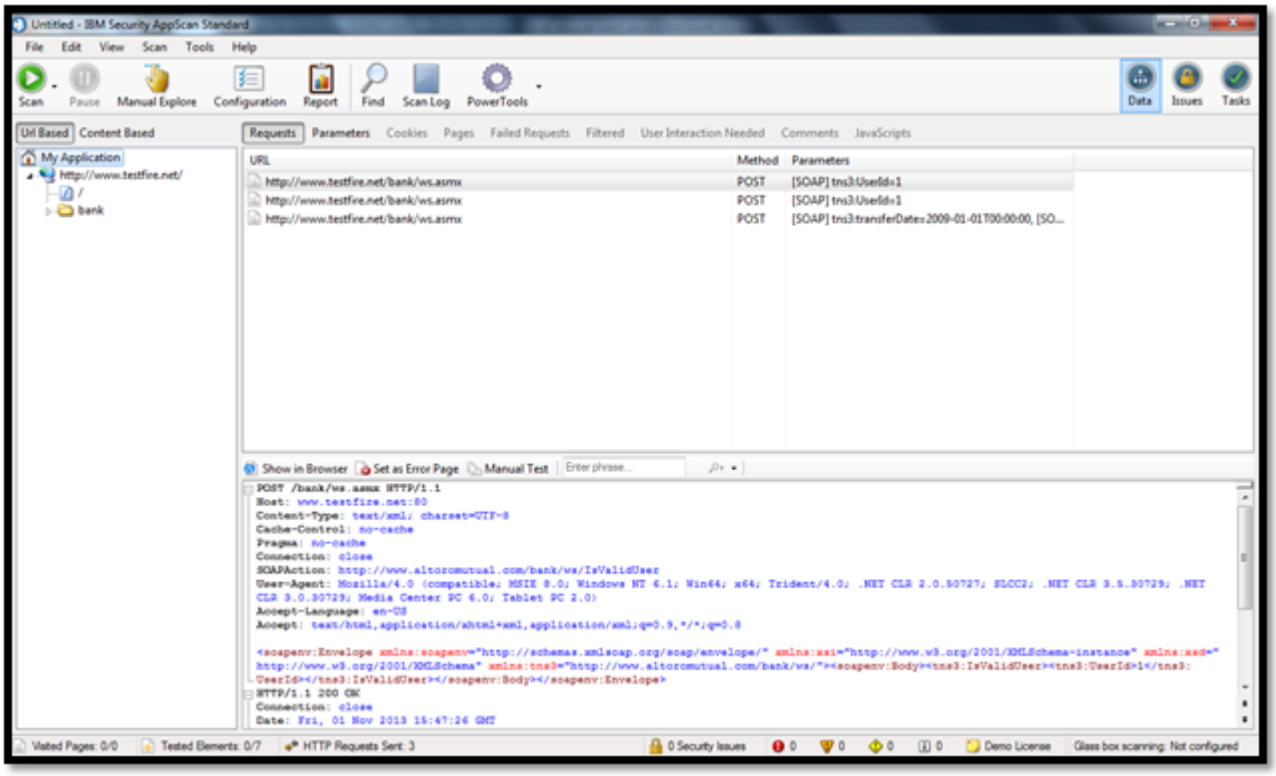


Figure 13: AppScan Test Window

As you can see, AppScan fetched all the requests from GSC request history and is all set to start the test. Just click on the "Test Only" option in the top left corner to start the test. After completion of all test cases, you will get the result in AppScan, as shown in Figure 14.

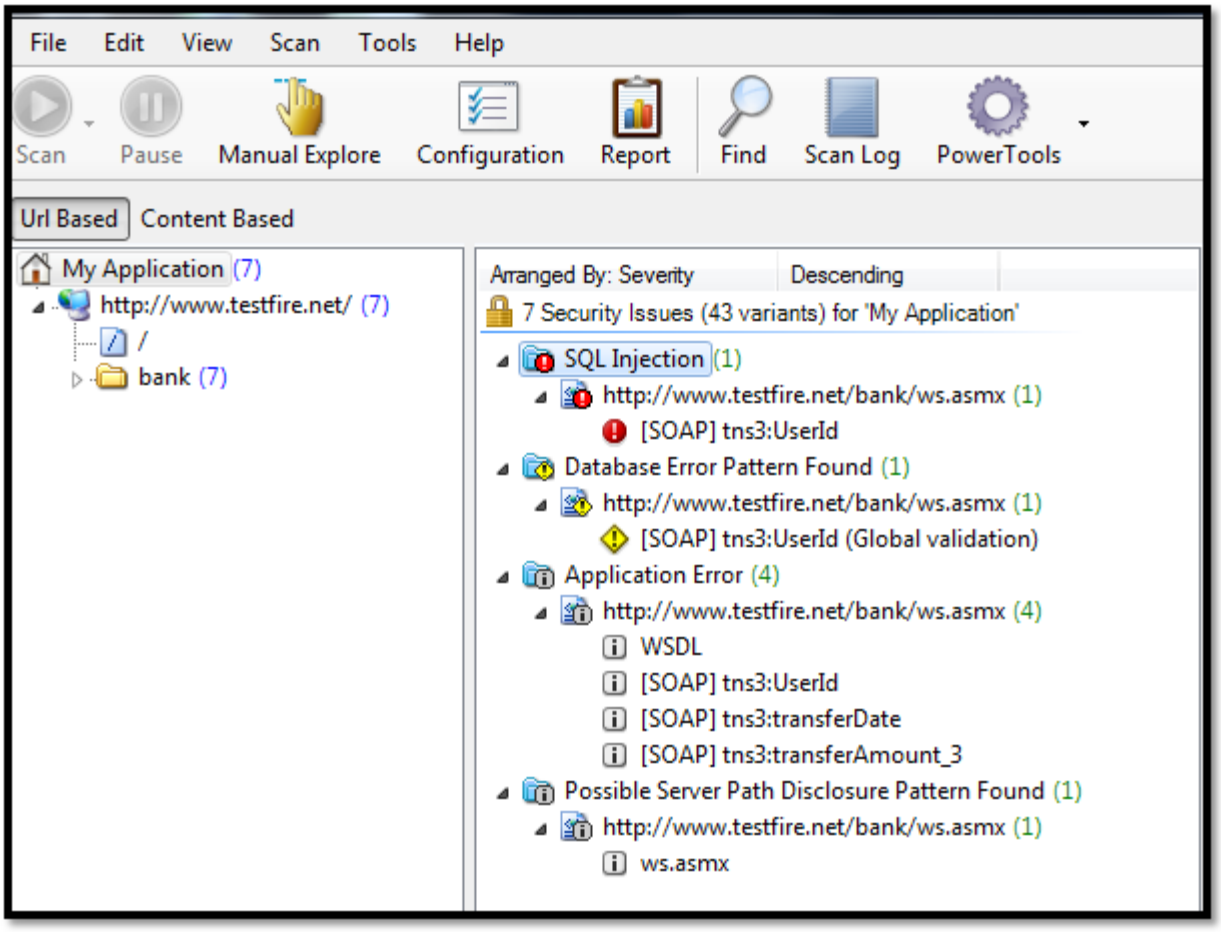


Figure 14: AppScan Result

Here are the results of the web service scan using AppScan. Now you can use AppScan to test any web service to discover the vulnerabilities present. And you need to verify it manually to avoid False-positive.

WebInspect

HP WebInspect
(http://www8.hp.com/in/en/software-solutions/software.html?compURI=1341991)
is another very popular tool for web application penetration testing. It uses real-world hacking techniques and attacks to thoroughly analyze your web applications and web services to identify security vulnerabilities. It contains some features in web services penetration testing that make it one of the popular black box web services penetration testing tools. Now we will focus on how to test web services using HP WebInspect.

Open WebInspect and you will find its start page containing "Recently Opened Scans," "Scans Scheduled for Today," "WebInspect Messages," "What's new in WebInspectxx.x!" (where "xx.x" is the version of WebInspect you are using), along with options to start a new scan, as shown in Figure 15.

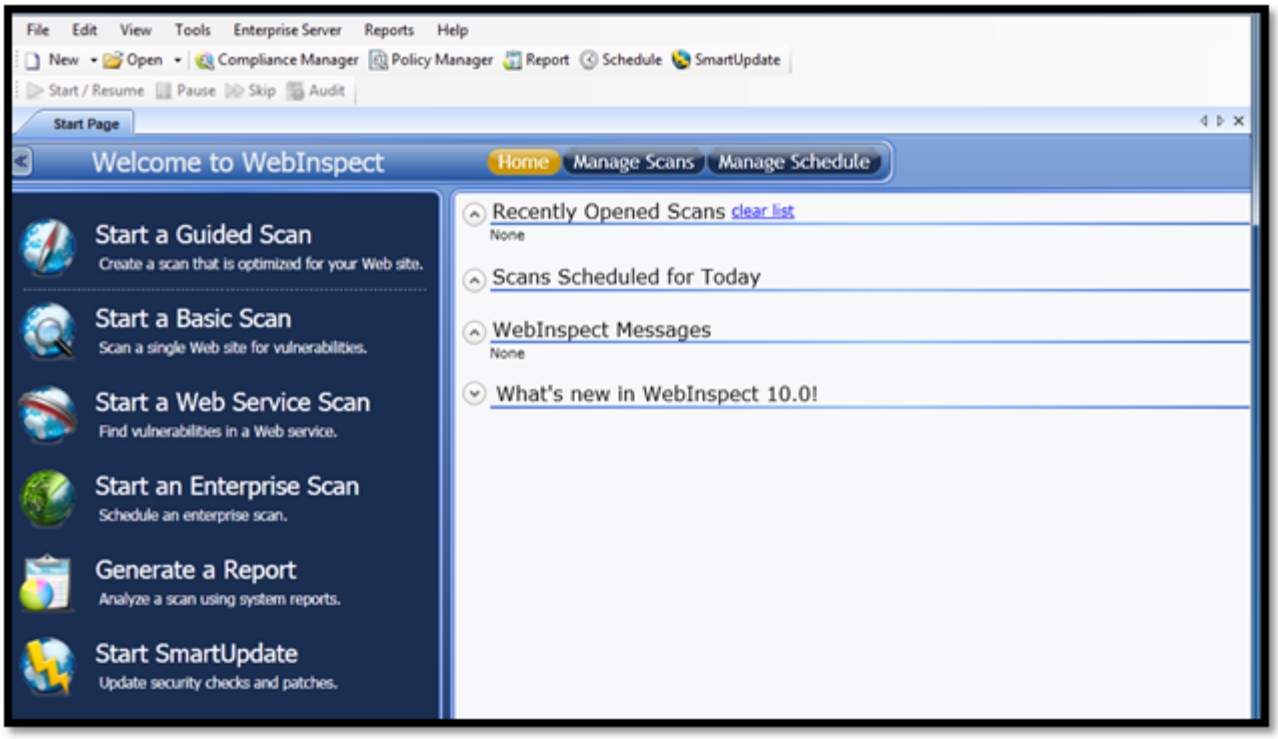


Figure 15: WebInspect Start Page

Click on "Start a Web Service Scan," which will open a "Web Service Scan Wizard," as shown in Figure 16.

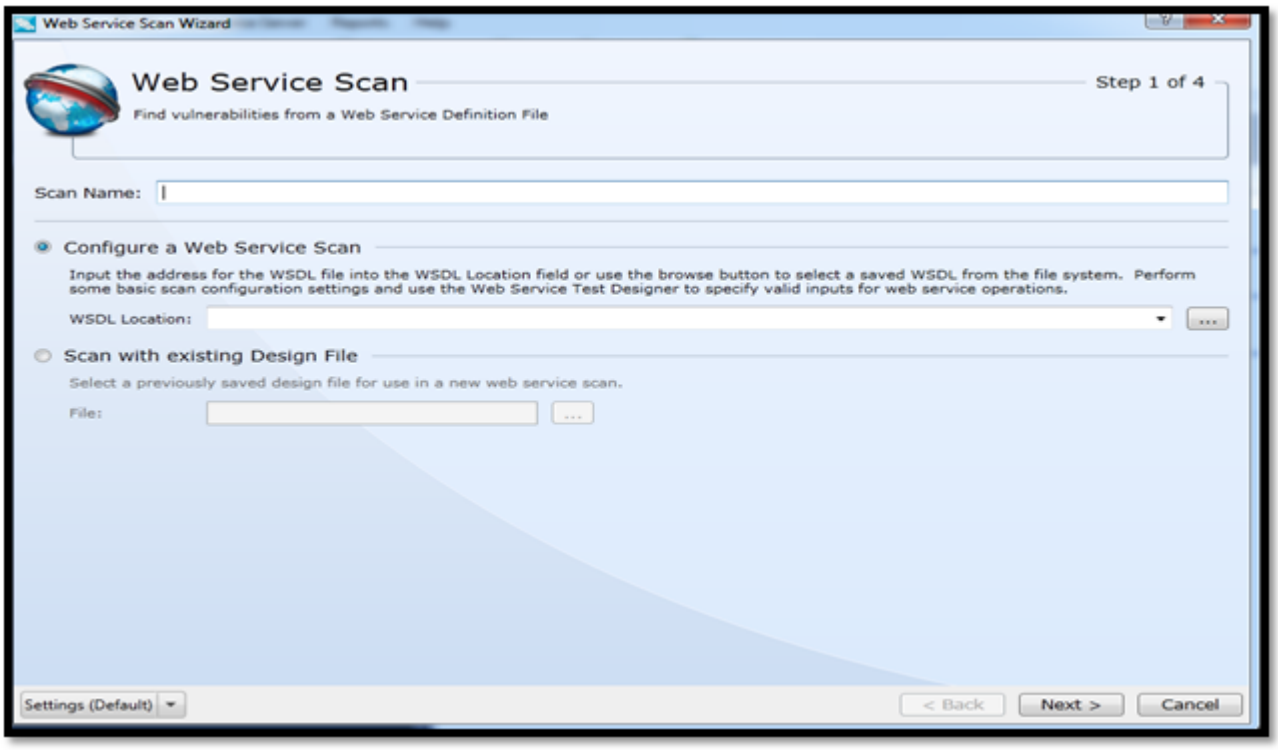


Figure 16: Web Service Scan Window

Select "Configure a Web Service Scan" and in the space for "WSDL Location" insert your WSDL URL. In my case, I am using the same http://www.testfire.net/bank/ws.asmx?WSDL. And in "Scan Name" enter a name. I am using testfire, as shown in Figure 17.

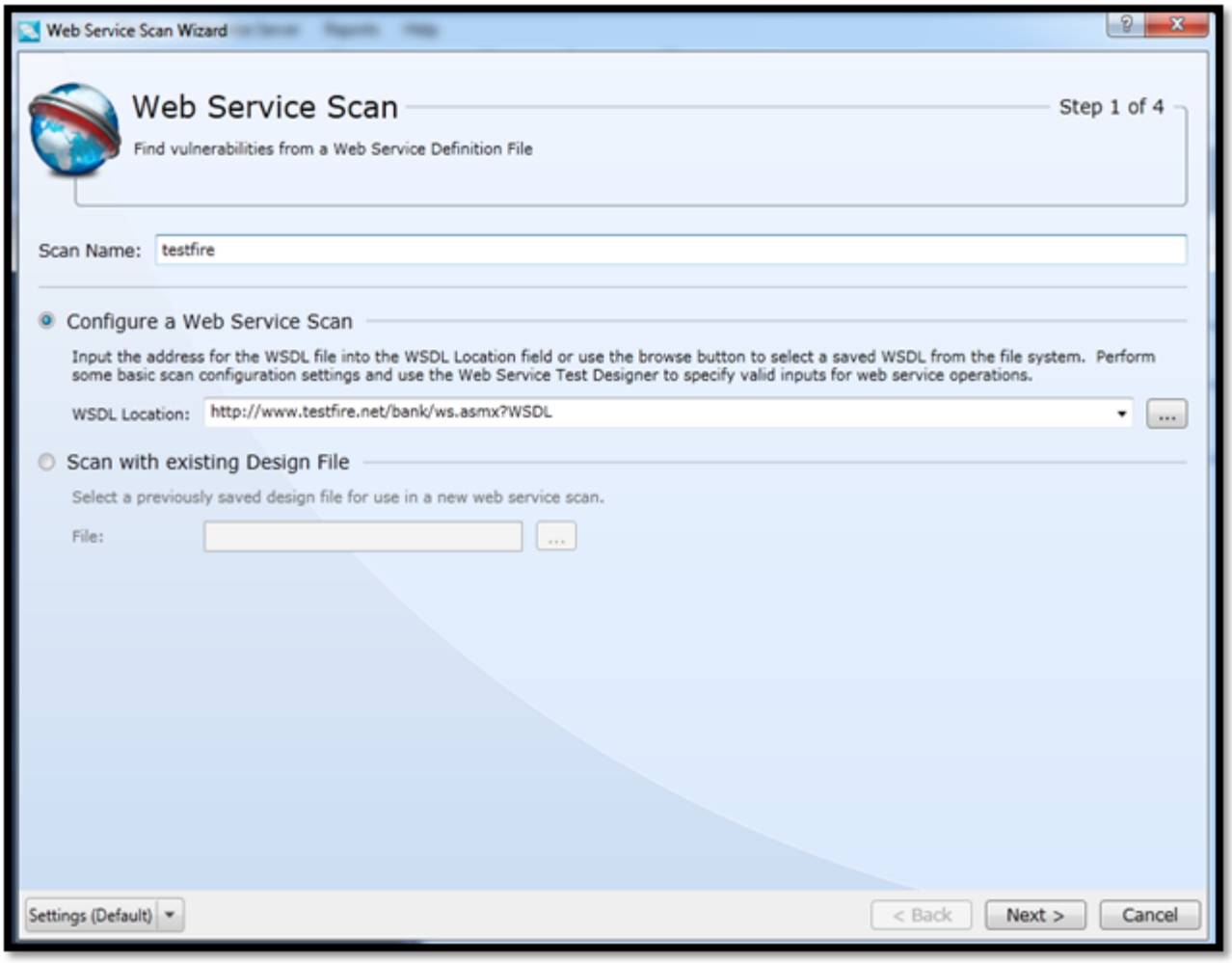


Figure 17: Web Service Scan Window

Click on "Next" to get the "Authentication and Connectivity" window, where you have to provide all the required details, as shown in Figure 18.

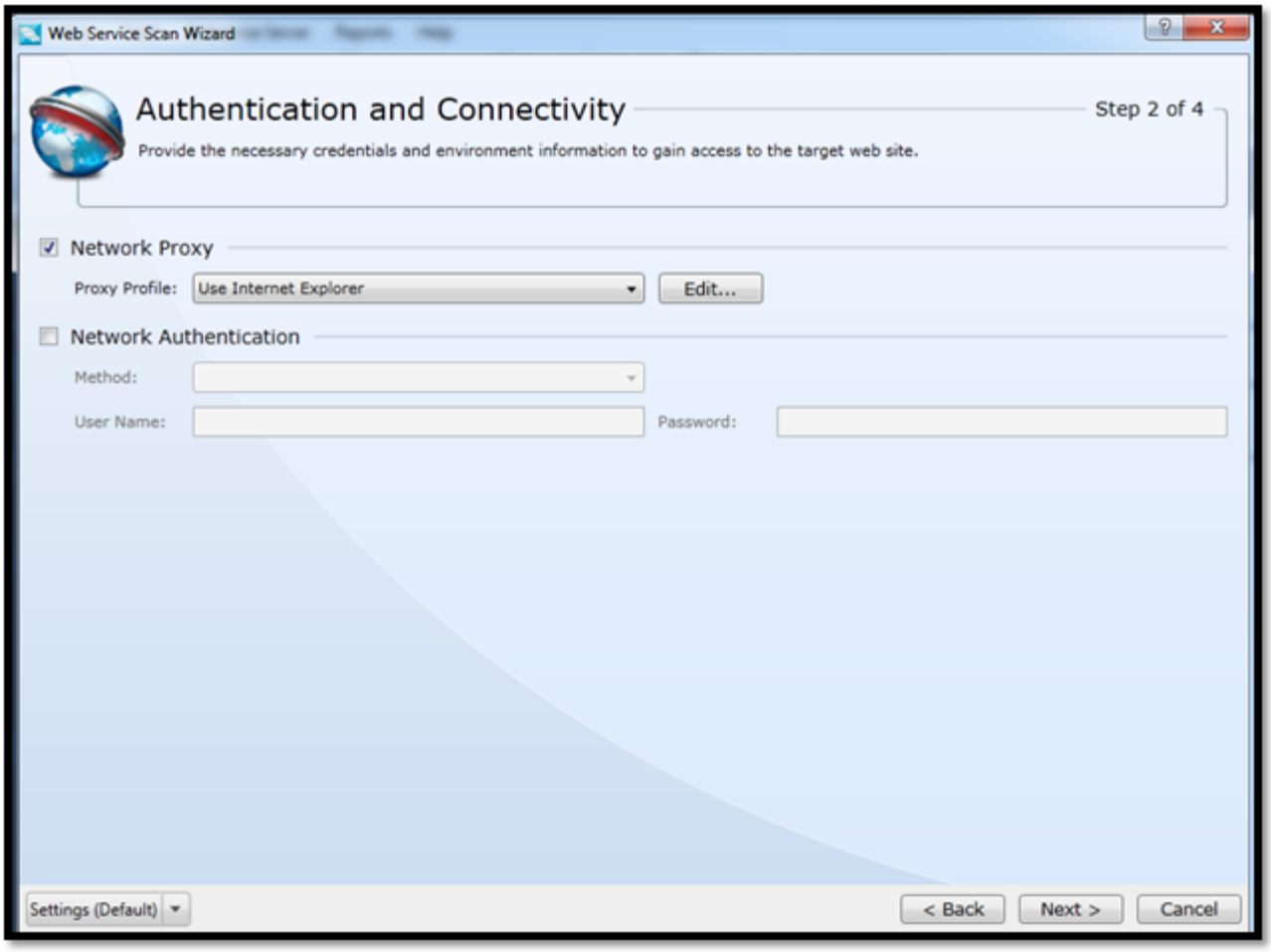


Figure 18: Authentication and Connectivity Window

As in our case we don't need any "Network Proxy" or "Network Authentication," uncheck the "Network Proxy" option, as shown in Figure 19.

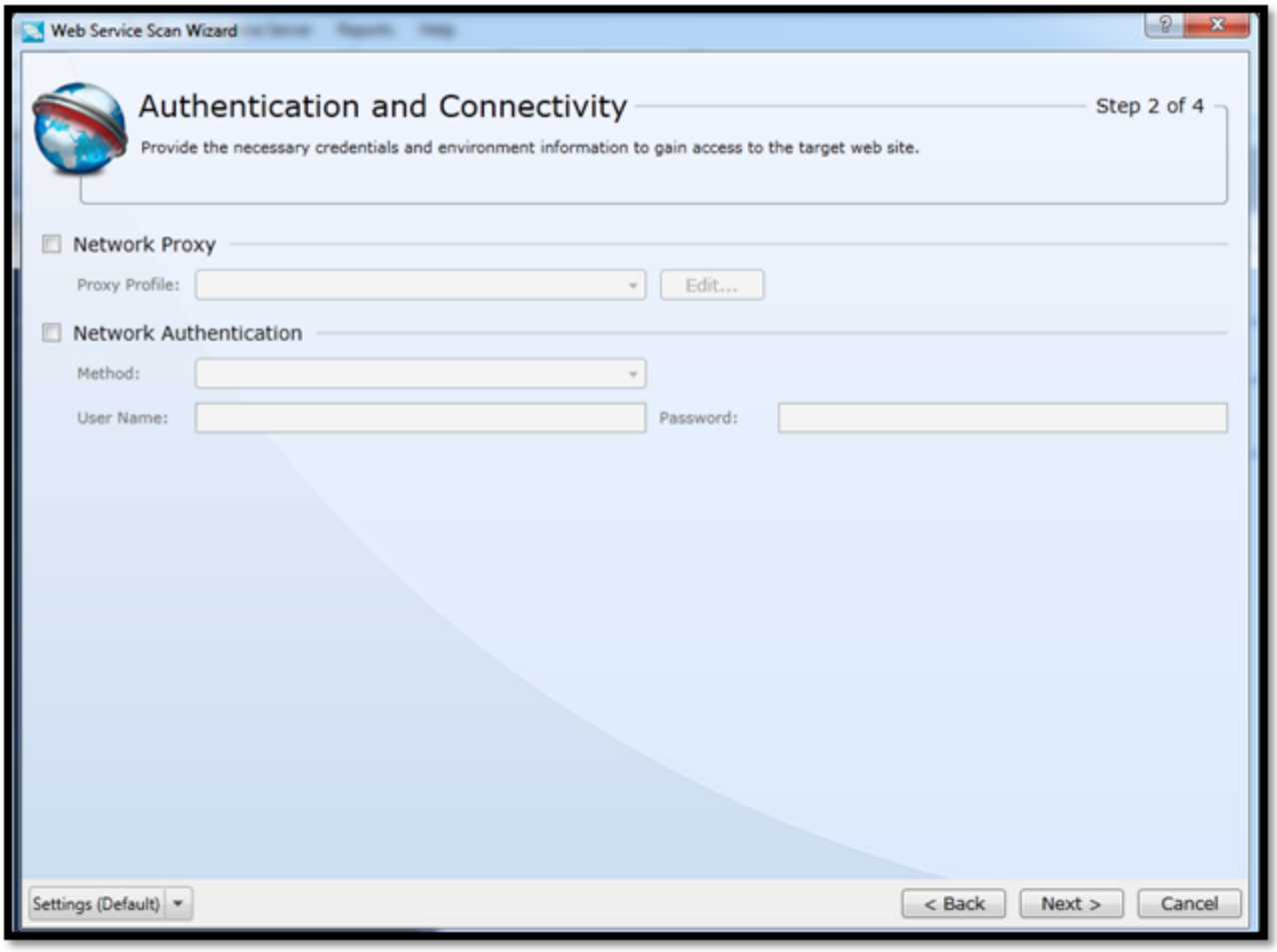


Figure 19: Authentication and Connectivity Window

Click on "Next" to open the next window, which contains "Detailed Scan Configuration" but, before that you will be prompted with a pop-up, "Would you like to launch the Web Service Test Designer Now?", as shown in Figure 20.

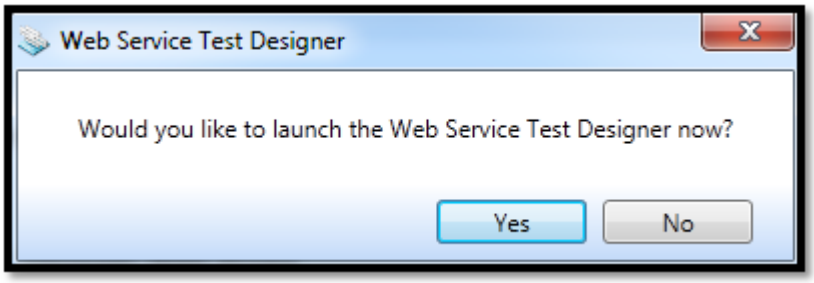


Figure 20: Web Service Test Design Prompt

Click on "Yes" to open a new "Web Services Test Designer" window. This window contains all the methods in the provided WSDL file in the top left corner. If you want to add other methods or want to remove any methods, just check or uncheck that method, as shown in Figure 21.

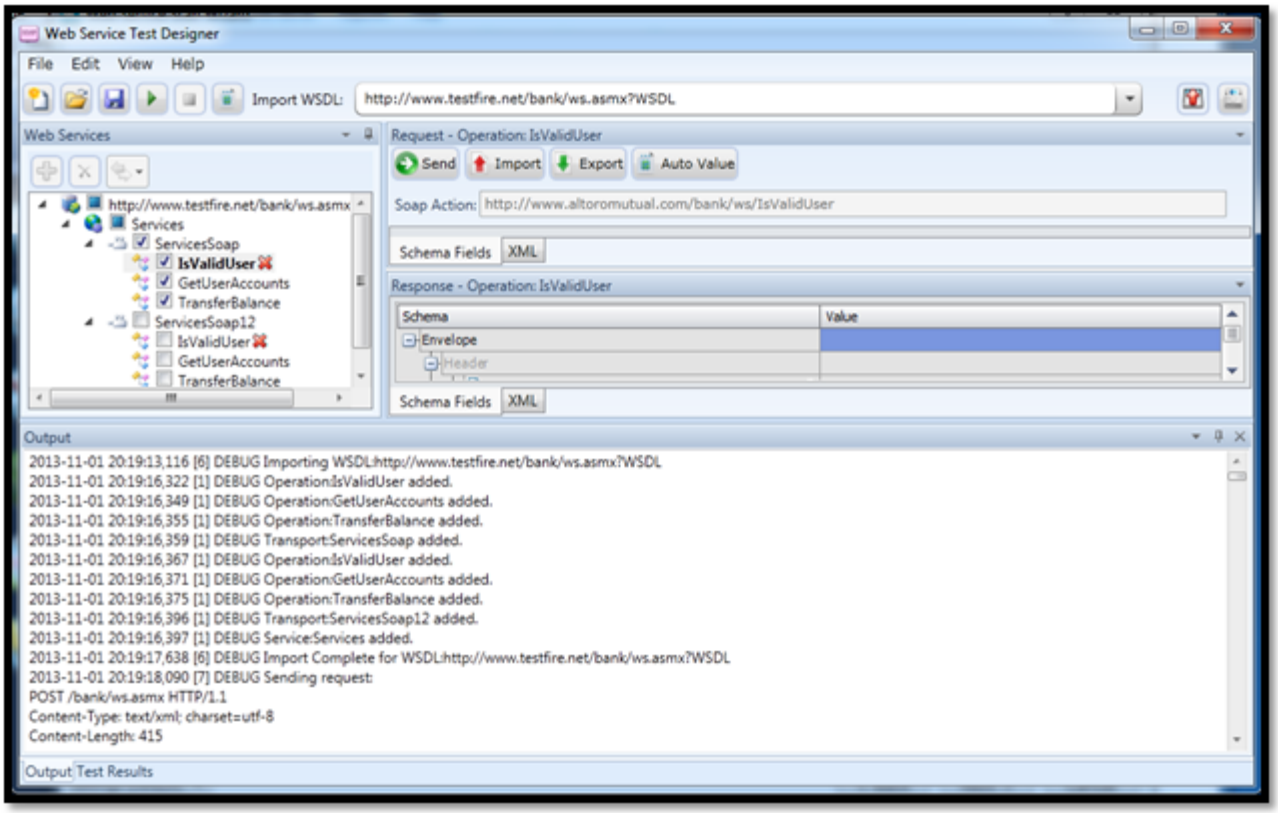


Figure 21: Web Service Test Designer Window

I mentioned earlier that WebInspect is a popular black box web services testing tool and here is the reason: It not only imports all the methods from the WSDL but also fills in the values of required data types in the parameter. So, as a pen tester, you just need to provide a valid WSDL to WebInspect and it will do the rest of the things for you, unlike the other tools, where you need to manually insert data in each method. There is one limitation: Sometimes WebInspect is unable to fill in the proper data type in a required parameter and is unable to detect that method. In our case, WebInspect is unable to detect the IsValidUser method, so a red cross is displayed at the right side of that method.

Now close the "Web Services Test Designer" window. It will prompt you to save the designer file. Click on "Yes" and save it by providing a name for your test designer file in your computer. I saved it as test4. And you will get the name auto set in the names field in Detailed Scan Configuration window as shown in Figure 22.

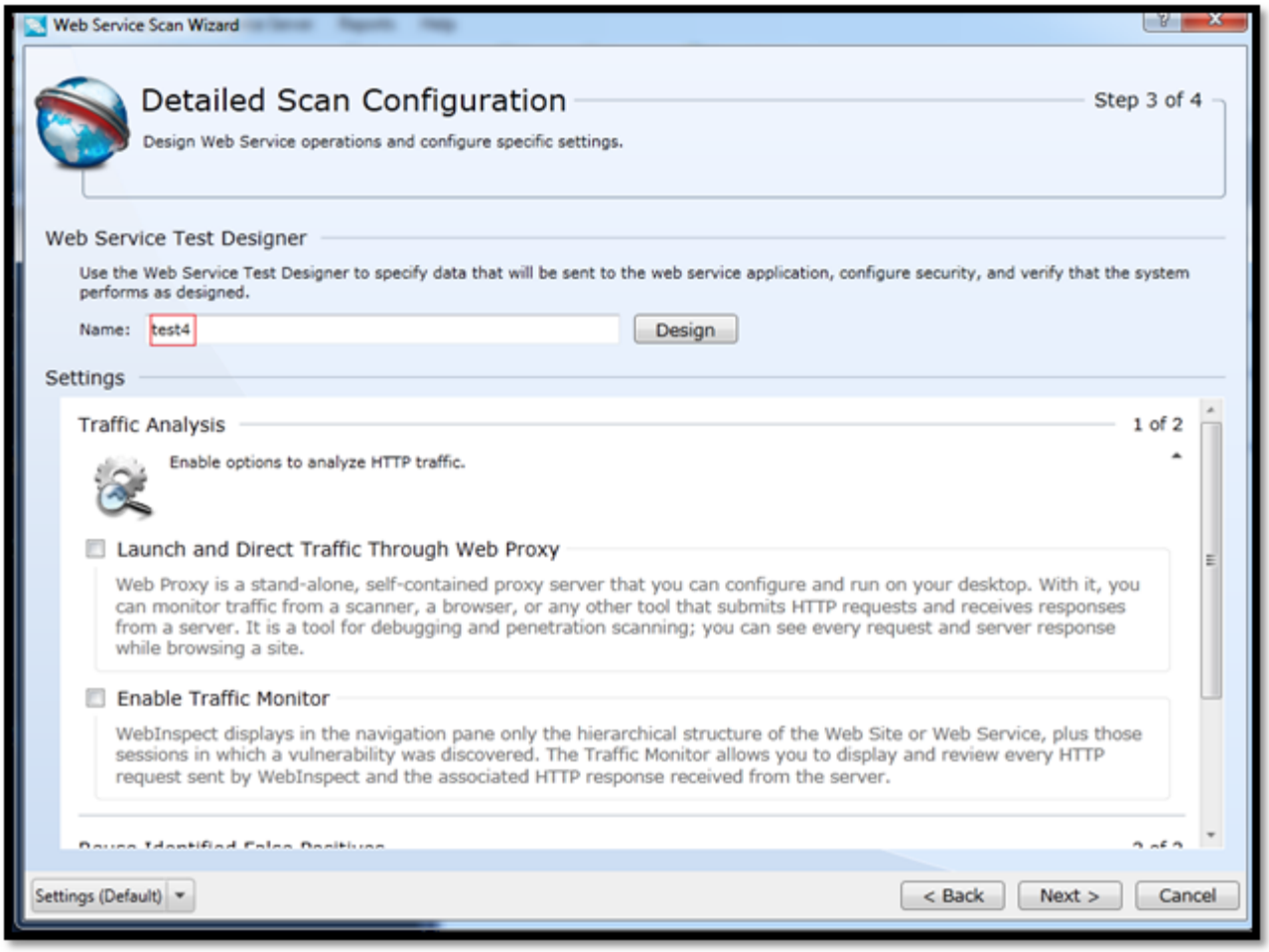


Figure 22: Detailed Scan Configuration Window

You can see other settings in the same window. If you want any customized settings, you can enable them but, in this case, I am proceeding with the default settings. Click on "Next" to complete the wizard, as shown in Figure 23.

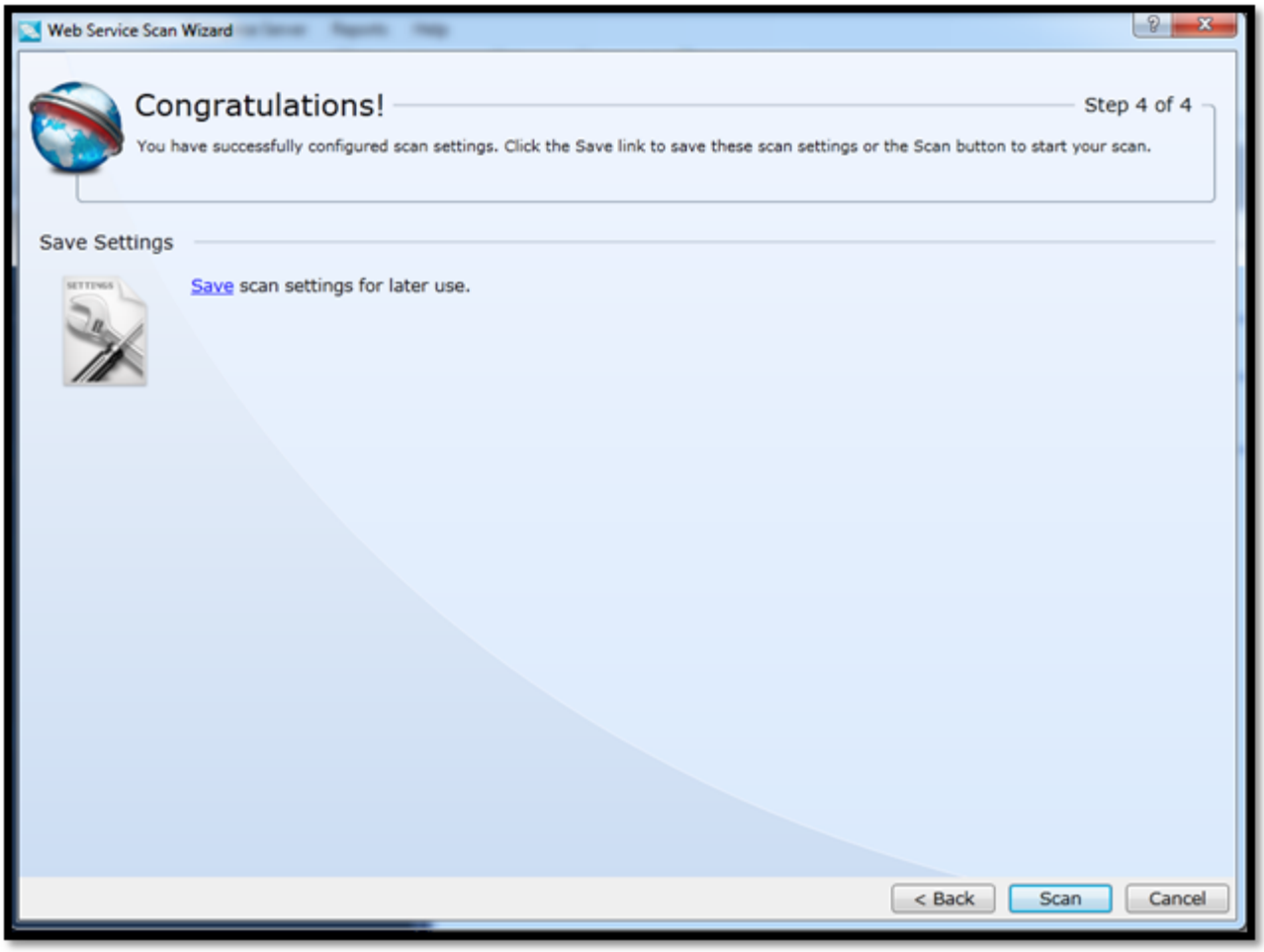


Figure 23: Web Services Scan Wizard Final Window

You will get a congratulation message there. Now click on "Scan" to start the scan. WebInspect will scan the web service and provide you with the vulnerability report, as shown in Figure 24.

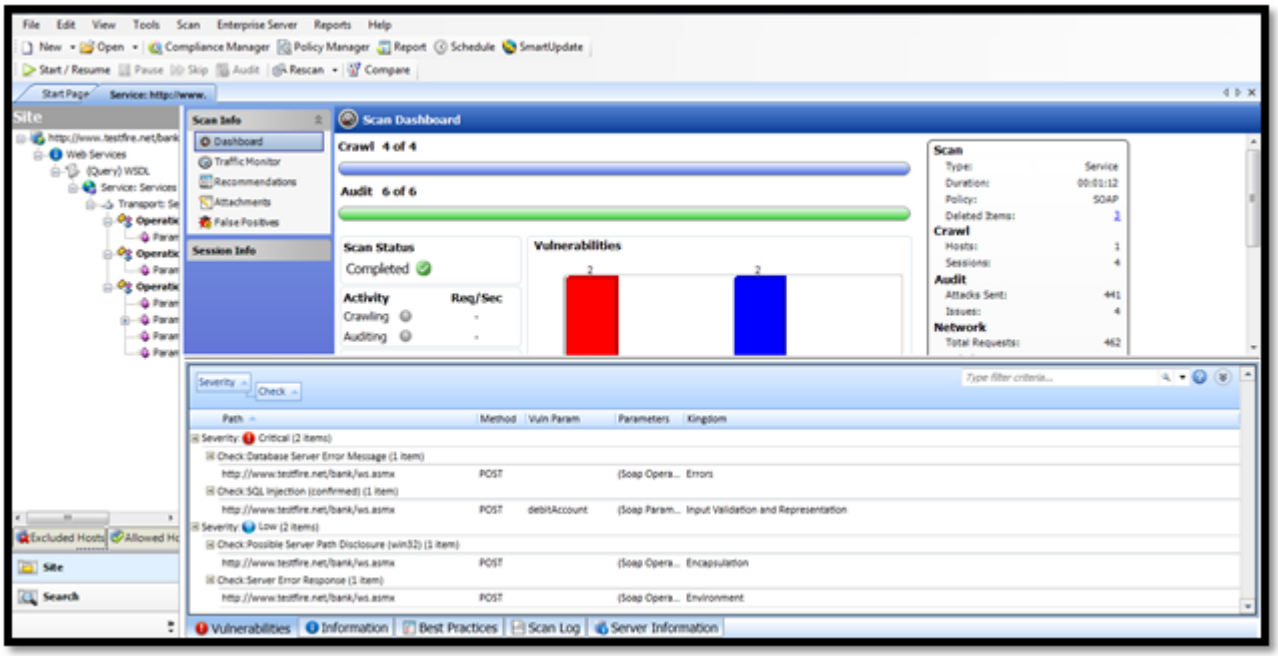


Figure 24: WebInspect Result

Here the results of the web service scan are displayed. Now you can use WebInspect to test any web service, especially black box, to discover the existing vulnerabilities and you need to verify it manually to avoid false-positives.

Conclusion

Any automated tool will help you to get good results from a penetration testing and will reduce your time and effort, but it's always better to use them just for coverage and focus more on manual testing, because automated tools can provide false positives and false negatives as well.

Chapter 4

In the previous article, we discussed the automated tools available for testing web services, how to automate web services penetration testing using different automated tools, and also why the automation of web services penetration test is not sufficient and manual testing is needed.

In this article, we will focus on the open source tools available to pen test web services manually and why it is so important.

Manual Testing

Manual testing covers lots more different types of nontraditional test cases that will help a pen tester to understand the functionalities and trying different new approaches based on the scenario, rather than fuzzing the general payloads to cover traditional vulnerabilities. It allows a pen tester to think out of the box, which may lead to a zero-day. It also provides freedom to a pen tester to test different business logic vulnerabilities that are literally impossible to cover by a auto scanner.

For manual testing also we need some kind of tools that will help us, so today we will start with the open source manual tools that can be used test web services. We will start with a simple yet effective Mozilla Firefox add-on, SOA Client.

SOA Client

SOA Client (https://addons.mozilla.org/en-US/firefox/addon/soa-client/) is a Mozilla Firefox add-on by Michael Santoso. It is a portable client to access web services and UDDI registries. It is easy to install and it has a user-friendly interface to perform web services penetration testing manually, as shown in Figure 1.

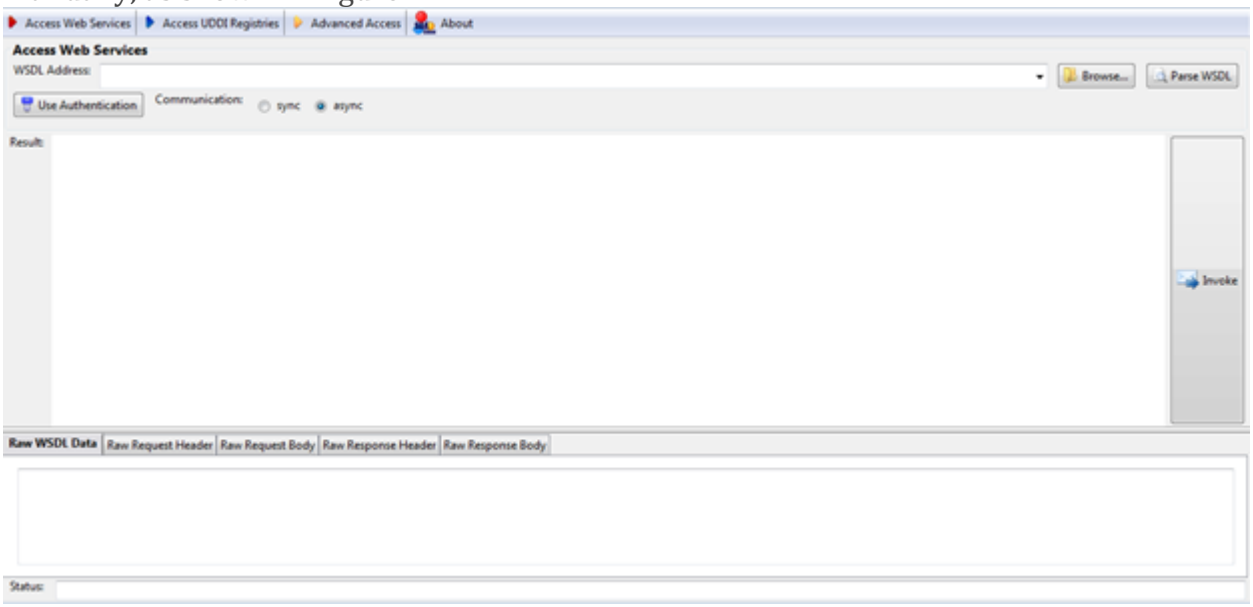


Figure 1: SOA Client window

As I stated earlier, SOA Client is used to access WSDL and UDDI. Below is a list of some web services and UDDI registries that are on the SOA Client page.

Web Services

Credit card verification: https://ws.cdyne.com/creditcardverify/luhnchecker.asmx?wsdl
Census information: http://ws.cdyne.com/DemographixWS/DemographixQuery.asmx?wsdl
Currency foreign exchange: http://www.xignite.com/xCurrencies.asmx?WSDL

Email address validation: http://www.webservicex.com/ValidateEmail.asmx?WSDL
English dictionary: http://services.aonaware.com/DictService/DictService.asmx?WSDL
Number conversion:
http://www.dataaccess.com/webservicesserver/numberconversion.wso?WSDL
Image converter (e.g., PSD into JPG): http://www.bigislandcolor.com/imageconvert.wsdl
IP address into location: http://ws.cdyne.com/ip2geo/ip2geo.asmx?wsdl
Stock quote: http://ws.cdyne.com/delayedstockquote/delayedstockquote.asmx?wsdl
Translator (English to Chinese):
http://fy.webxml.com.cn/webservices/EnglishChinese.asmx?wsdl
FIFA World Cup 2010:
http://footballpool.dataaccess.eu/data/info.wso?WSDL
Weather forecast: http://www.webservicex.net/WeatherForecast.asmx?WSDL

UDDI Registries
http://hma.eoportal.org/juddi/inquiry
http://registry.gbif.net/uddi/inquiry
http://test.uddi.microsoft.com/inquire
As you can see in Figure 1, There are four tabs in SOA Client:

1. Access Web Services
2. Access UDDI Registries
3. Advanced Access
4. About

In this article, we will be using mostly the "Access Web services" and "Advanced Access" options, since we are going to use a public-facing demo web service and not the UDDI registry. So, for this case, we are going to use the same (http://www.testfire.net/bank/ws.asmx?WSDL) for manual testing.

Manual Testing with SOA Client
First open your SOA Client and put this WSDL URL, as shown in Figure 2.

Figure 2: Access Web Services tab
Then Click on "Parse WSDL" to import all the operations present in that web service, as shown in Figure 3.

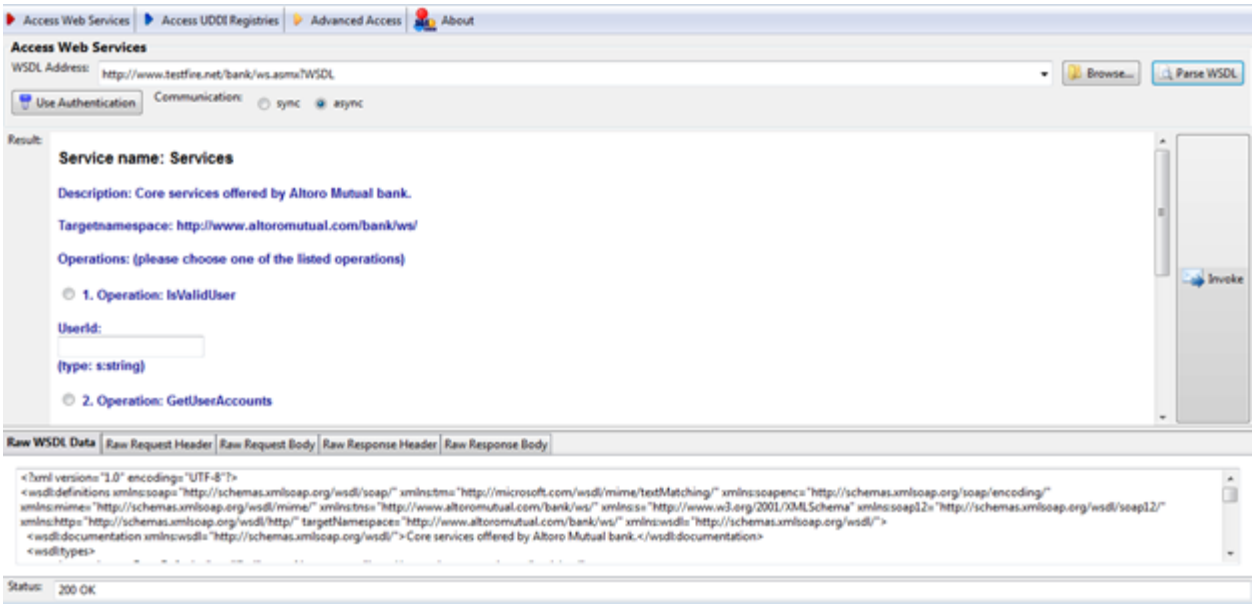


Figure 3: Parsed WSDL window

In the result page, you will get all the operations available in the web service. And, as you can see, there are five tabs:

1. Raw WSDL Data
2. Raw Request Header
3. Raw Request Body
4. Raw Response Header
5. Raw Response Body

This is a very good, light-weight, simple GUI tool to test web services manually. Let's start the test by selecting the "GetUserAccounts" operation, as shown in Figure 4.

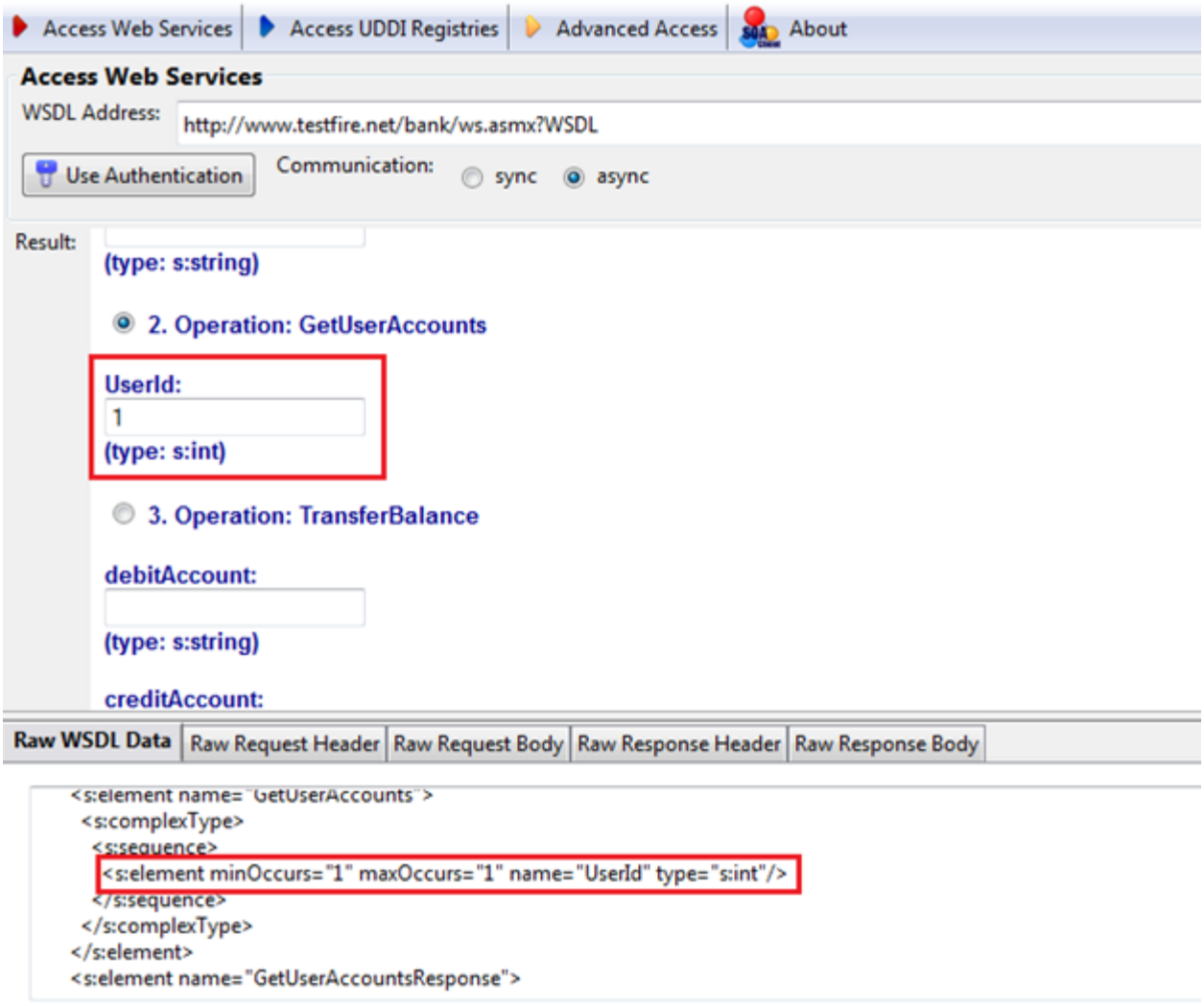


Figure 4: invoking GetUserAccounts operation

As we can see here, the "GetUserAccounts" operation contains only one parameter of "int" data type and we provided an appropriate value there. Now click on "Invoke" (because Figure 4 is a partial image, you will see the "Invoke" button on the very right side of this window. as shown in Figure 3) to send the request.

Since we provided it with a proper request, we get a valid response "200 OK" without any error. (If you remember, in the previous article "**Web Services Penetration Testing, Part 2: An Automated Approach With SoapUI Pro**" in Figure 10: XML View, we used the same method and value and got a response with no error; here we got a similar response, as shown in Figure 5.)

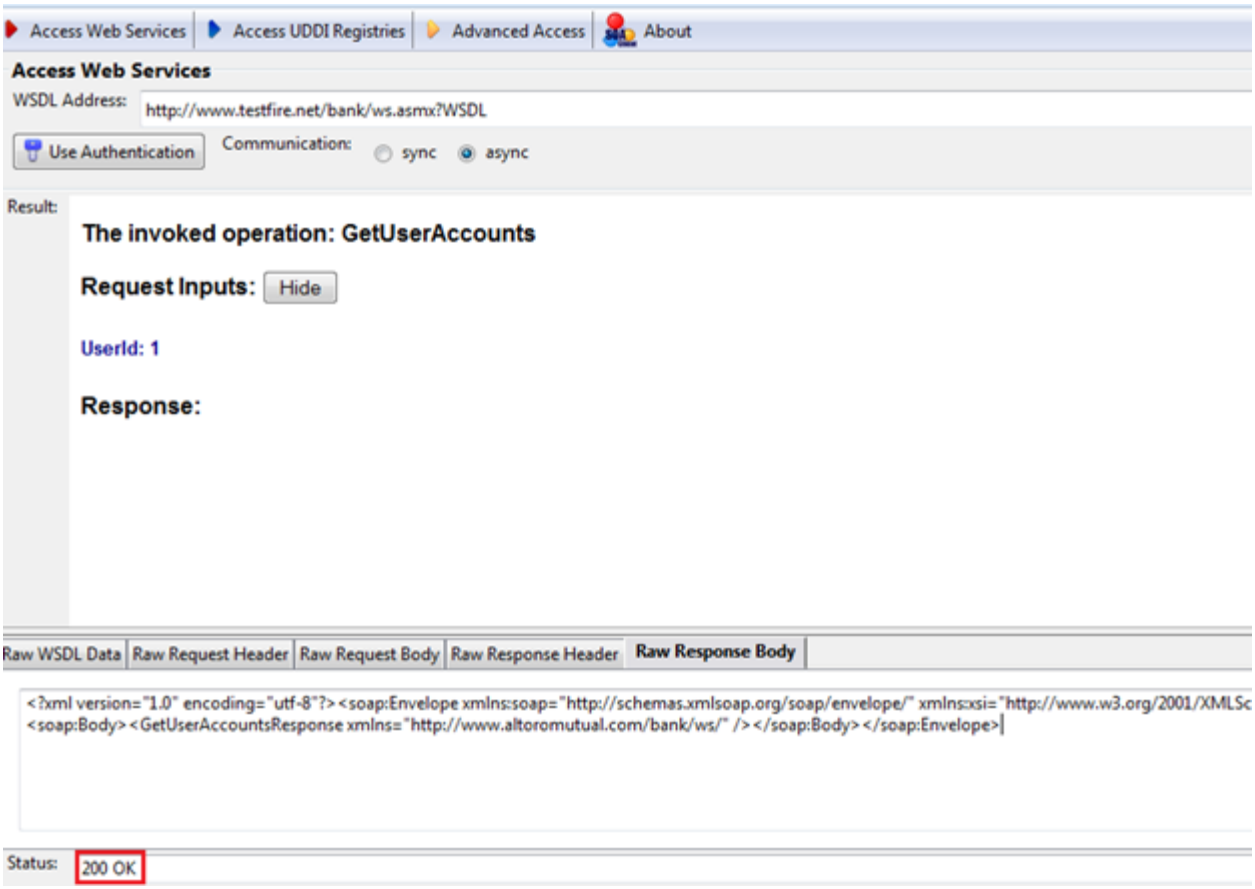


Figure 5: Response to GetUserAccounts requests

Similarly, you can change the values in different parameters and invoke the request to get the response. But there are certain problems with this SOA Client; one problem is that sometimes it won't parse the WSDL properly. That's what happened here. As you can see in Figure 6 in the Raw WSDL Data tab, the operation "TransferBalance" has four parameters, but in the GUI, the SOA Client shows only two.

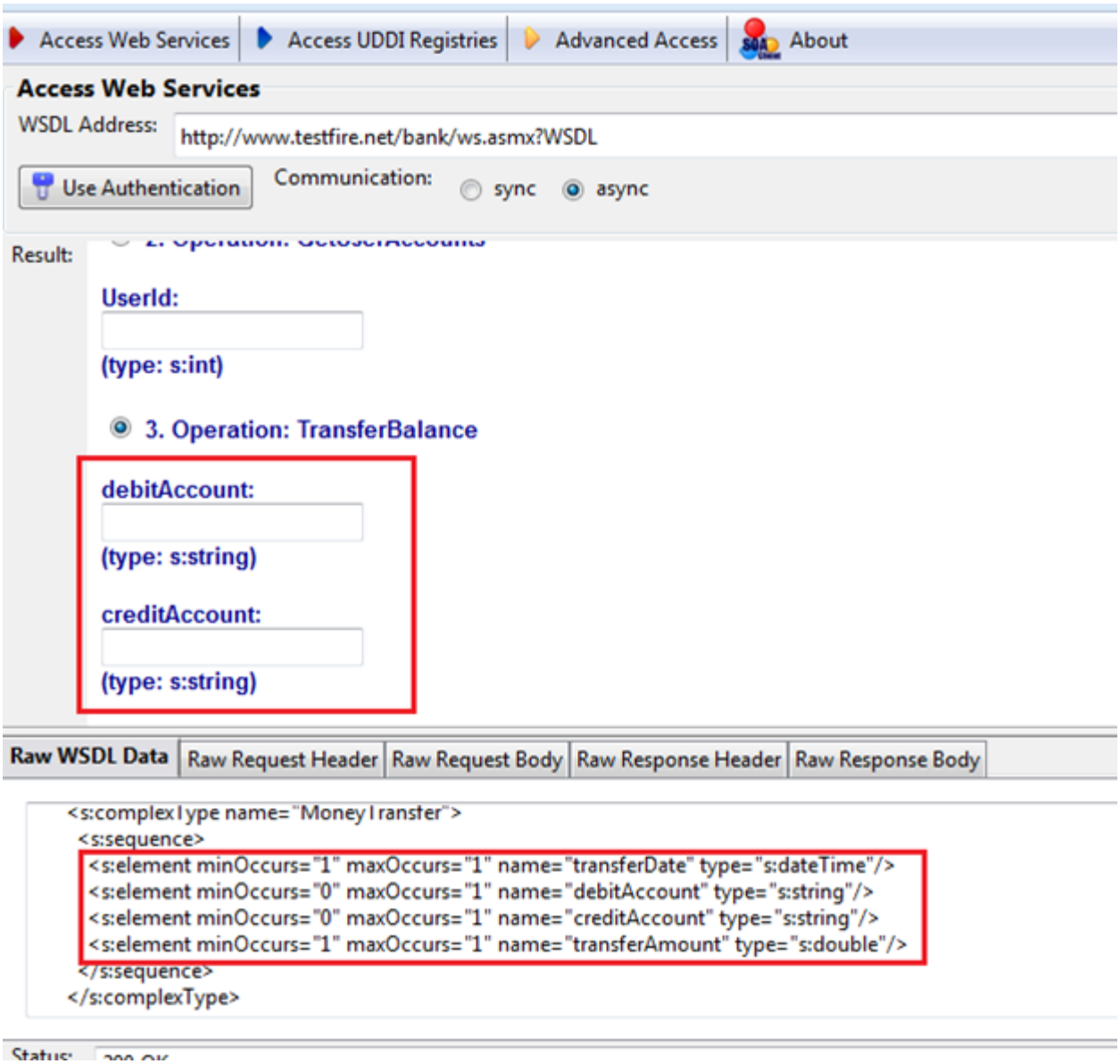


Figure 6: Parsed WSDL operations window

As is very clear from the raw WSDL data, the "TransferBalance" operation needs four values:

1. transferDate (date-time data type)
2. debitAccount (string data type)
3. creditAccount (string data type)
4. transferAmount (double data type)

But the SOA Client only parsed two parameters, "debitAccount" and "creditAccount."

We can come out of this and still perform web service manual penetration testing using its "Advanced Access" option. All we need is the sample request. Most probably, in gray box testing, we are provided with sample requests, but sometimes we get those sample requests due to an information leakage vulnerability.

I prefer this tool when I get a WSDL file and the sample requests are disclosed publicly due to server misconfiguration while performing a Web application penetration test. I use the "Advanced Access" module to check the operation to find vulnerabilities.

So here we assume that we found the sample requests of this WSDL (http://www.testfire.net/bank/ws.asmx?WSDL) and we use them to learn how to use the "Advanced Access" module in the SOA Client tool.
We will use the sample request of the "TransferBalance" operation, which can be found on (http://www.testfire.net/bank/ws.asmx?op=TransferBalance), as shown in Figure 7. The normal "Access Web Services" module is unable to parse it properly.

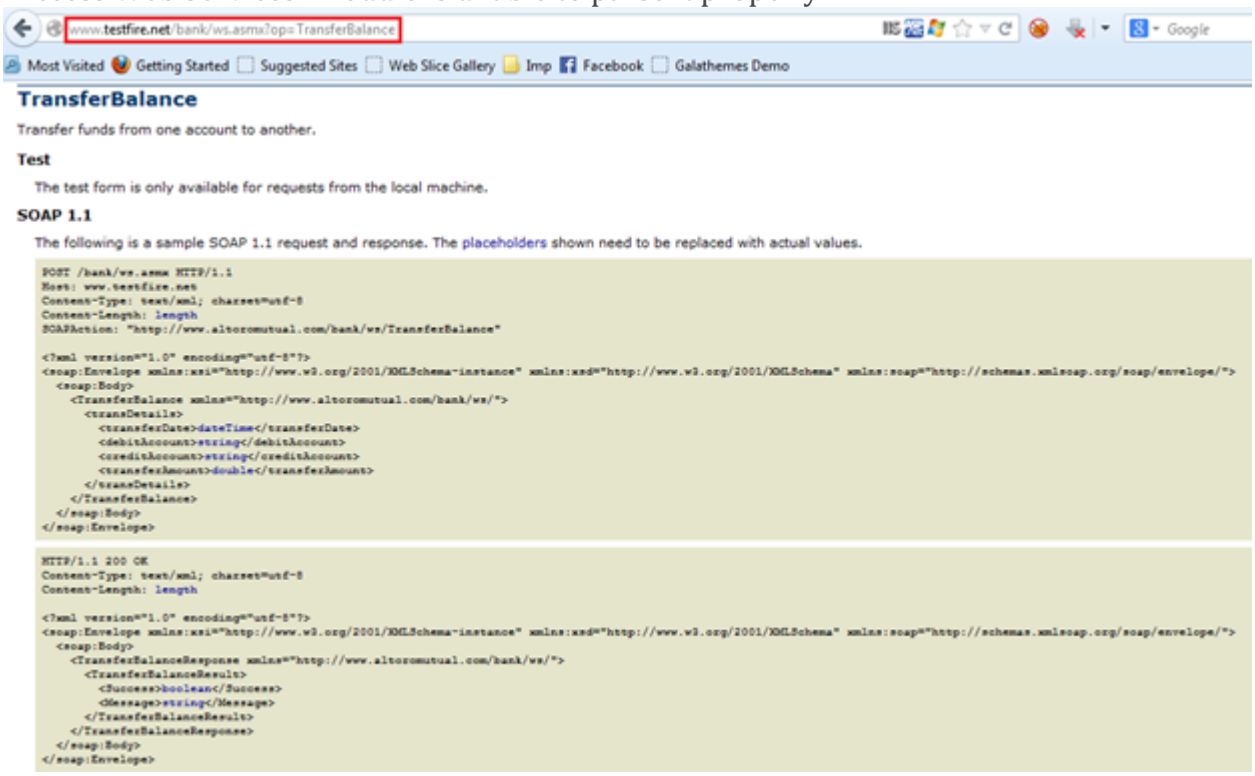


Figure 7: Sample request for the TransferBalance operation

Before starting the testing, first let's have a check on the "Advanced Access" module interface to understand what data we need to start the test. The interface can be found in Figure 8.

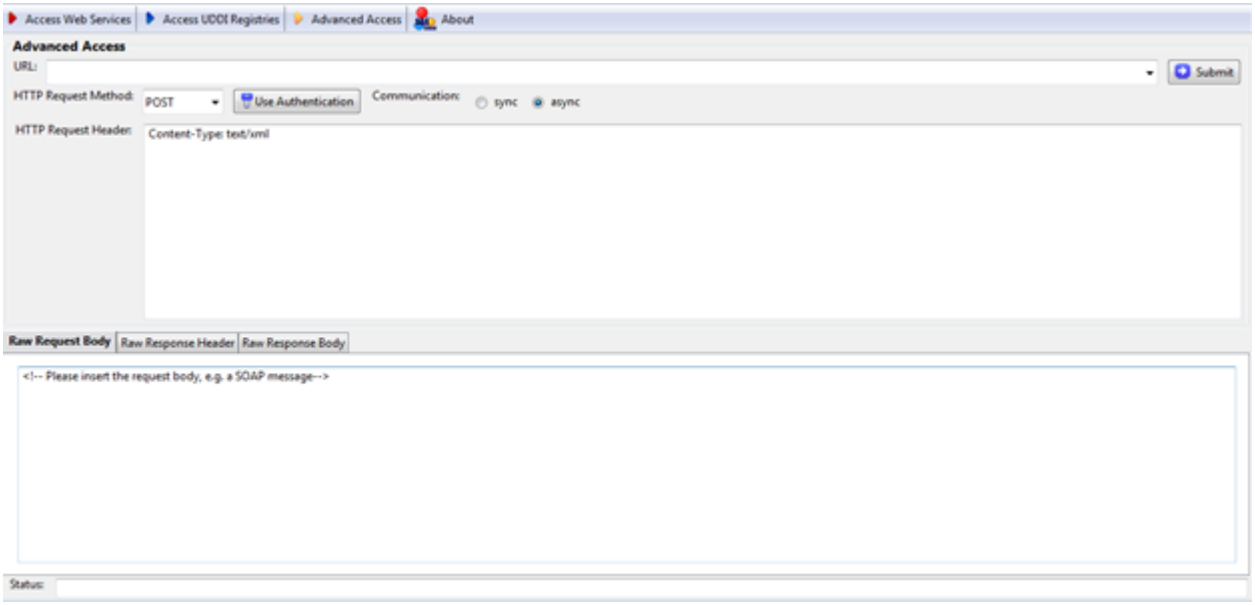


Figure 8: Advanced Access module window

As you can see, we need a URL (which is an end point URL to perform the test). We have to specify the request type (whether we want to use GET, POST, HEADER, TRACE, or OPTIONS). We have to deal with the authentication mechanism, if any is needed, and we need to determine the communication type, synchronized or asynchronized. We also need the HTTP request header (If any specific header has to be added) and, last but not least, the SOAP request. We can find all these details in the sample request. But before that, we need to understand how to differentiate all these requirements from that one sample request, as shown in Figure 9.

```
POST /bank/ws.asmx HTTP/1.1
Host: www.testfire.net
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://www.altoromutual.com/bank/ws/TransferBalance"

<!--?xml version="1.0" encoding="utf-8"?-->

dateTime string
string
double
```

Figure 9: Sample request of Transfer Balance operation

This is the same request from Figure 7. Here we can see in the very first line that the request method is POST. And from line 1 and line 2 we can deduce that the URL used must be http://www.testfire.net/bank/ws.asmx. In the header there is nothing special that we need to use in the "Advanced Access" module. The type of communication is not specified, so we can go with the default options. There is no authentication mechanism required for this request, so we will not touch that option in the "Advanced Access" module. And last but not least, we have the SOAP request in the second part of the sample request. This SOAP request contains four parameters.

1. transferDate (date-time data type)
2. debitAccount (string data type)
3. creditAccount (string data type)
4. transferAmount (double data type)

As it's a black box testing, we don't have the exact data that we can use in this request, so we will use some data with the required data type.

1. transferDate (date-time data type) = 2009-01-01T00:00:00
2. debitAccount (string data type) = test
3. creditAccount (string data type) = tester
4. transferAmount (double data type)= 1.0

When we enter all these details in the "Advanced Access" module it will look as shown in Figure 10.

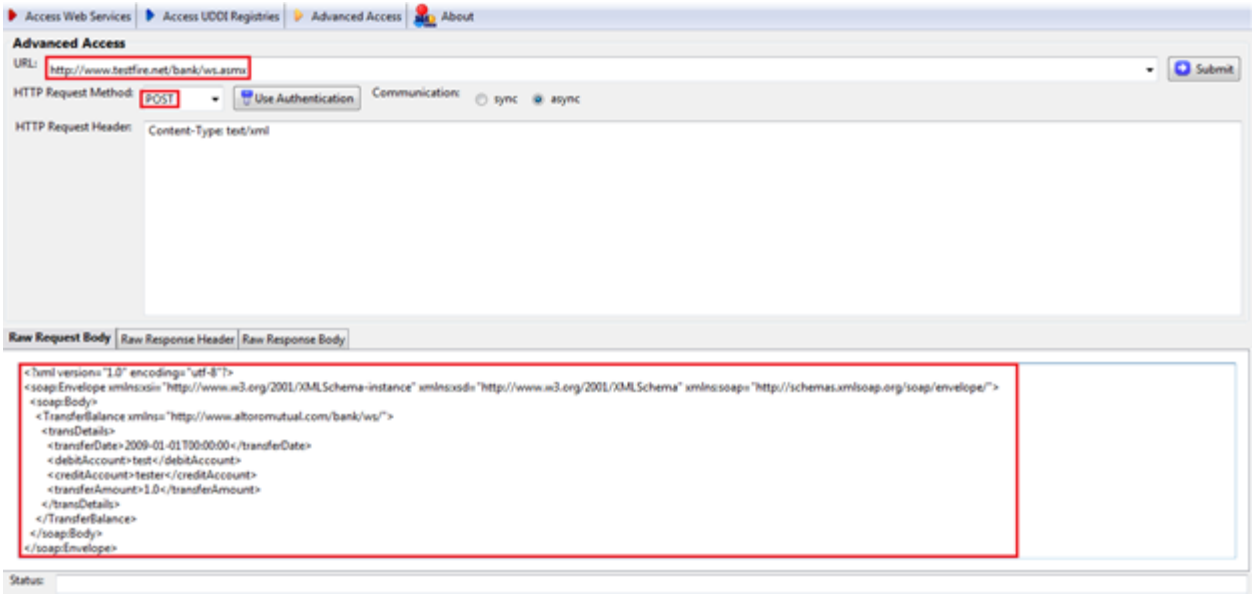


Figure 10: Advanced Access window with required data

The things added or changed are marked in red in Figure 10. The rest are the default settings. Now submit the request by clicking the "Submit" button in the right top corner to get a response. The response is shown in Figure 11.

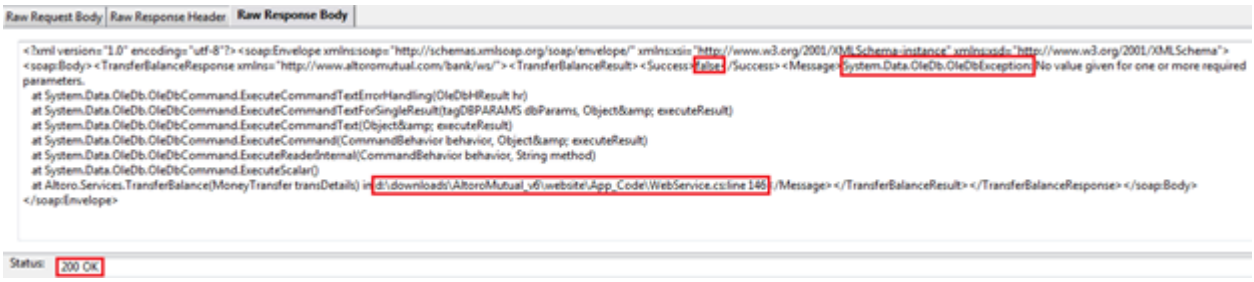


Figure 11: Response of TransferBalance request

As you can see in Figure 11, in the "Raw Response Body" tab we got the "200 OK" with some interesting information.

1. TransferBalanceResult success is false (we are unable to transfer the balance).
2. System.Data.OleDb.OleDbException (we got a "DB Exception"; this is a hint to a possible SQLI).
3. d:\downloads\AltoroMutual_v6\website\App_Code\WebService.cs: line 146 (server path disclosure).

This is how we can use the "Advanced Access" module of the SOA Client tool. This is how we manually test for various vulnerabilities. As we can see, by using one request we got one vulnerability, the server path disclosure, and a hint of another vulnerability, i.e., SQLI. One/some/all of the parameters might be vulnerable to SQLI; you just need to check each of the parameter to get the result.

What we have learned so far is how to use different modules of SOA Client to perform the manual web service penetration testing. There are restrictions, such as the fact that it sometimes won't able to parse the request properly and, to test efficiently, we need the sample request.

Conclusion

Although there are certain limitations, SOA Client is a very light-weight and user-friendly GUI tool. We can use it in most of the cases while performing the penetration testing of web services. But there are certain cases where we are bound to choose another option.

Let's say we need to perform a black box web services penetration testing: In that case, there is no way we will get the sample request from the client. What if the sample request is not exposed to public as it is exposed in our case for testfire.net? What if SOA Client is unable to parse some requests, as we see earlier in this article? We need some other tool to perform manual web services penetration testing and we will learn about that in the next installment.

Chapter 5

In the previous article, we discussed the importance of manual web services penetration testing, how to perform a manual test using SOA Client, how SOA client helps us in most cases, and what the restrictions are that require us to choose other options.

In this article, we will find the solution to those problems that we discussed in the previous part.

The scenario was: What if we are performing a black box test and the sample request is not disclosed to the public? What if, in the same scenario, this SOA Client is unable to parse one of our requests properly? In that case, we need other options to perform the test, and nothing is better than an open source specialist web services testing tool.

Yes, it is soapUI. As we discussed in one of the previous articles, "**Web Services Penetration Testing, Part 2: An Automated Approach With SoapUI Pro**," SoapUI by SMARTBEAR (http://www.soapui.org) is the only one popular tool to test for soap vulnerabilities. It comes in two versions: 1. SoapUI (which is open source) and 2. SoapUI Pro (the commercial version). We have already learned how to use SOAP UI Pro, now it's time to learn about how to use the open source version, i.e., SoapUI (http://www.soapui.org/Downloads/latest-release.html).

Testing Web Services with soapUI

Although the initial process of creating a project is same in both soapUI and soapUI Pro, I just want to quickly cover all of those again. To create a project, open your soapUI. You will see the initial window, as shown in Figure 1.

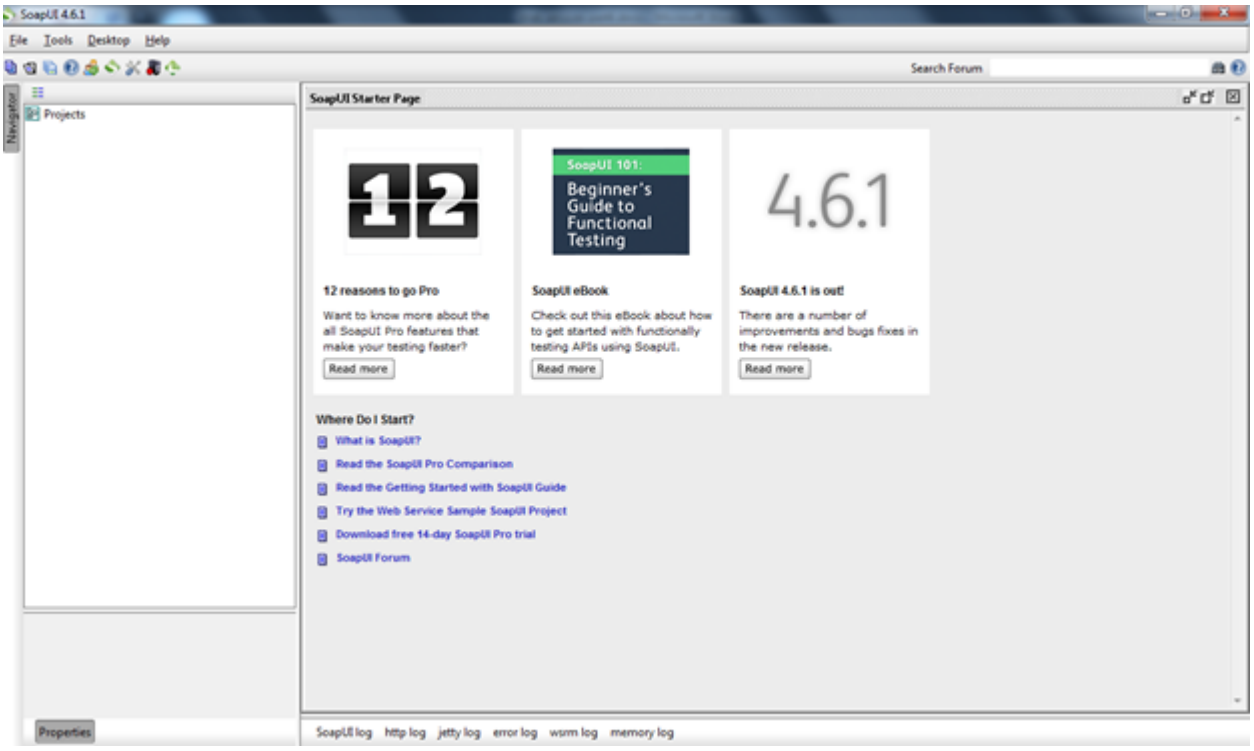


Figure 1: soapUI start window

Click on "File" and "New SOAP Project" to open another window, as shown in Figure 2.

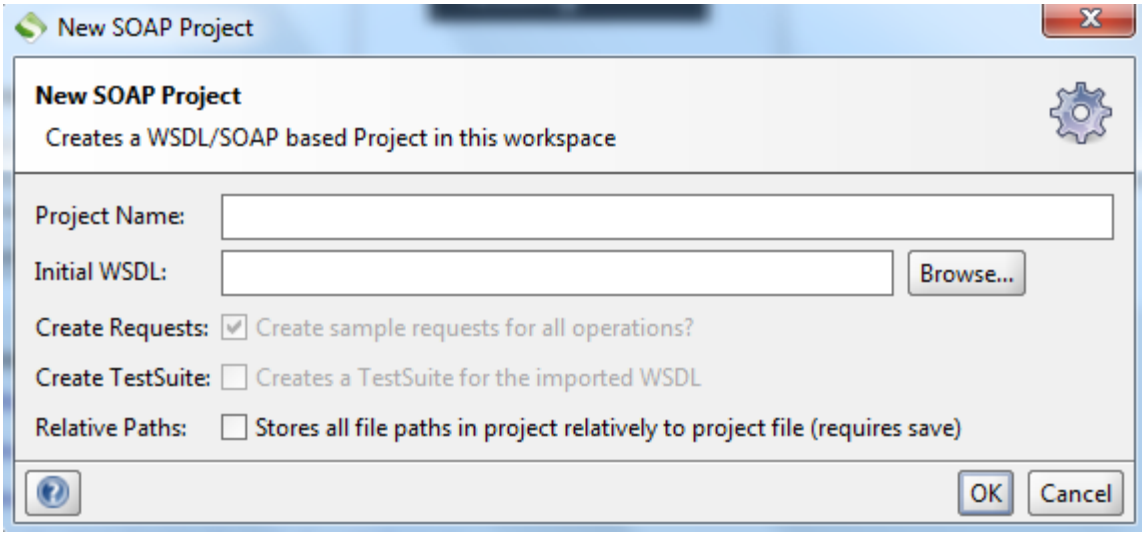


Figure 2: New SOAP Project window

Fill in all the details. Name the project anything you want and the initial WSDL should be either the WSDL file URL or the local path if you have the WSDL file locally. I will use the http://www.testfire.net/bank/ws.asmx?WSDL URL for testing, as shown in Figure 3.

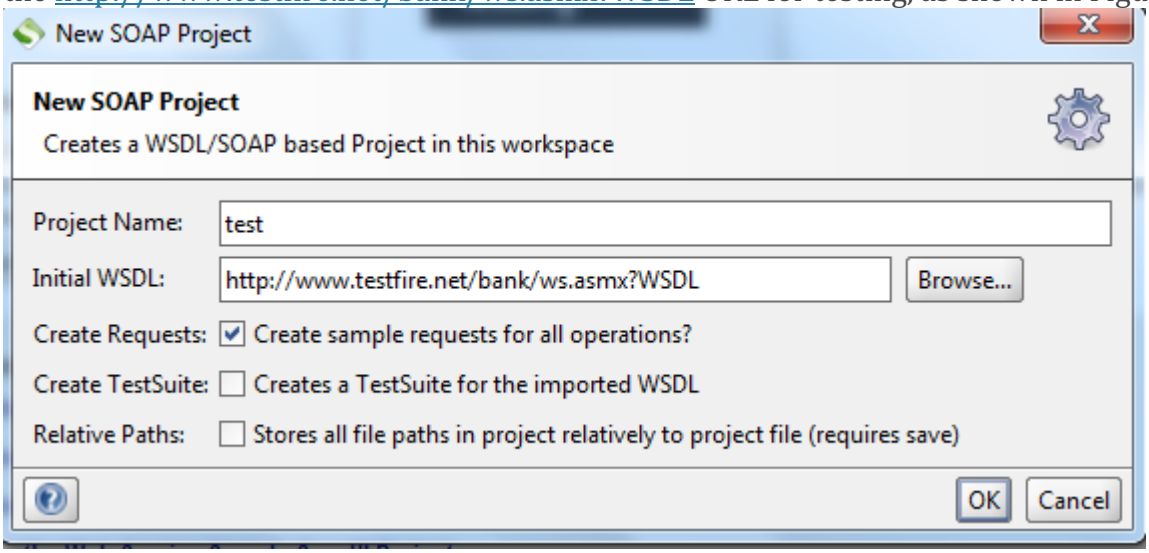


Figure 3: New SOAP Project window with data

Then click "OK" to load all the definitions and you will get all the definitions in your soapUI, as shown in Figure 4.

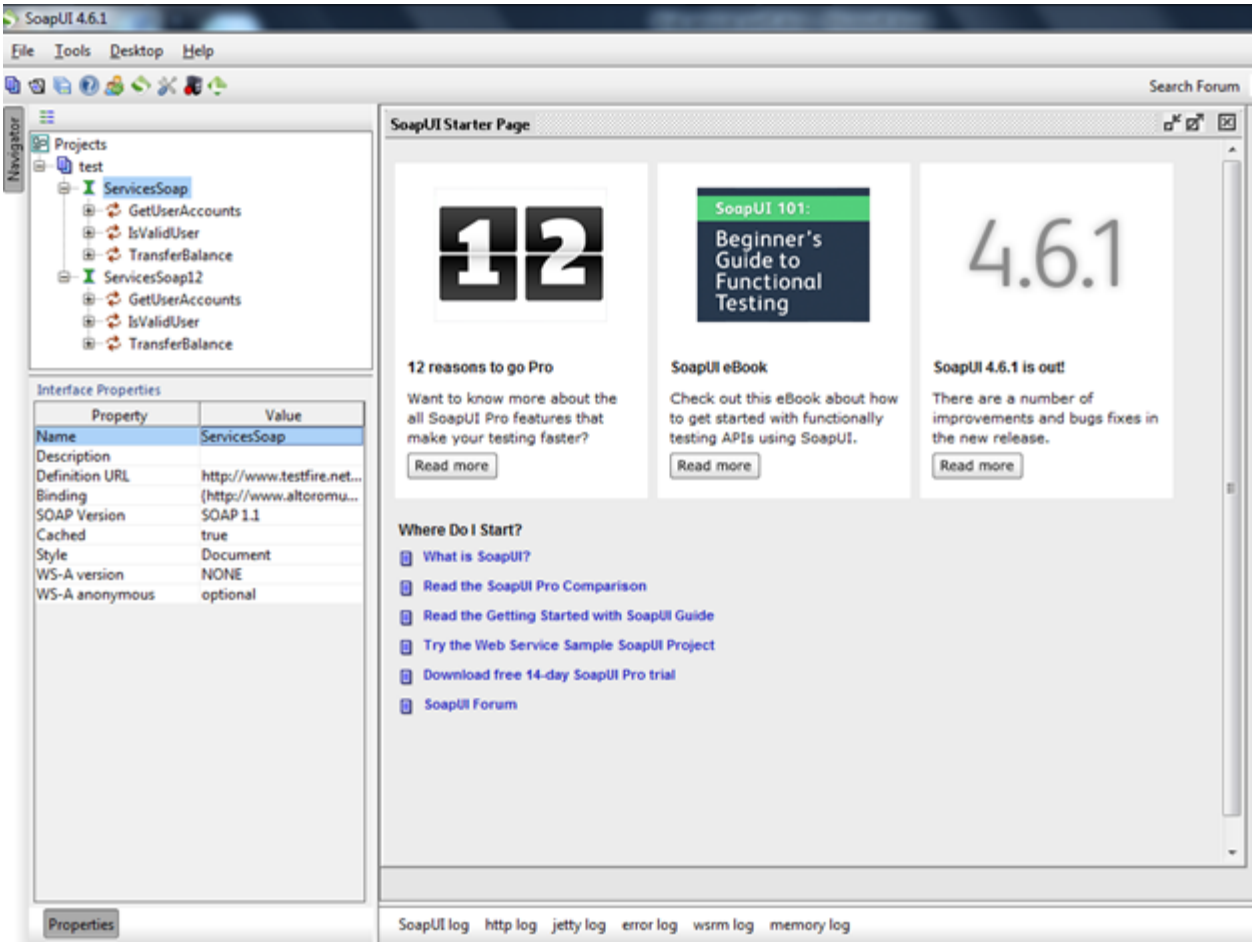


Figure 4: soapUI window with imported definitions

A similarity between soapUI and soapUI pro is that you can edit the interface properties, as shown in the left bottom corner, but in soapUI we can't add new properties. And the most important thing is that the security automation feature is only present in soapUI Pro but, apart from that, in the case of manual testing, soapUI is as powerful as soapUI Pro.

Now click on any method you want to test. Let's say we were facing a problem in the TransferBalance method in SOA Client, so we will start from there to check whether or not we will face something like that here. The request editor window is shown in Figure 5.

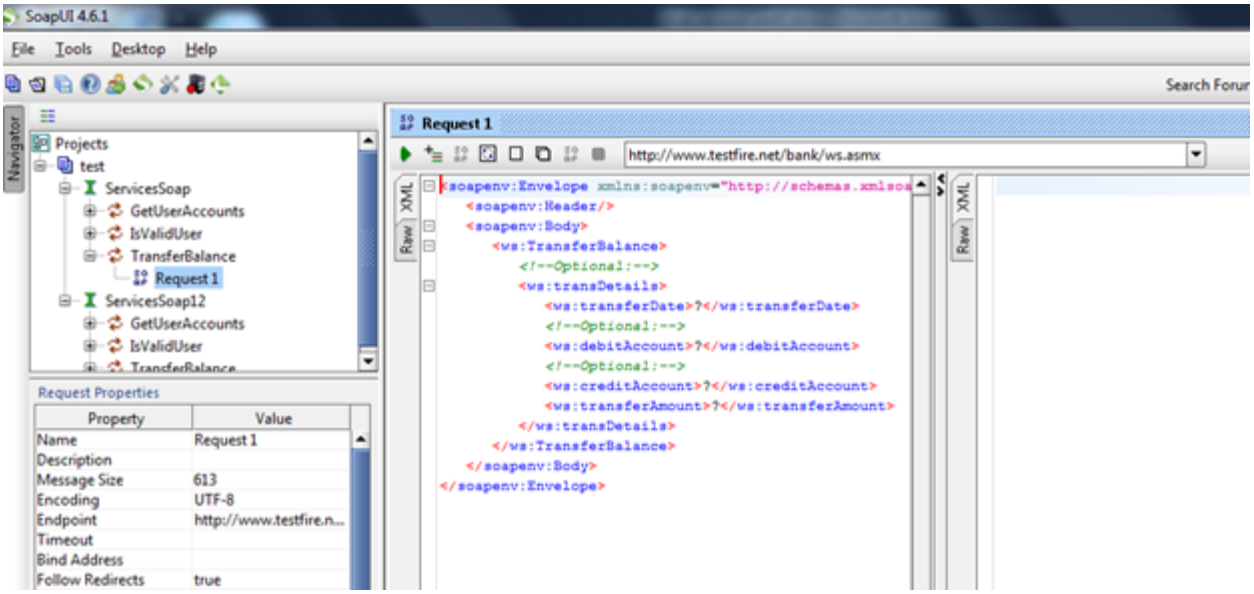


Figure 5: Request editor window

Unlike soapUI Pro, the request editor window of soapUI contains only two tabs:

1. XML
2. RAW

Mostly we will use XML and, unlike the SOA Client, soapUI parsed the method successfully, so we don't have a problem generating proper requests.

Though soapUI is an excellent tool, it has a major disadvantage in the case of black box testing: It won't show you the data type required to test a particular request. In this case we have three options:

1. You need to guess or fuzz all types of data types in different parameters. But this is not a smart way of testing and also very time-consuming.
2. You can get help from some other tool, such as SOA Client, to get these details.
3. Read the WSDL file and understand the requirements of the request (to understand different elements of WSDL go through **Web Services Penetration Testing Part 1**)

Let's start with the first option to guess the values we need to understand the request properly. We can see the request in Figure 6.

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ws="http://www.altoromutual.com/bank/ws/">
<soapenv:Header/>
<soapenv:Body>
<ws:TransferBalance>
<!--Optional:-->
<ws:transDetails>
<ws:transferDate>?</ws:transferDate>
<!--Optional:-->
<ws:debitAccount>?</ws:debitAccount>
<!--Optional:-->
<ws:creditAccount>?</ws:creditAccount>
<ws:transferAmount>?</ws:transferAmount>

</ws:transDetails>
</ws:TransferBalance>
</soapenv:Body>
</soapenv:Envelope>

Figure 6: transferBalance Request

Here we have four parameters; let me guess the data type by its parameter name.

1. transferDate(this must be in date format) = 2013-11-11
2. debitAccount (this must be integer type) = 112233
3. creditAccount (this must be also of integer type) = 445566
4. transferAmount (this must be double) = 100.00

As it is black box testing and we don't have a clue for the data types, we will use what we have guessed. After inserting all those data, the request looks like Figure 7.

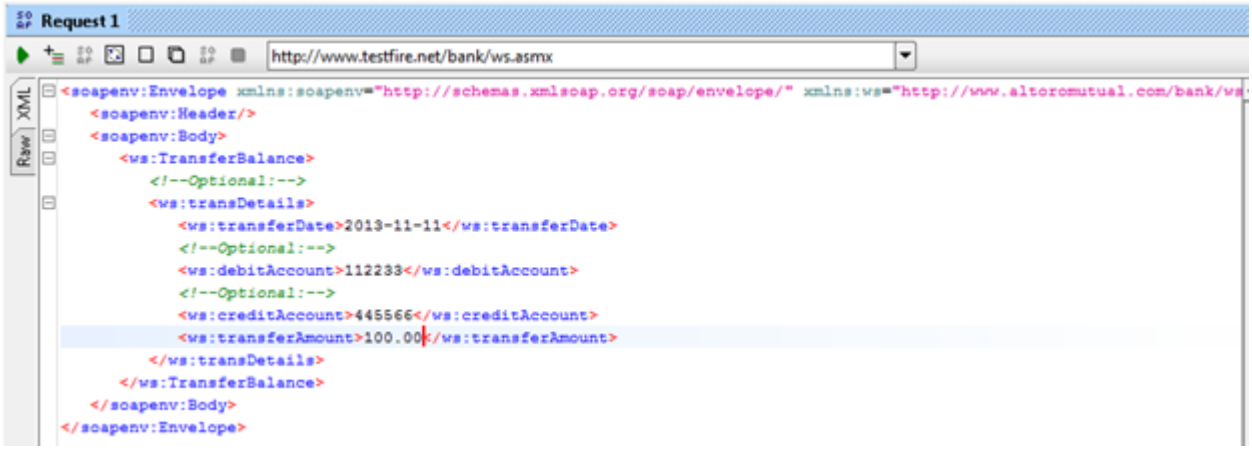


Figure 7: transferBalance request with data

Let's send this request by clicking the green button in the top left corner of the request window, as shown in Figure 7, and we will get the response as shown in Figure 8.

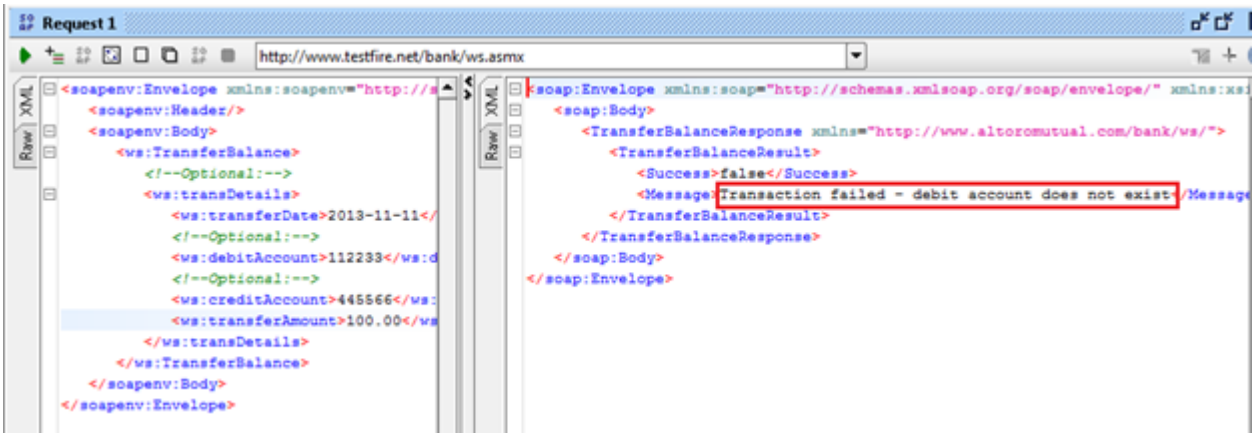


Figure 8: showing response of transferBalance request

We got the response, "Transaction failed – debit account does not exist," but we don't know whether it's a successful response or error response. To check that, click on the RAW tab of the response window as shown in Figure 9.

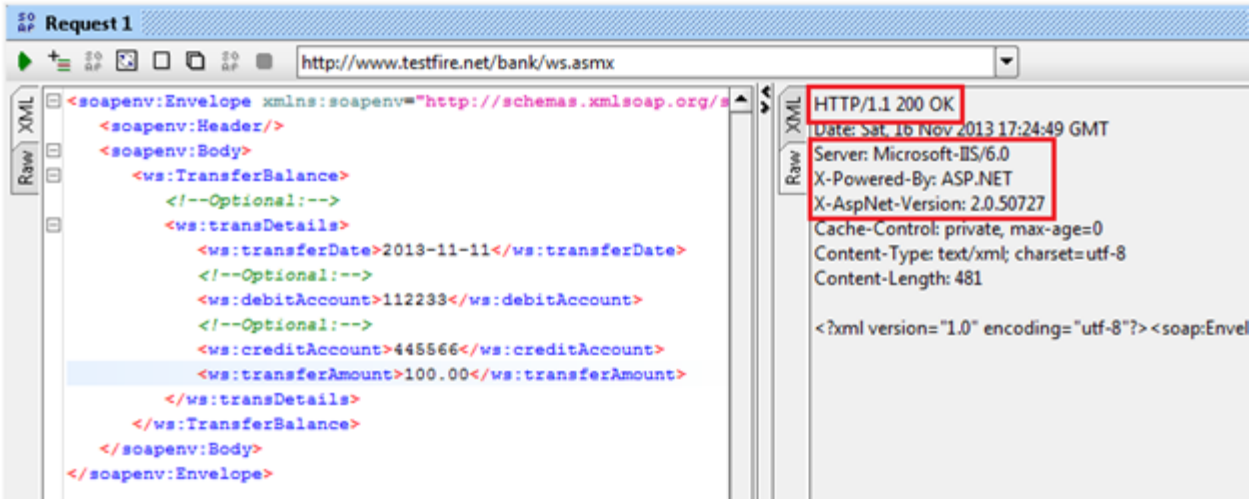


Figure 9: Raw response window

Two things we get from that response:

1. Our request executed properly.
2. We got a vulnerability, i.e., a header version disclosure.

The SOAP response header discloses the following version details:

Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727

Everything worked fine, but we still don't know whether the data types we assumed are correct or not, so let us move to the next option, i.e., collecting data type information from SOA Client.

We learned how to test using SOA Client in the "Web Services Penetration Testing Part 4: Manual Testing with SOA Client," so, without taking much time, I will directly show you where to collect this information. Just check it in Figure 10.

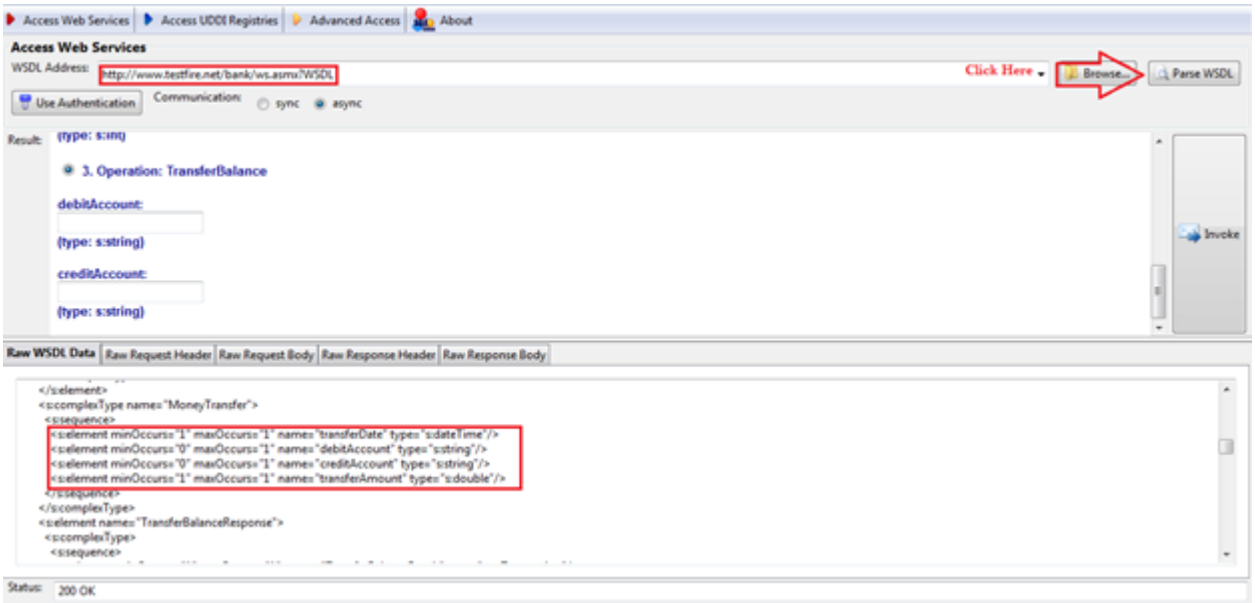


Figure 10: SOA Client showing parameters with required data type

It's a very simple process. Open SOA Client. In "URL," provide the WSDL URL. Click on "Parse WSDL" and in the result window you can find the parameters with data types. But sometimes it is unable to parse a WSDL properly; in that case, there is nothing to worry about; just go below and, in the "Raw WSDL Data," you can find the parameters as well as the data type needed under the particular method or operation name.

As you can see from Figure 10, the required parameters with needed data types are:

1. "transferDate"(date-time)
2. "debitAccount" (string)
3. "creditAccount" (string)
4. "transferAmount" (double)

As we discussed earlier, the same can be also found at the WSDL, as shown in Figure 11.

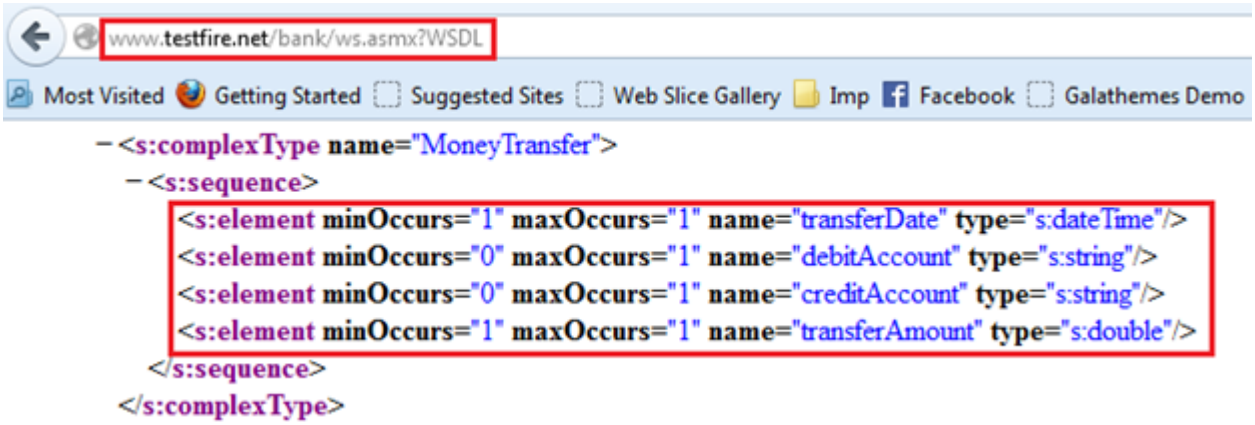


Figure 11: WSDL showing parameters with required data types

We can get these details in both ways discussed above, but I personally feel that it's better to understand the WSDL file and better to be tool-independent. But both the options are in front of you. You can choose whichever one you are comfortable with.

So, as we now have all these details, we can easily fill the appropriate data types and send the request in soapUI. The data we are going to use are:

1. transferDate(date-time) = 2013-01-01T00:00:00
2. debitAccount (string) = 1001160140
3. creditAccount (string) = 1001160141
4. transferAmount(double) = 1.0

Let's send the request with these data. It will look like Figure 12:

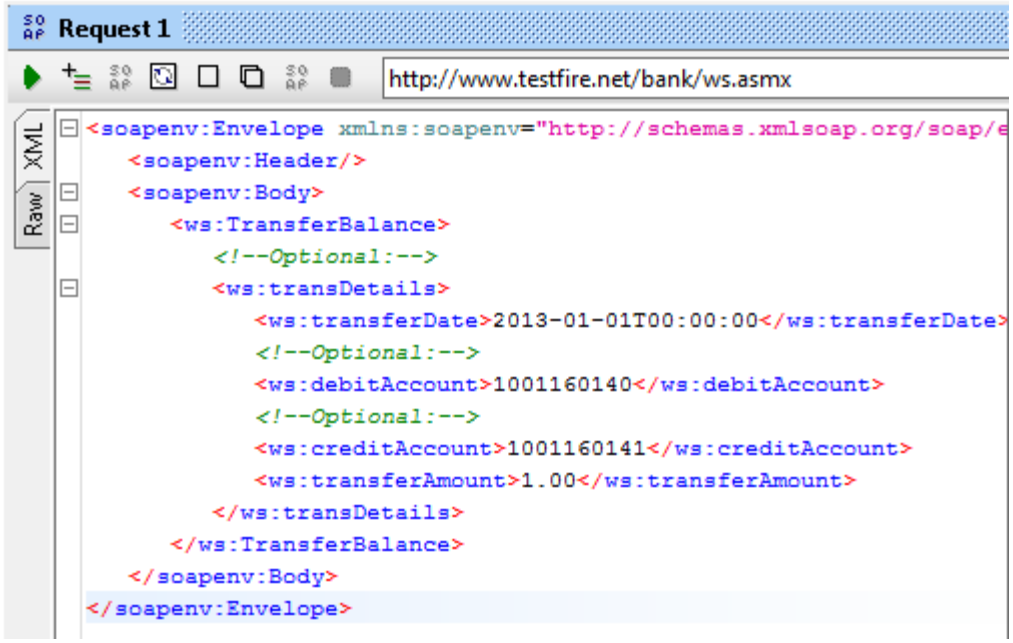


Figure 12: transferBalance request with data

Let's send this request by clicking the green button on the top of the left corner and we will get the result shown in Figure 13.

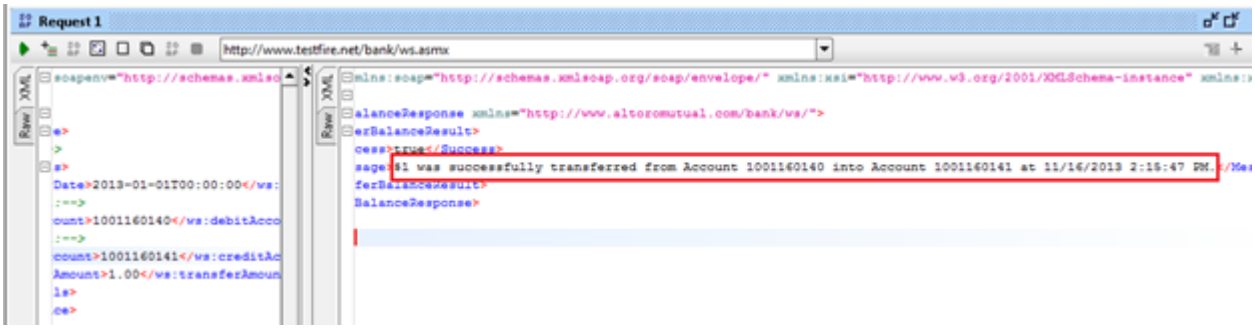


Figure 13: response window after successfully executing the request

Voila, we successfully sent the request and got the response that "$1 was successfully transferred from Account 1001160140 into Account 1001160141 at 11/16/2013 2:15:47 PM". Some might think how exactly I got these numbers correct; yes, this is a question but the other major question is how you can get all these data correct. It's simple by fuzzing the parameters.

We have four parameters here, of which two are independent, i.e., transferDate and transferAmount. And now we are left with two parameters. debitAccount and creditAccount. As there is no anti-automation we can fuzz both the parameters and enumerate the account number.

Now, since we successfully executed this request and collected some genuine data, we can use them to test some other test cases. Since it is a banking application and it allows us to send money from one account to another, let's check whether a negative balance transfer is allowed or not.

We will use the same data that we used in the previous request, just changing the transfer Amount value from 1.00 to -1.00. The data used:

1. transferDate(date-time) = 2013-01-01T00:00:00
2. debitAccount (string) = 1001160140

3. creditAccount (string) = 1001160141
4. transferAmount(double) =- 1.0

And let's check the result, as shown in Figure 14.

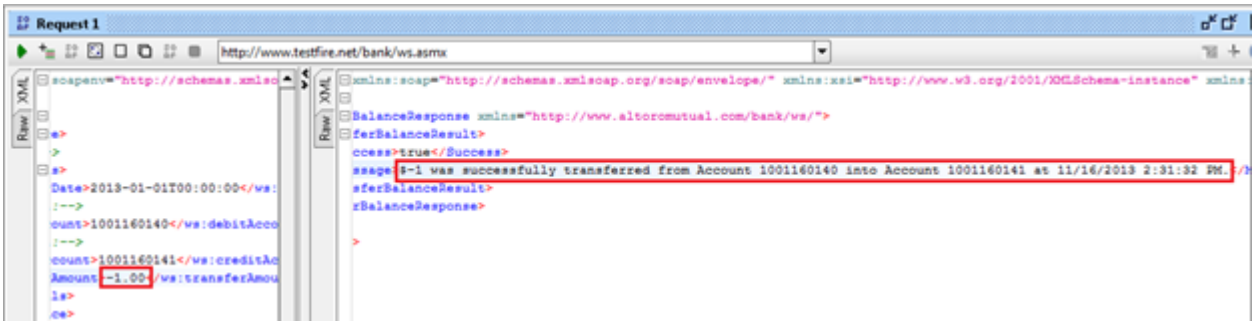


Figure 14: response window after successfully executing the request

Yes, we are able to transfer a negative balance. This is one of the critical business logic vulnerabilities in any banking application or service. And this is the power of manual testing. This kind of vulnerability can never be detected by any auto scanner.

Let's now look at another test case. We will use the same data used in the previous request, but this time with a $2.00 transferAmount. The only other thing we will change is that we will use the same account number for both debitAccount and creditAccount. Here we will check whether this web service validates the uniqueness of debitcardAccount and creditcardAccount before executing the service or not. The data used:

1. transferDate(date-time) = 2013-01-01T00:00:00
2. debitAccount (string) = 1001160140
3. creditAccount (string) = 1001160140
4. transferAmount(double) = 2.0

And let's check the result, as shown in Figure 15.

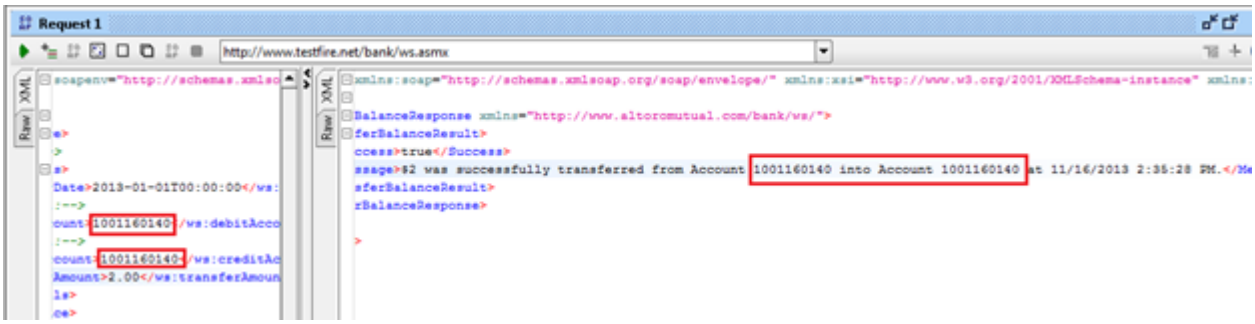


Figure 15: response window after successfully executing the request

This is again a critical business logic vulnerability. A banking service must validate that the sender's account and receiver's account are not the same.

Similarly, you can think out of the box according to your scenario; understand the functionality of the web service first and then experiment with different approaches to get new nontraditional vulnerabilities.

Now it's time to try some very popular test cases. The data we will use are:

1. transferDate(date-time) = 2013-01-01T00:00:00
2. debitAccount (string) = 1001160140'
3. creditAccount (string) = 1001160141
4. transferAmount(double) = 2.0

The result is shown in Figure 16.

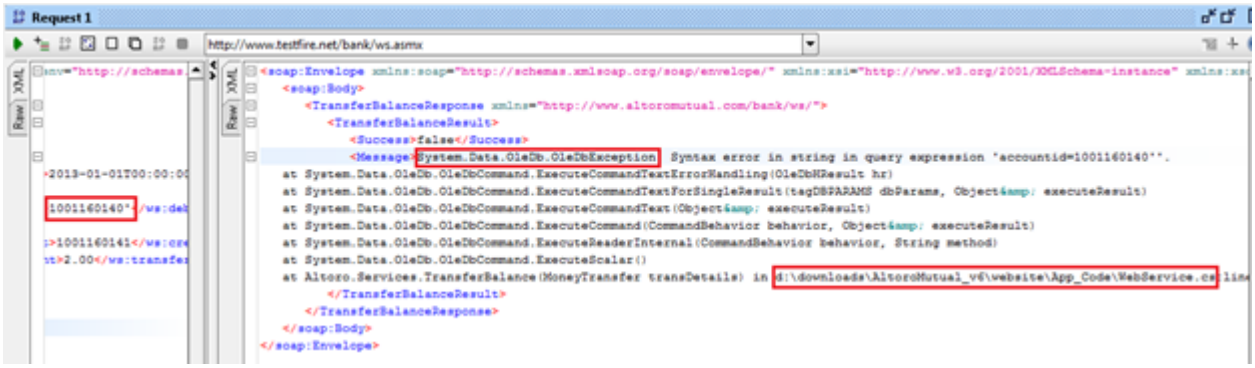


Figure 16: response window showing error message.
In this case we found:

1. System.Data.OleDb.OleDbException (we got a DB Exception or possible SQLI)
2. d:\downloads\AltoroMutual_v6\website\App_Code\WebService.cs:line 146 (server path disclosure)

Though we found some good business logic vulnerabilities, it's not possible every time to find the same. Most of the time, the common vulnerabilities that we find on a web service are information disclosures, such as header version disclosure, private IP address disclosure, database error pattern found, etc. Apart from that, authorization-related vulnerabilities are very common in this case. Some other common vulnerabilities, such as SQL Injection, can also be found. But we need to focus most on information disclosure, as web services are being used in complex enterprise level applications and those applications contain a huge number of vital, private and important information.

Conclusion

In this article, we focused on how to perform manual web services penetration testing using soapUI: How to get information regarding the data type needed for different parameters and how to generate different test cases according to the given scenario to test different business logic vulnerabilities. So, as I mentioned in the **Web Services Penetration Testing Part 1**, the testing approach of web services is quite similar to the testing approach used in web applications. You will agree with me after completing this article.

The only problem here is that it is very difficult to execute each and every request independently in soapUI. Let's say I want to fuzz a parameter: It is very difficult for me to do so just using soapUI. So, in the next installment, we will discuss how to integrate soapUI with other tools to automate the testing process.

Chapter 6

In the previous article we discussed in what cases we might face challenges performing manual web services penetration testing and how SoapUI will help in those circumstances. Now, what are the logical and business logic test cases when testing a web services, how do we test them, and what are limitations of SoapUI?

Though SoapUI is a very powerful tool while performing a manual Web services penetration testing, it does not allow a tester to fuzz a parameter. It's very important in case of a black box testing to fuzz. Let's take an example: if a Web service provides a login method, and you want to bypass the login method with SoapUI, you want to repeat the authentication request many times to brute force the credentials. But is it that easy with SoapUI? The answer is "NO". So that's why we will integrate SoapUI with other tools which provide us an interface to fuzz the parameters of a soap request generated by SoapUI. The tool we are going to use to perform the same is a very popular integrated platform to perform manual as well as automated testing: Burp Suite.

Burp Suite

Most security professionals use Burp Suite. It is a very popular tool to perform Web application penetration testing. It is an integrated platform for performing security testing of Web applications, and in most of the cases we can use the same to test Web services and mobile applications by proper configuration and integration with some other tools. Its various tools give you full control to enhance and automate the testing process.

Key Components of Burp

1. **Proxy** runs on port 8080 by default. It intercepts the request and let you inspect and modify traffic between your browser and the target application.
2. **Spider** is used for crawling content and functionality by auto submission of form values.
3. **Scanner** is used for automating the detection of numerous types of vulnerabilities. The type of scanning can be passive, active or user-directed.
4. **Intruder** can be used for various purposes, such as performing customized attacks, exploiting vulnerabilities, fuzzing different parameters, etc.
5. **Repeater** is used for manipulating and resending individual requests and to analyze the responses in all those different cases.
6. **Sequencer** is mainly used for checking the randomness of session tokens.
7. **Decoder** can be used for decoding and encoding different values of the parameters.
8. **Comparer** is used for performing a comparison between two requests, responses or any other form of data.
9. **Extender** allows you to easily write your own plugins to perform complex and highly customized tasks within Burp. Or you can also include different types of Burp extensions created by different developers or security professionals.

Burp Suite comes in two different editions.

1. Free Edition
2. Professional Edition

The major difference between these two is that in the Free Edition, some features like Scanner and Extender are not present. And also you can find that Intruder has certain limitations.

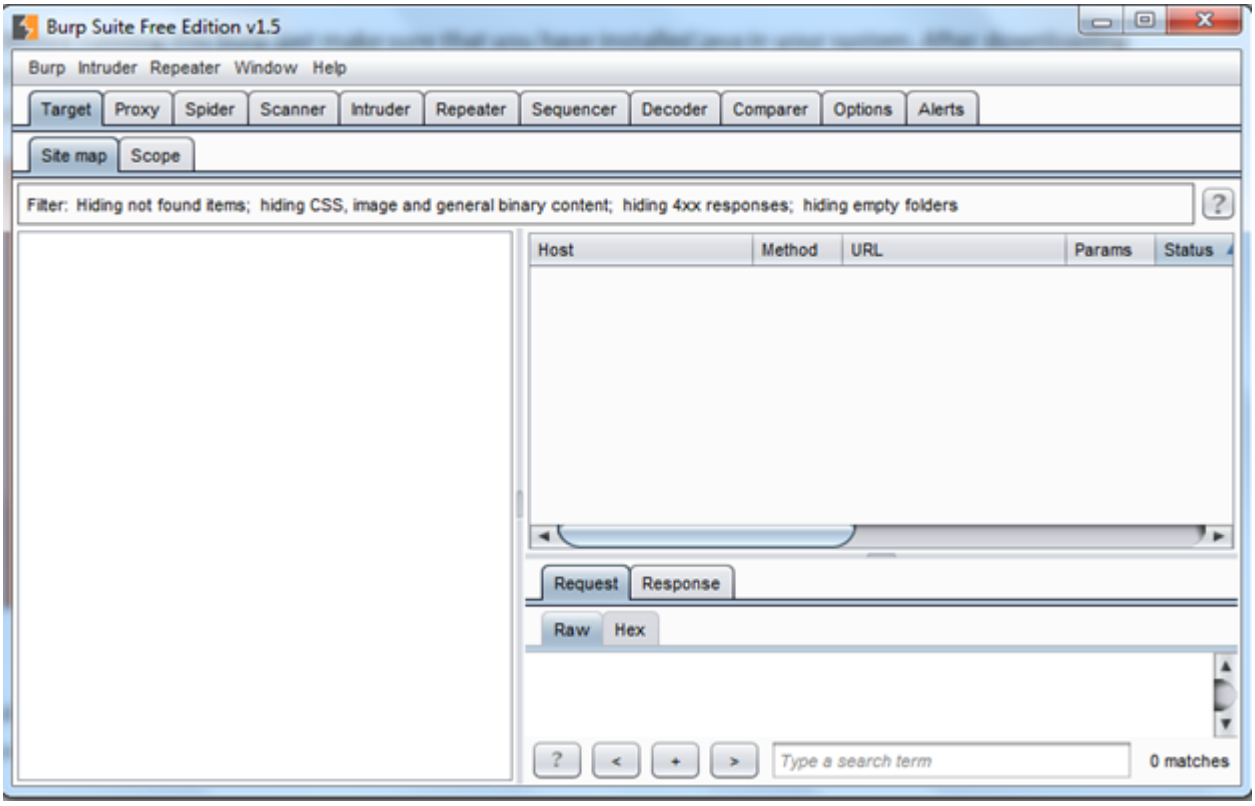
Though Burp Suite Professional Edition is one of the widely used tools for its unique features (which we will discuss in forthcoming articles), right now we will use Burp Suite Free Edition to fuzz different parameters of the request by integrating it with the SoapUI.

Burp Suite integration with SoapUI:

Burp Suite Free Edition is a fine product of Portswigger. You can download it from the below mentioned URL: http://portswigger.net/burp/downloadfree.html
Before running it, make sure that you have installed Java in your system. After downloading the Burp Suite Free Edition, you will get a jar file. Just click on that to open your Burp Suite. It will show as shown in Img1.

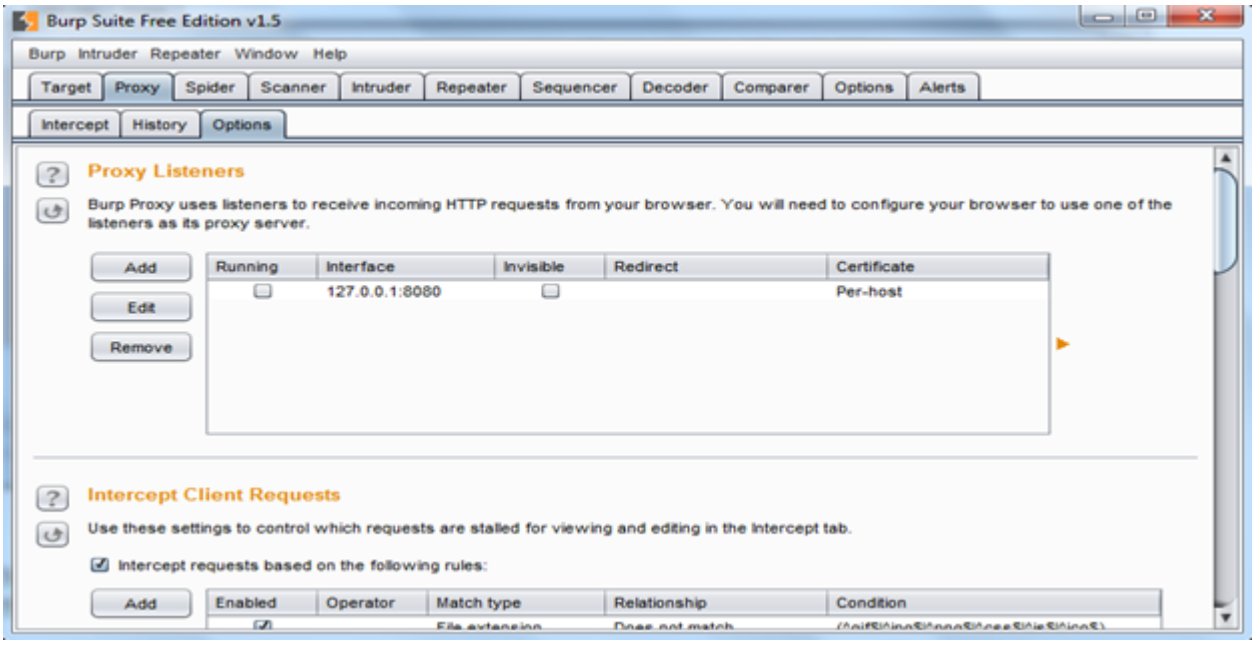
Img1: Burp Suite Free Edition starting banner

After this banner, you will find that your Burp is running properly, and with a little configuration it will be ready to fuzz parameters. The Burp window is shown in Img2.



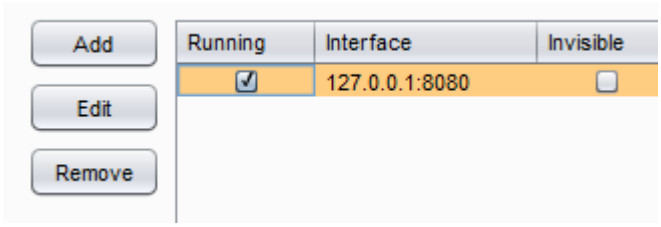
Img2: The burp window

For initial configuration, click on the Proxy tab on the top and then on the Options sub tab as shown in Img3.



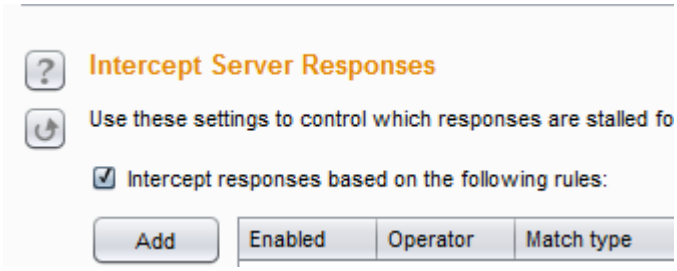
Img3: Options tab to configure settings

As shown in Img3, Burp Listen uses local host IP and 8080 port number by default. If you want to change the port or IP, click on Edit or just check the running checkbox to run the Listener as shown in Img4.



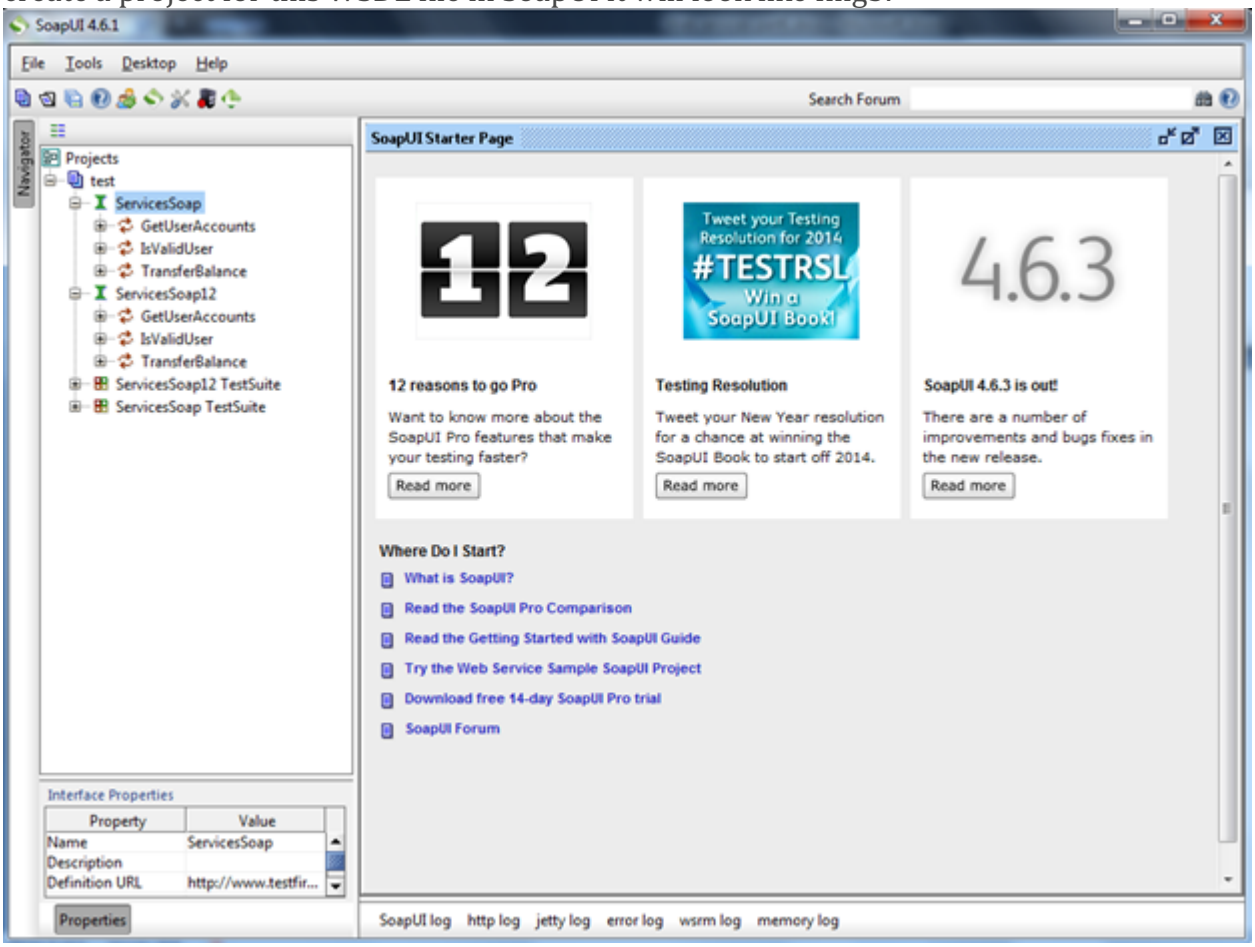
Img4: Start the Listener

We started the Listener but still we need to check a couple of things in the Options tab, such as Intercept Client Request and Intercept Server Response. By default, Intercept Client Request is enabled in Burp Suite Free Edition, and we can see that in the bottom part of Img3. But the Intercept Server Response is disabled by default, so we need to enable that as shown in Img5.



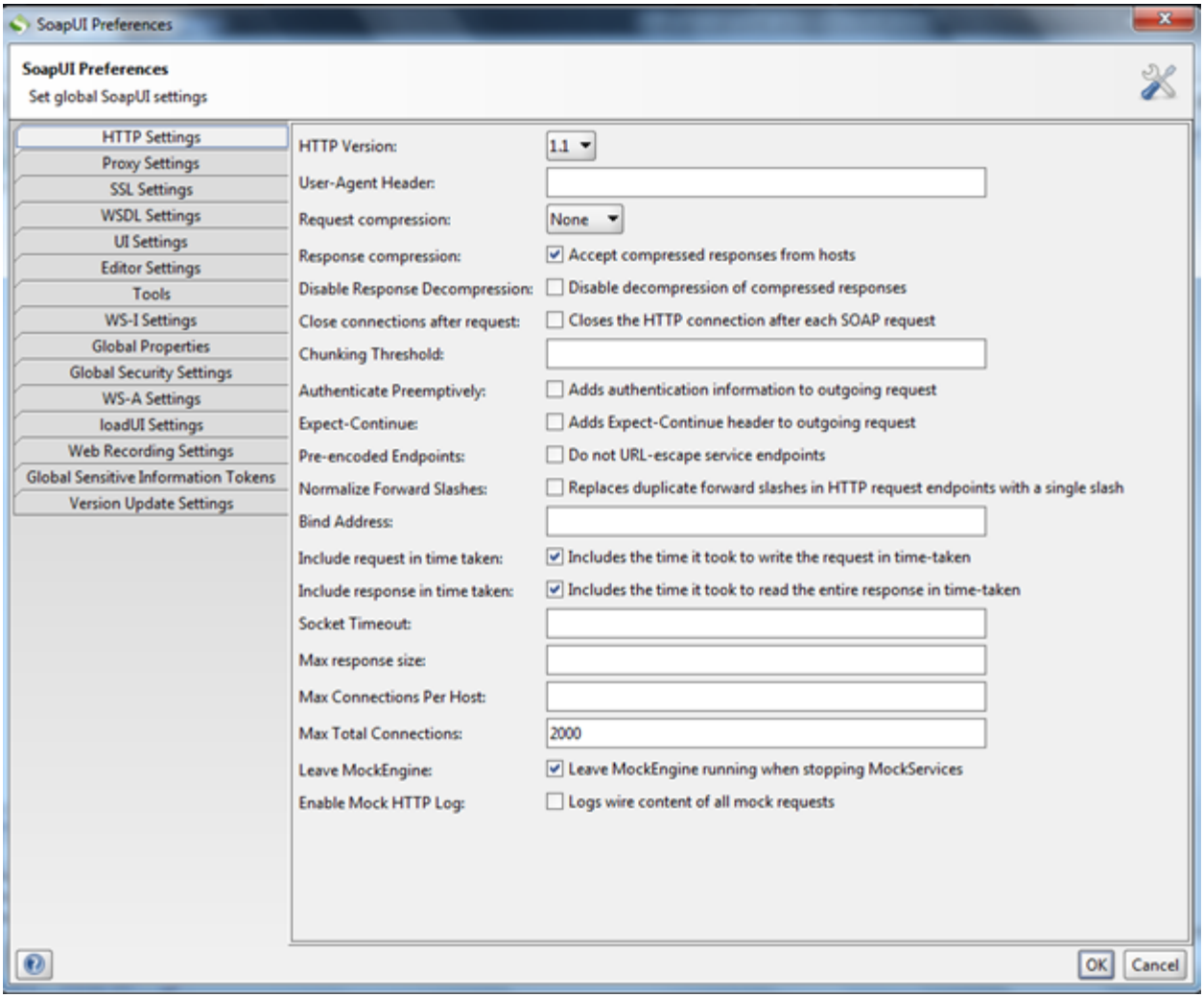
Img5: Enabling Intercept Server Response

Now Burp is ready to be integrated with the SoapUI. Open SoapUI. The initial creating project in SoapUI is discussed in the previous article. So I will not discuss the same again here. I will use the same URL http://www.testfire.net/bank/ws.asmx?WSDL to test Web services, and once you create a project for this WSDL file in SoapUI it will look like Img5.



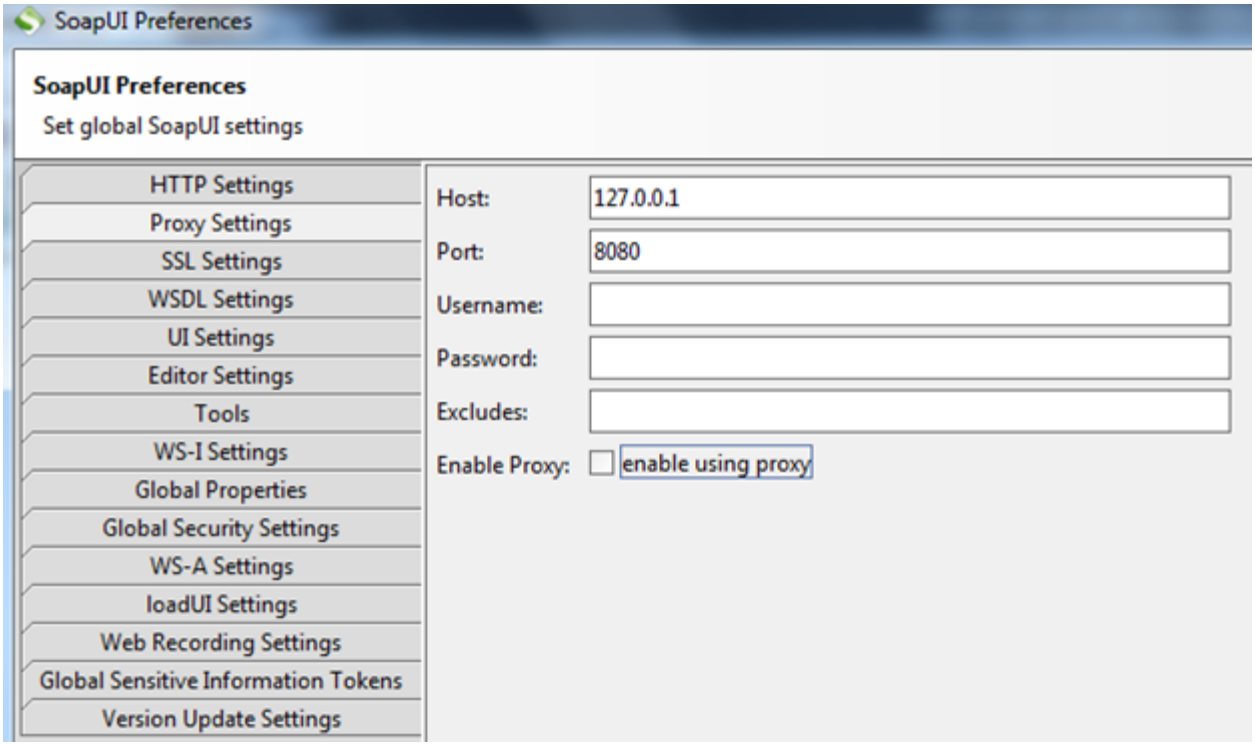
Img6: SoapUI window with all the methods of Web service

Now as everything is ready, to start with let's integrate the Burp Suite Free Edition with the SoapUI. To do so, you have to click on the Settings icon present in third position from right side in the left top corner on Img6, and you will get a new window as shown in Img7.



Img7: The settings window

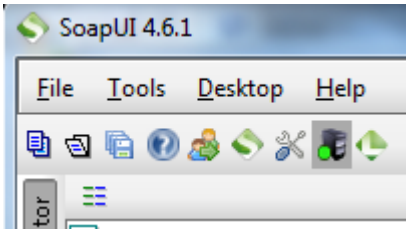
To change the proxy settings, click on the Proxy Settings tab, and another new window opens as shown in Img8.



Img8: Proxy Settings

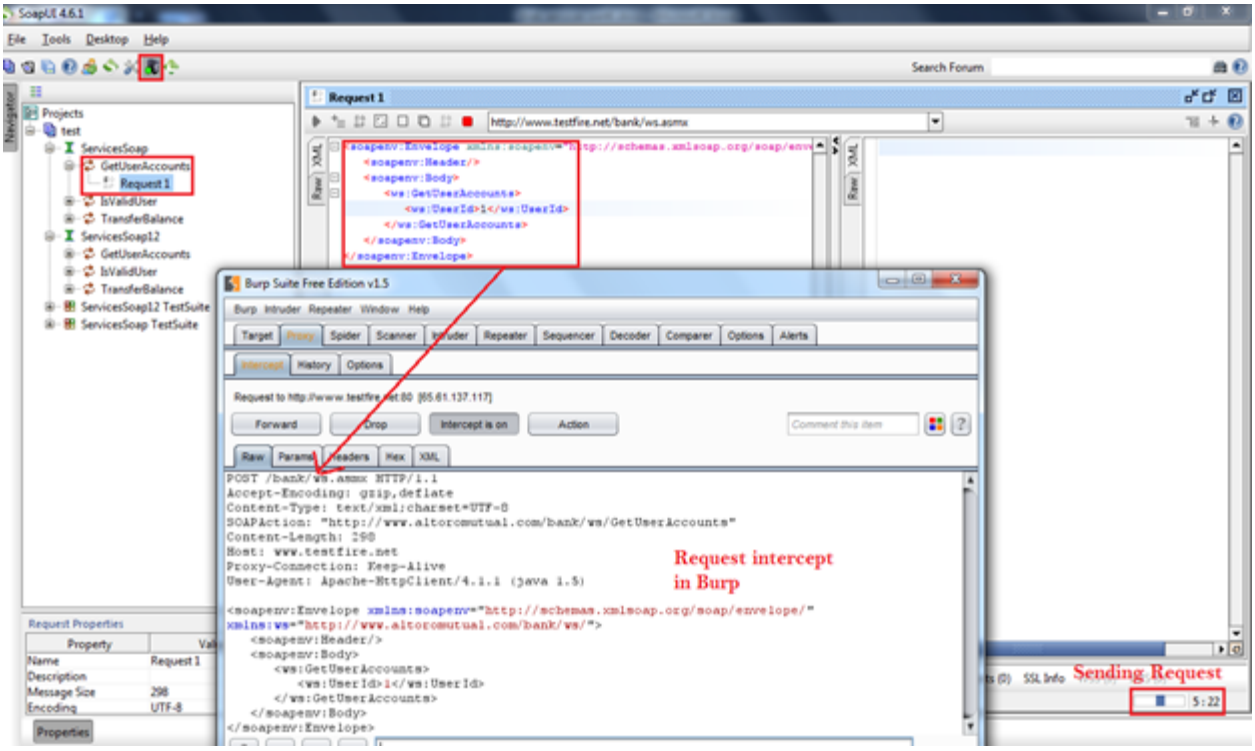
You can set your proxy listener IP and port number here, as in my case my proxy is running on 127.0.0.1:8080 I used the same. Then you can either enable using proxy from here or from the home page. Click OK and save the settings.

To enable the proxy, click on the proxy icon directly, this is present in the second position from right side in the left top corner of Img6. The icon in red means interception is off, and the icon in green means the interception is on. As shown in Img9.



Img9: Proxy icon

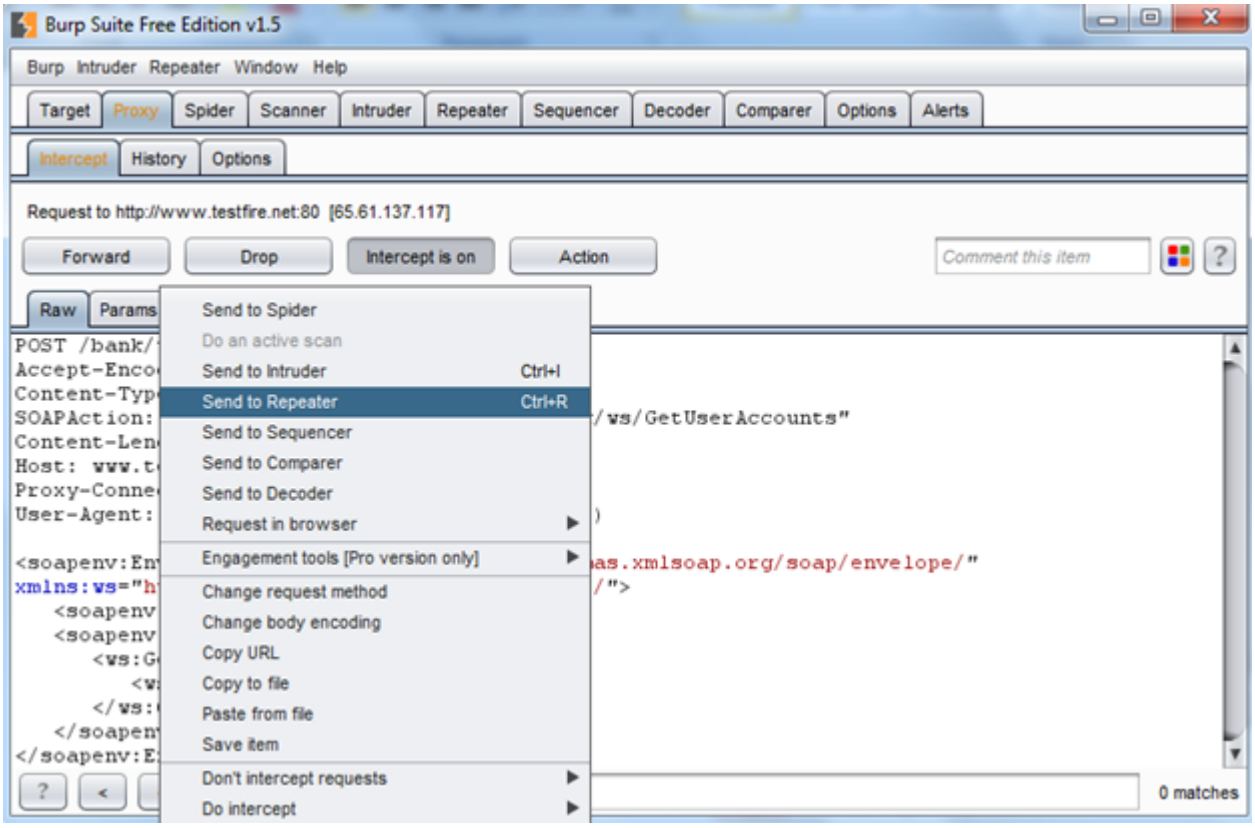
Now the integration part is complete. Just check your Burp proxy if the intercept is on or not, if not just make that on, then go to the SoapUI to send a request to check whether both are integrated properly or not. So I will use “GetUserAccounts” method in SoapUI and change the value of the parameter “UserId” to one to generate a proper soap request and check if the request is getting intercepted by the Burp Suite or not, as shown in Img10.



Img10: Burp Suite integration with SoapUI to intercept request

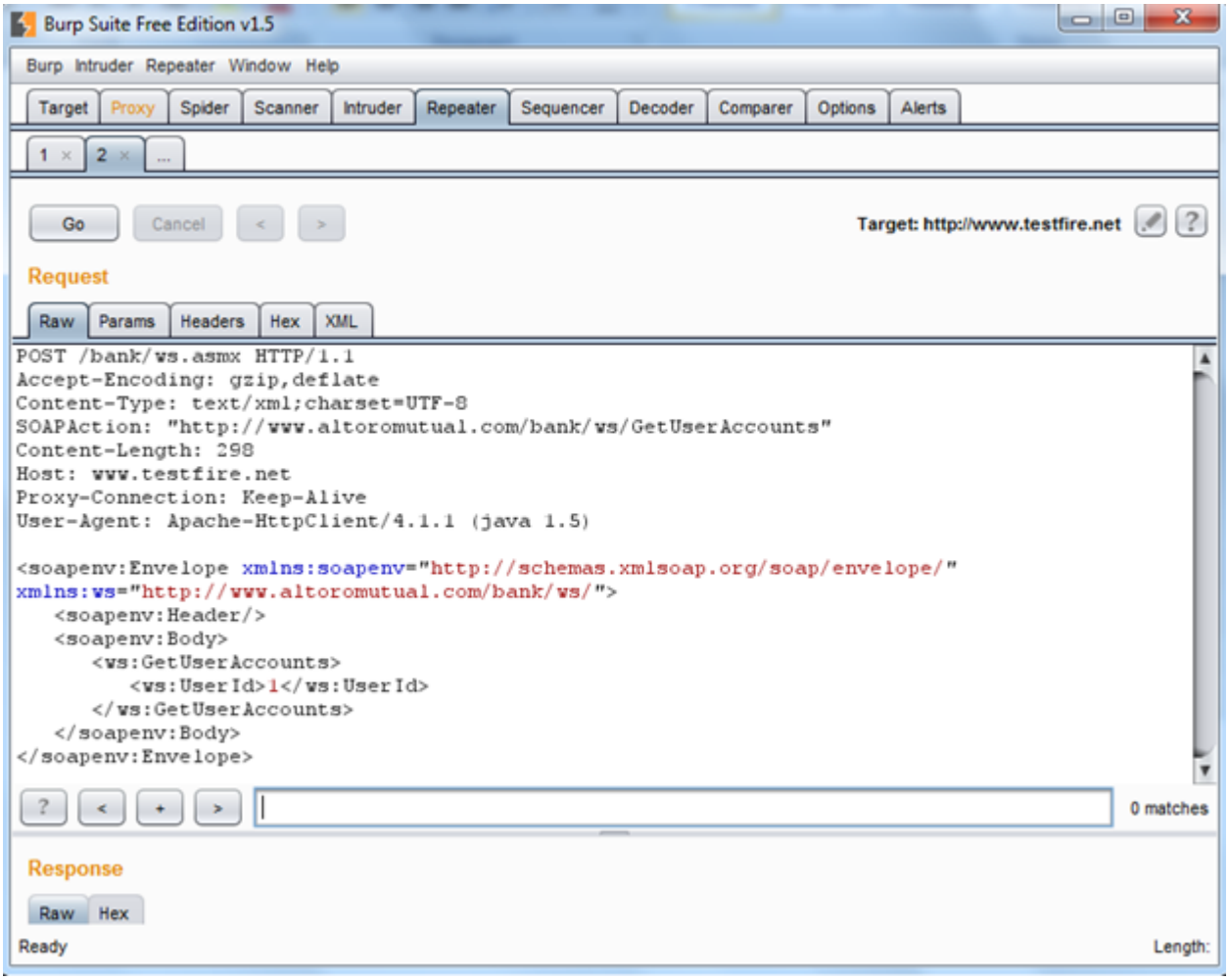
Testing the soap request with Burp Repeater

As we are able to integrate Burp Suite Free Edition with SoapUI successfully and able to intercept the request, now let's test for some test cases. First we need to send the request to the repeater as shown in Img11.



Img11: Sending request to Repeater

It's very simple, right click on the request and choose the option "Send to Repeater". Now go to the Repeater tab to check the response as shown in Img12.



Img12: Request in Burp repeater

Now send that request to get the response and to check whether we are getting anything special or not, as shown in Img13.

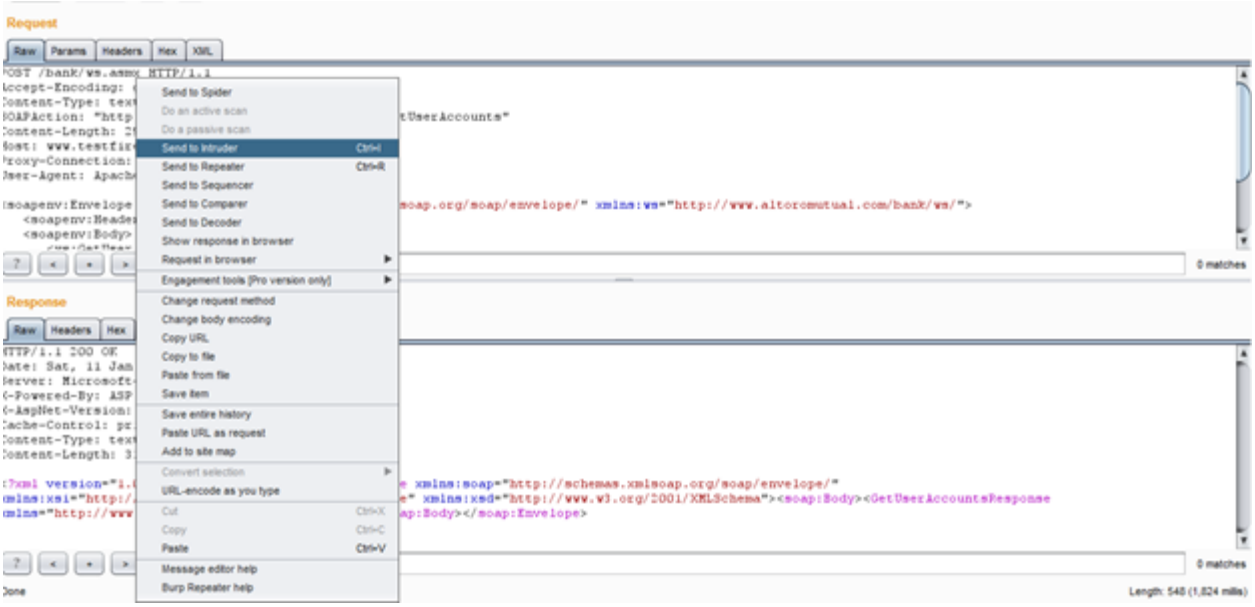
Img13: Soap Request and Response

As we can see in Img13, we have executed the soap request properly because we get a 200 OK response, but nothing interesting found in the body of the response. Though we get the version disclosed in the response header. What it states is that we used the proper data format; that's why it did not generated any error, so now we have to repeat the same process to get more information.

As the "UserId" is an integer type (As by providing an integer value it does not return any error, we can assume that it is an integer type parameter) or else by visiting the WSDL we can confirm it as shown in Img14.

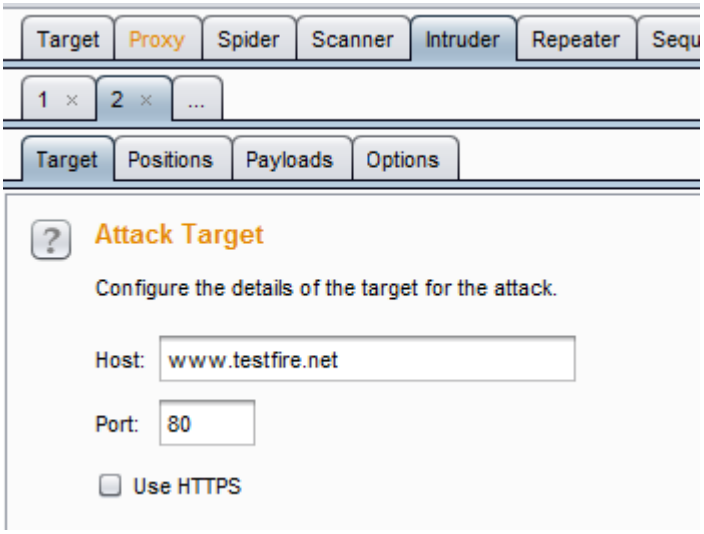
Img14: Data types of Request and Response

By looking at the WSDL file, we confirmed that the "UserId" parameter is an integer type parameter, and by providing a proper value we might get an array of account data. Now we need to fuzz the parameter to enumerate user data. To do so, send the request to the Intruder by right clicking on the request as shown in Img15.



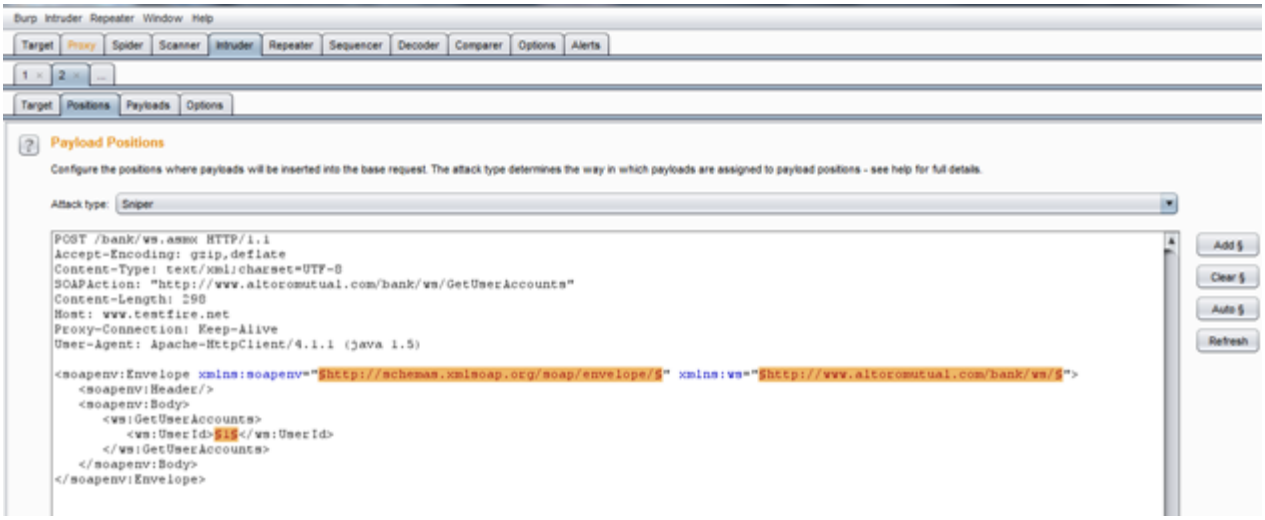
Img15: Sending request to intruder

By clicking on the Intruder tab in Burp, you will find following details as shown in Img16.



Img16: Target details of the request in intruder tab

Click on Position tab to select the proper position to fuzz or insert your payload as shown in Img17.

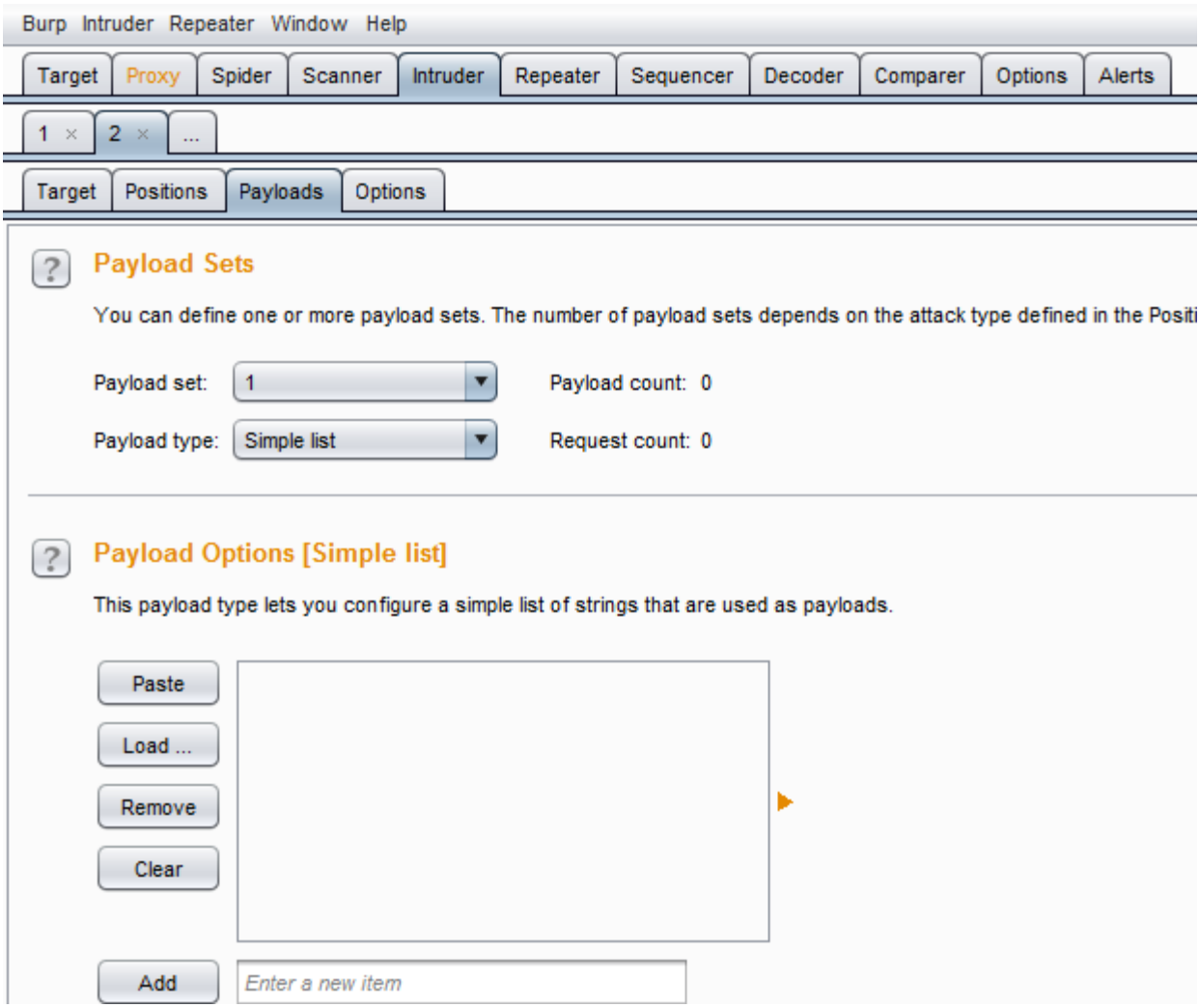


Img17: Positions to insert payload

The best thing about this Intruder is that it always auto selects all the possible insertion points present in a request. But in this case we only need to fuzz the “UserId” parameter, so now remove the other two selected options with the clear button present in the right side by selecting only those two as shown in Img18.

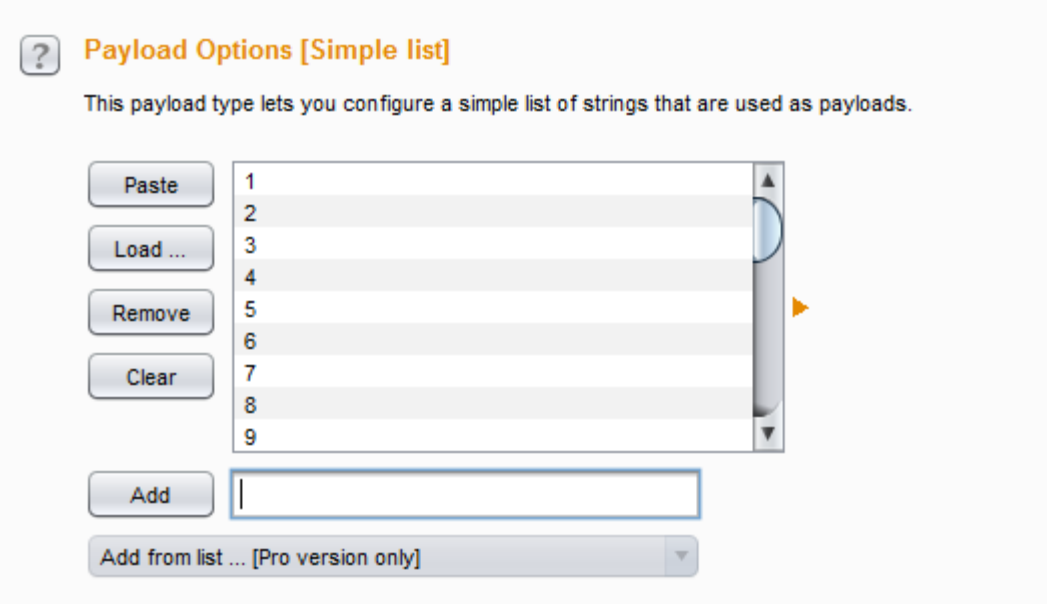
Img18: Selecting only the required parameter

Now move to the payload window to provide payloads as shown in Img19.



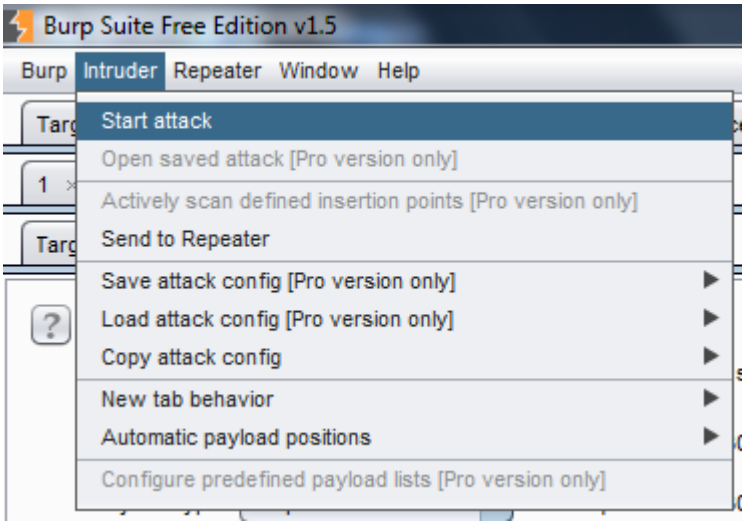
Img19: Payload window

We can specify the payloads manually by adding one by one, or we can create a list of payloads in a text file one below another and add it with the load option. Do whatever you want to do, but add some payloads as shown in Img20.



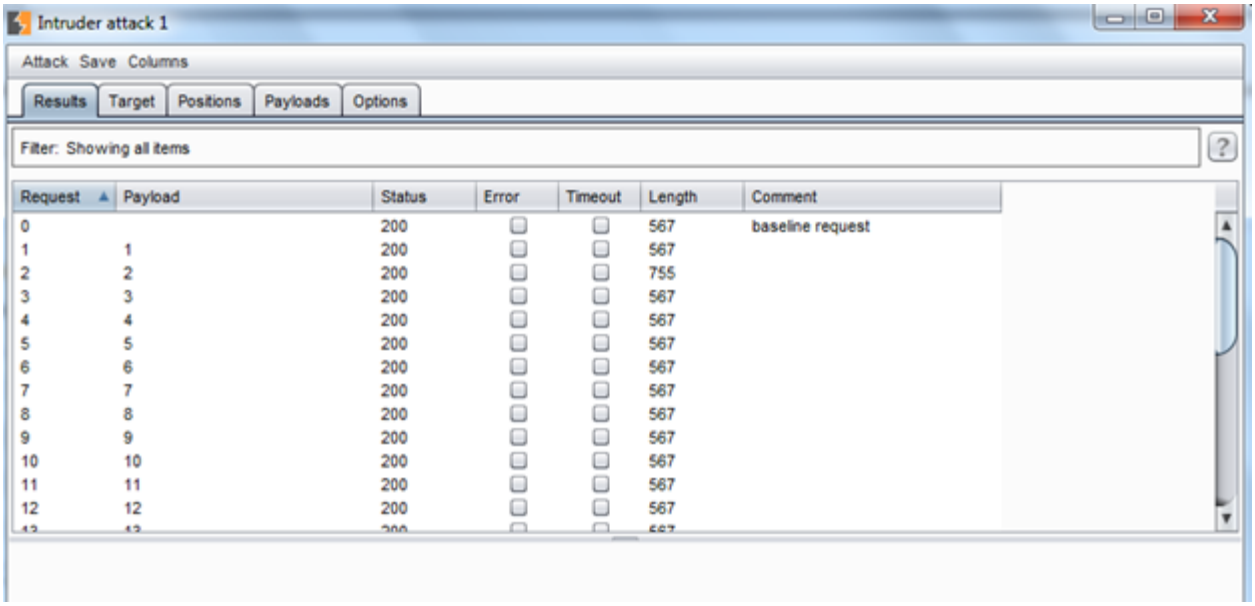
Img20: Specifying payloads

I inserted 1 to 30 to fuzz the "UserId" parameter and to check whether I am able to enumerate some user data or not. To do so, click on the Intruder menu in the top of your Burp Suite, and click on start attack as shown in Img21.



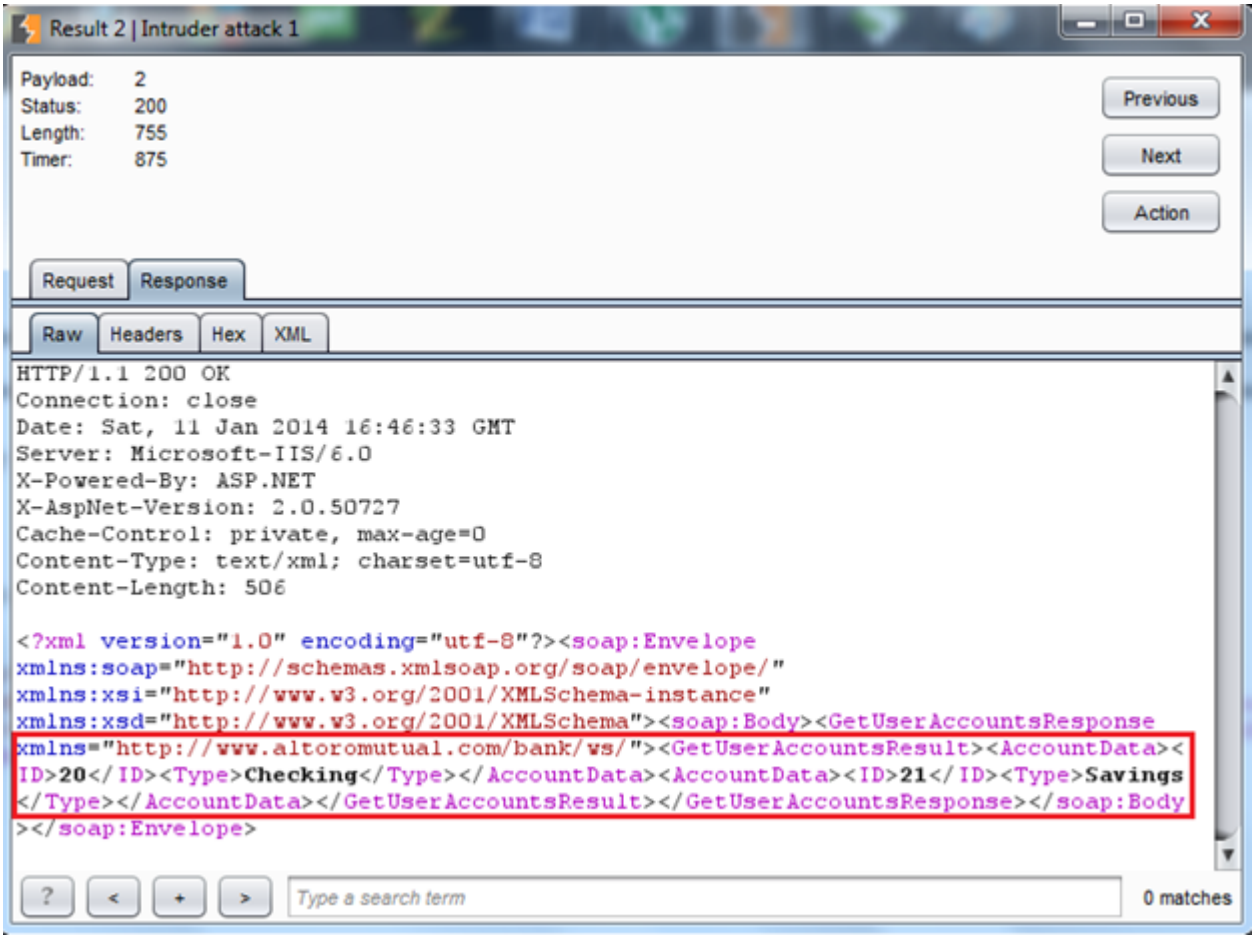
Img21: Start fuzzing with intruder

Now it will open an attack window to fuzz all the payloads in "UserId" parameter as shown in Img22.



Img22: Summery of Intruder attack

Now it is a very important part of the Intruder, you need not to go to each and every request to check the response. Just check in this window that if the status code or content length differs from all in any request, if yes then we are interested in that request. You can see clearly in the second request that the content length increased from 567 to 755. Double click there to open a new tab and click on the Response tab to check that data as shown in Img23.



Img23: The intruder response

You can see in Img23 that we enumerated certain user details. It is very critical information disclosed here. We got a vulnerability called “Sensitive information disclosure”. An attacker might use the information to create any other critical and sophisticated attack.

1. The user Id 2 is a valid user id.
2. It contains two accounts.
3. Account ID 20 is a checking account.
4. Account ID 21 is a saving account.

Conclusion:

We learned how to integrate Burp Suite Free Edition with SoapUI to fuzz different parameters of a soap request, how to configure Burp, and how to use different features like Burp Repeater and Intruder. And we also learned how to fuzz different parameters to collect different data or test some other test cases. In the next installment, we will cover how the sensitive information we got here leads us to other critical attacks and the challenges and limitations of Burp Suite Free Edition integration with the SoapUI tool.

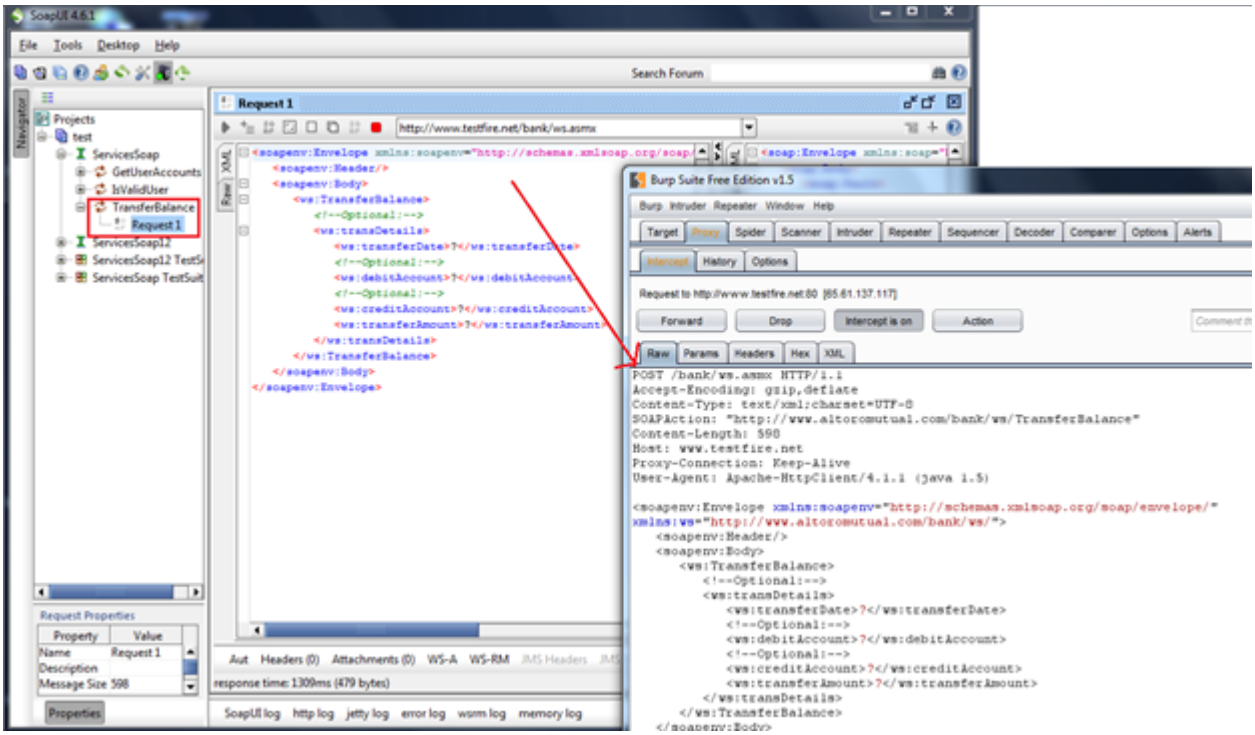
Chapter 7

In the previous article, we discussed how to integrate Burp Suite Free Edition with SoapUI to fuzz different parameters of a soap request, how to configure Burp, and how to use different features like Burp Repeater and Intruder. I assume that whoever is going through this article have that knowledge or if you don't have you just go through the previous article first.

We also learned how to fuzz different parameters to collect different data or test some other test cases, and we found some sensitive user information, such as:

1. The user Id 2 is a valid user ID.
2. It contains two accounts.
3. Account ID 20 is a checking account.
4. Account ID 21 is a savings account.

Now we will focus on how we can use this information for creating other attacks. I will continue from where I left in the last article. As we have gone through the "GetUserAccounts" method in the previous article, now we will check for the "TransferBalance" method. Let's generate a soap request from SoapUI and then intercept in Burp as shown in Img1.



Img1: Request intercept in Burp

Now if we see the request clearly, it contains four parameters, and we don't know the data type, as shown below in Img2.

POST /bank/ws.asmx HTTP/1.1

Accept-Encoding: gzip,deflate

Content-Type: text/xml;charset=UTF-8

SOAPAction: "http://www.altoromutual.com/bank/ws/TransferBalance"

Content-Length: 598

Host: www.testfire.net

Proxy-Connection: Keep-Alive

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

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ws="http://www.altoromutual.com/bank/ws/">

<soapenv:Header/>

<soapenv:Body>

<ws:TransferBalance>

<!--Optional:-->

<ws:transDetails>

<ws:transferDate>?</ws:transferDate>

<!--Optional:-->

<ws:debitAccount>?</ws:debitAccount>

<!--Optional:-->

<ws:creditAccount>?</ws:creditAccount>

<ws:transferAmount>?</ws:transferAmount>

</ws:transDetails>

</ws:TransferBalance>

</soapenv:Body>

</soapenv:Envelope>

Img2: “TransferBalance” soap request

So first we have to find the associated data types of those parameters from the WSDL file as shown in Img3.

Img3: WSDL file showing data type

Now as we know that:

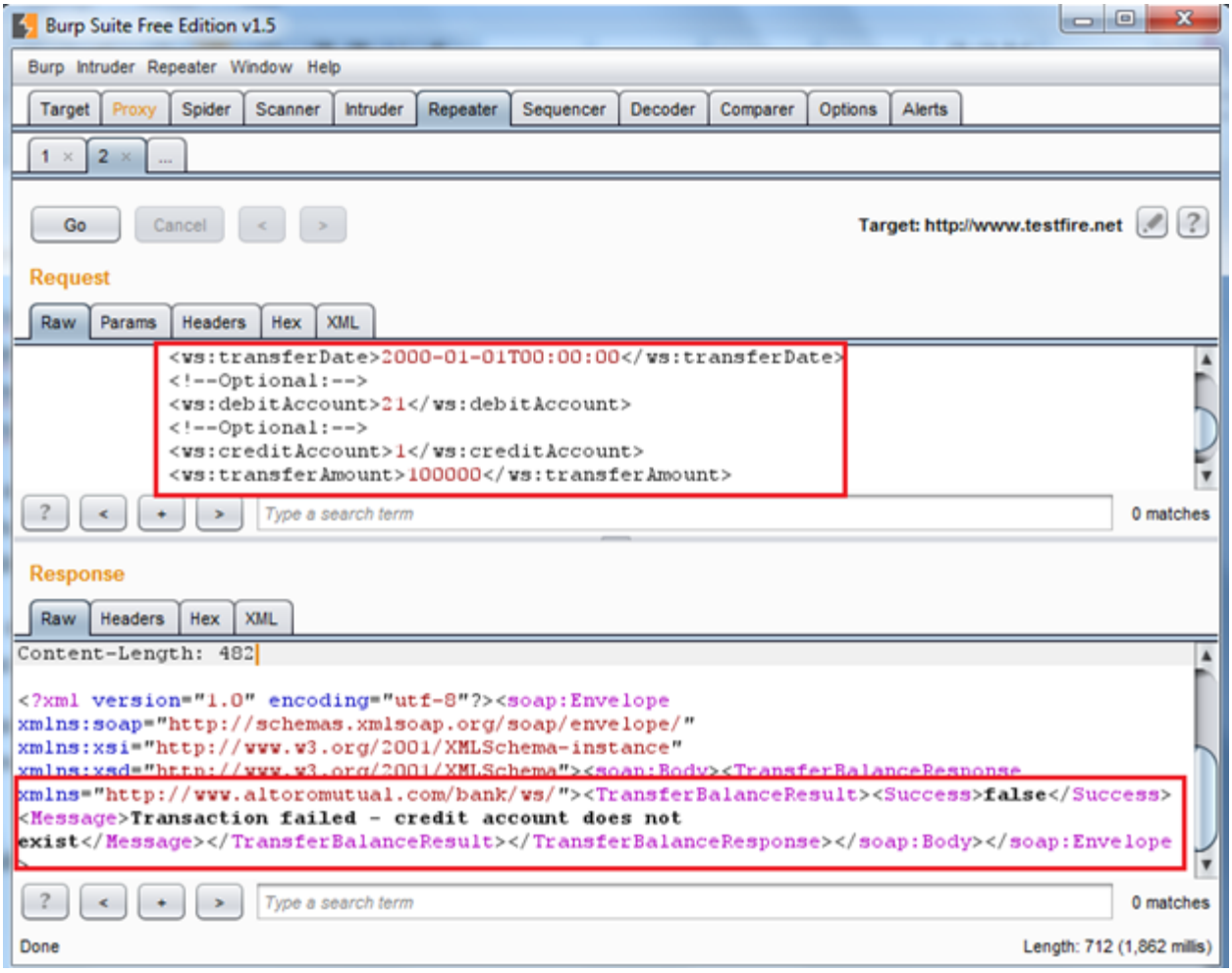
1. transferDate(dateTime) = 2000-01-01T00:00:00
2. debitAccount (string) = 21
3. creditAccount (string) = xx
4. transferAmount(double) = 100000

As this is a sensitive request and does not ask for any kind of authentication, an attacker (if he/she is an account holder of this bank) can easily transfer money from other user accounts. So it leads to a critical vulnerability of unauthorized money transfer.

Let’s say the attacker doesn’t have an account, but still he/she can transfer money from one account to another account. This is also another critical business logic vulnerability i.e. Anonymous money transfer.

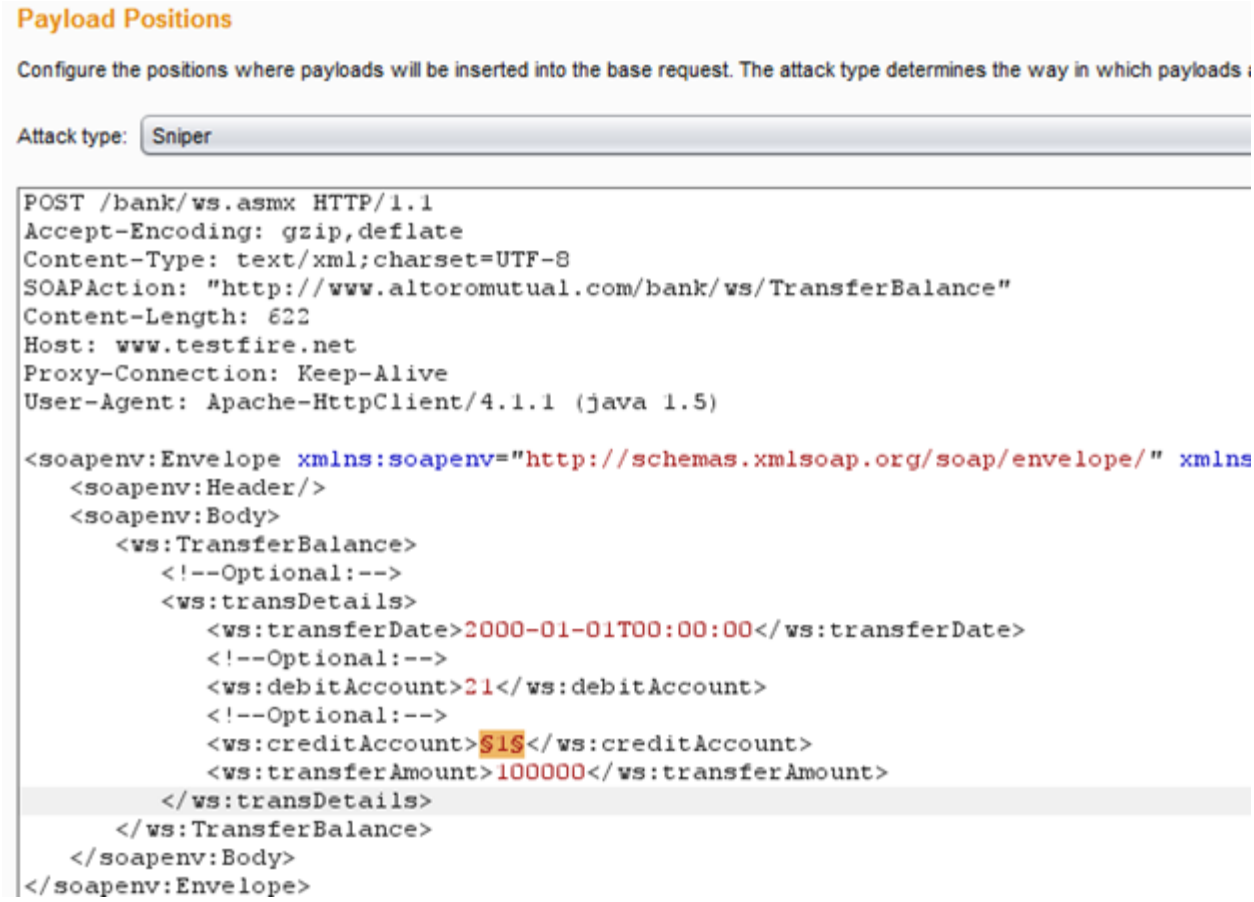
In this case, let’s assume I am an attacker and I don’t have an account in this bank, but I can easily transfer money from account 21 that we have got from the previous vulnerability to any other user’s account.

So let’s submit the request in Repeater, and check the response as shown in Img4.



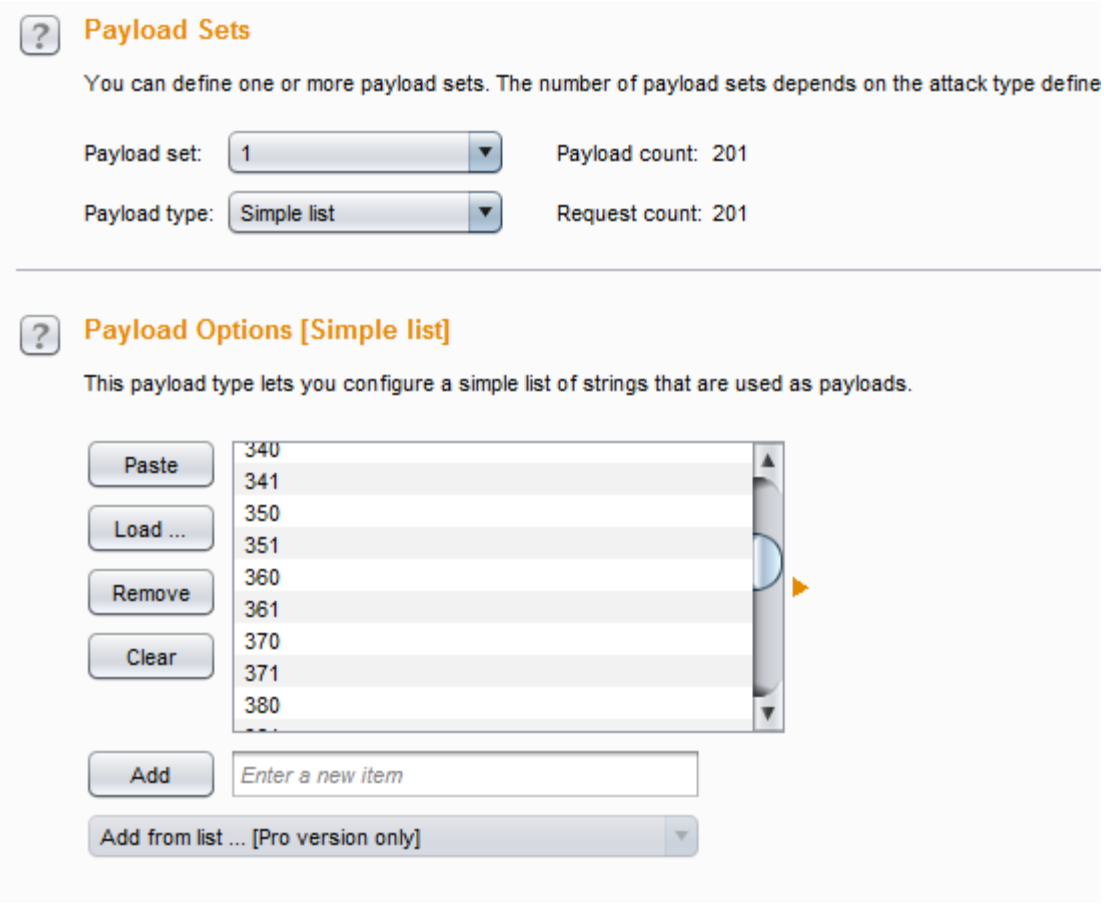
Img4: request response in burp repeater

Here we found another critical vulnerability: Account number enumeration. If we fuzz this "creditAccount" parameter we might able to transfer the amount. Send the request to intruder to fuzz this parameter. In the Positions section only select the "creditAccount" as shown in the Img5.



Img5: Selecting the insertion point

Now add all the payloads as shown in Img6.



Img6: Adding all payloads

Now start attack on intruder to check the responses as shown in Img7.

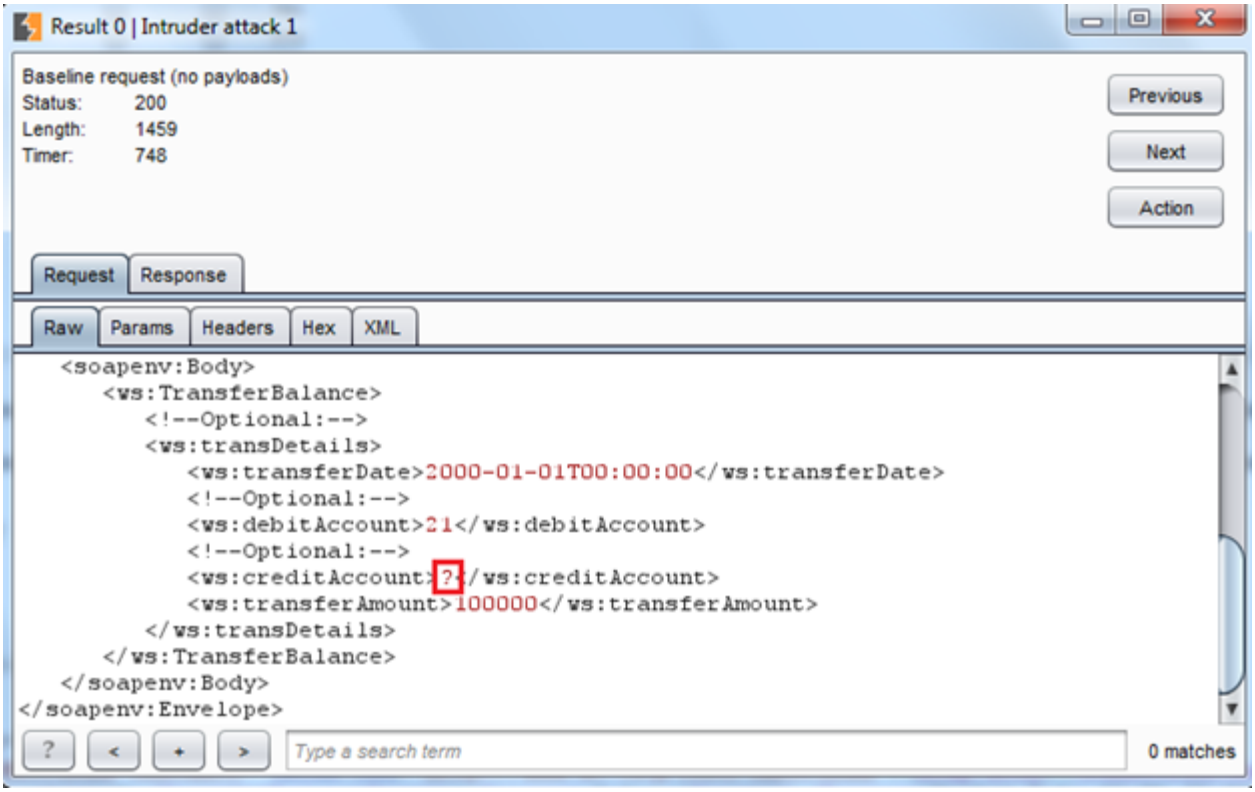
Filter: Showing all items

Request	Payload	Status	Error	Timeout	Length
0		200	☐	☐	1459
144	1001160140	200	☐	☐	781
3	20	200	☐	☐	773
4	21	200	☐	☐	773
1	10	200	☐	☐	731
2	11	200	☐	☐	731
5	30	200	☐	☐	731
6	31	200	☐	☐	731
7	40	200	☐	☐	731
8	41	200	☐	☐	731
9	50	200	☐	☐	731
10	51	200	☐	☐	731

Img7: Intruder attack result

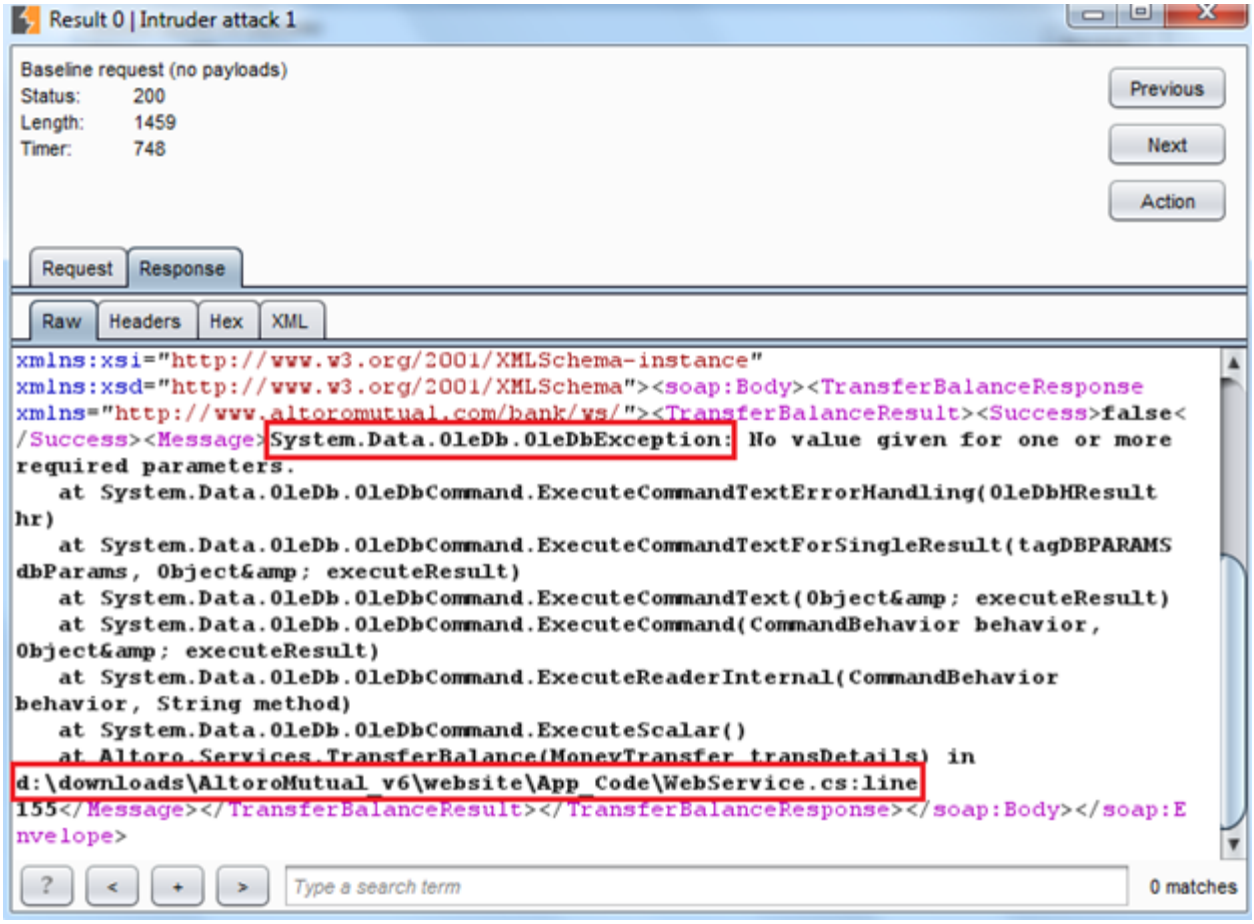
As we can see quite clearly, the first four requests according to the content length differ from other requests, so we are interested in those requests. Let's check one by one to find out what we have here.

The first request we will check is the base request and its response as shown in Img8.



Img8: The base request

The base request contains a special character in place of the “creditAccount” parameter, and we got the response as shown in Img9.



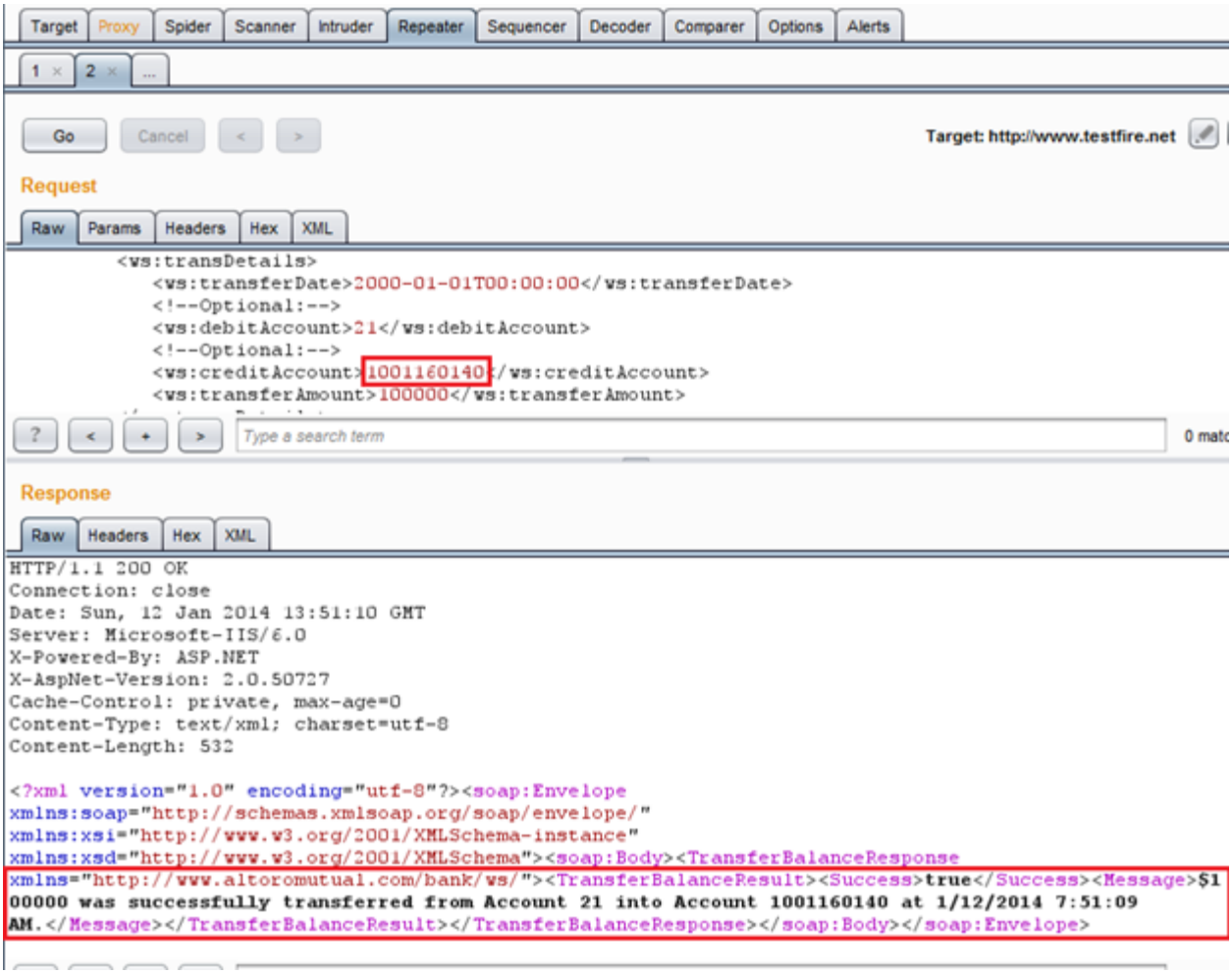
Img9: the base response

It contains server path and dberror pattern, so we have two vulnerabilities:

1. Server path disclosure

2. Possible SQLI

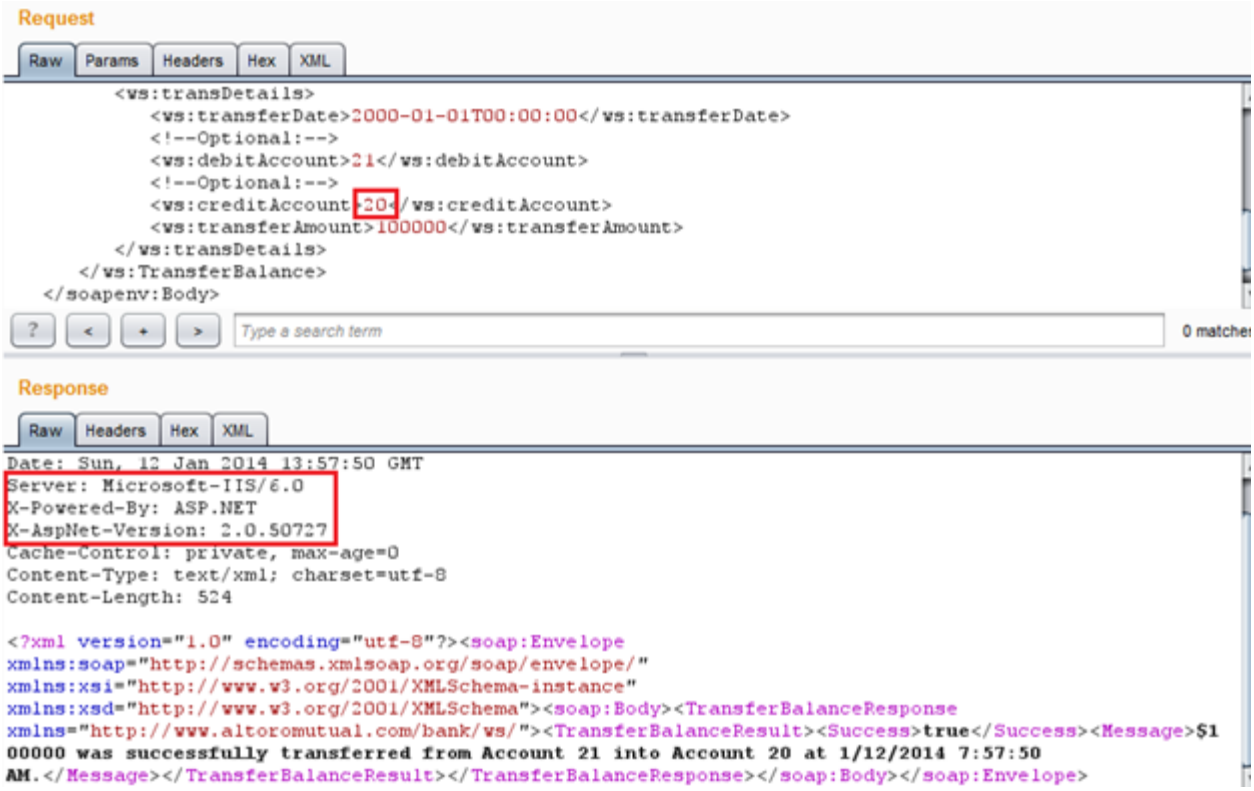
Similarly, let's check the next request and response. For easy understanding, I will send the request to the Repeater, where we can see both request and response in same page as shown in Img10.



Img10: Request response in Burp Repeater

As it shows that the transaction is successful, we successfully created an anonymous account money transfer. Though I don't have any credentials for either of these two accounts, still I am able to transfer money from one account to another. Other than that, we also enumerated other existing accounts of this bank by the account enumeration vulnerability.

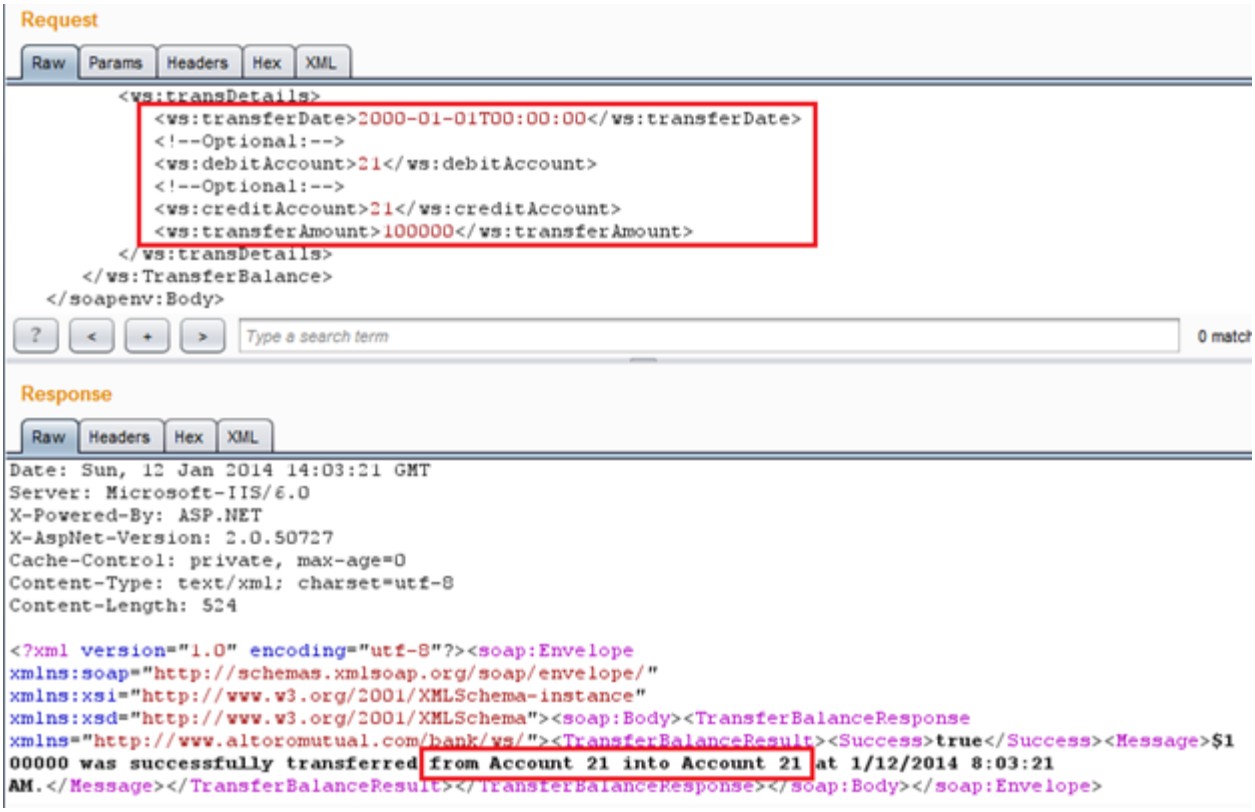
Let check the next request and response as shown in Img11.



Img11: Request response in Burp Repeater

Here, if we remember in the previous article, we enumerated some user account information, and according to that account no 20 and 21 belong to same userid=2. So here, nothing interesting apart from version disclosure is found because it's a genuine request; a user can transfer money from one of his/her accounts to another.

Now let's check the last request as shown in Img12.



Img12: Request response in Burp Repeater

We found vulnerability here that same account transfer. Generally, it's business logic vulnerability. The sender's account and receiver's account must not be same in any cases.

Though in one of the previous articles I already demonstrated some of these test cases, here I again tested all those test cases from scratch. How to collect various information using fuzzing, and how to use it in other attacks, are demonstrated here in detail. It's not that in black box testing you are restricted to only some vulnerability like information disclosure or authentication bypass, it's always a challenge to do much more in a black box test that any one should ever imagine.

Challenge 1:

Some time you might face some of the problems that I have faced while integrating Burp with SoapUI. The first one is that if you have integrated your Burp with SoapUI and you want to test some web services which run on port 443. You might not get the request in your Burp if you just follow the configuration shown in the previous article. You need to make certain changes in your configuration.

Burp integration with SoapUI for HTTPS:

If you are using SoapUI for a site let's say https://www.example.com/main.aspx?WSDLthen you can easily test it in SoapUI, but when you want to integrate it with Burp, you won't get the request intercepted at the Burp end. The root cause of the problem is that it seems SoapUI ignores the proxy settings if the target service runs on HTTPS. So, what to do?

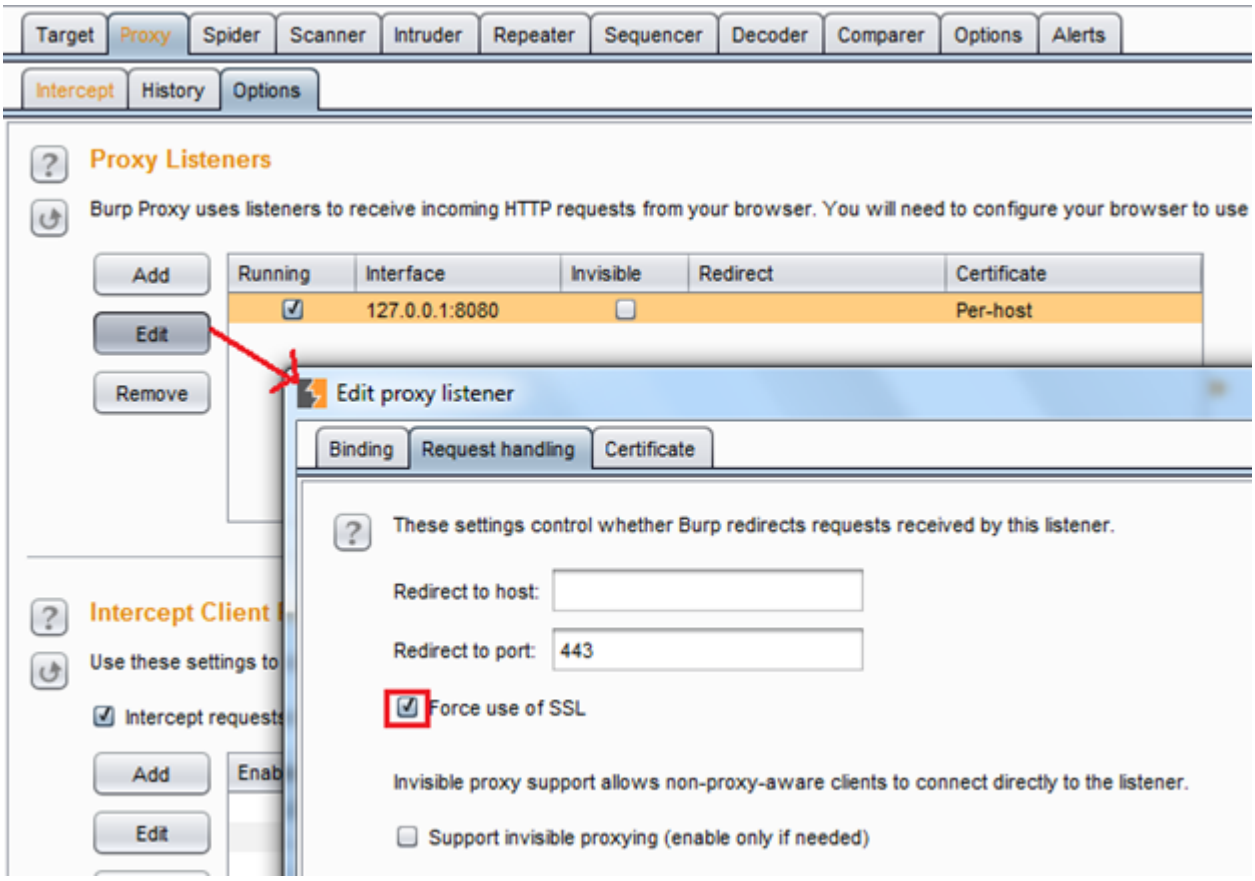
Solution:

Open any request in SoapUI. In the top of that, you can find the endpoint URL. Edit the endpoint there as shown in Img13.

Img13: Edited endpoint

You might ask why to edit the endpoint every time, why not edit the project while fetching the methods of the WSDL file while creating the project. Yes, it can be done, but sometimes SoapUI won't support this, and you might not be able to send a request response from the SoapUI.

So change the endpoint every time in each request you want to forward to Burp. This configuration edit is done in the SoapUI side; similarly, we have to change the configuration in the Burp options as shown in Img14.



Img14: Burp force SSLsetting

First click on the Proxy tab, the click on the Options sub tab. Then select the proxy listener you are using. Then click on Edit. Then a new edit proxy listener window will open. Click on the Request handling tab. Check the force use of SSL and select OK.

After this configuration, you will able to intercept all the requests coming from SoapUI for the port 443 endpoint, because with this solution the SoapUI proxy will properly work and the requests will be sent to Burp without SSL, but Burp will force the SSL with the server, so for the server everything will be the same.

Challenge 2:

Apart from the form-based authentication, if any web services use BASIC or NTLM authentication, you might not able to create a project in SoapUI directly, or in some cases you won't able to invoke a request properly. First let's go for Basic Authentication. In Basic Authentication, username/password credentials are from Windows Active Directory domain and need to be sent in the HTTP header. So, how to authenticate a request in SoapUI?

Solution:

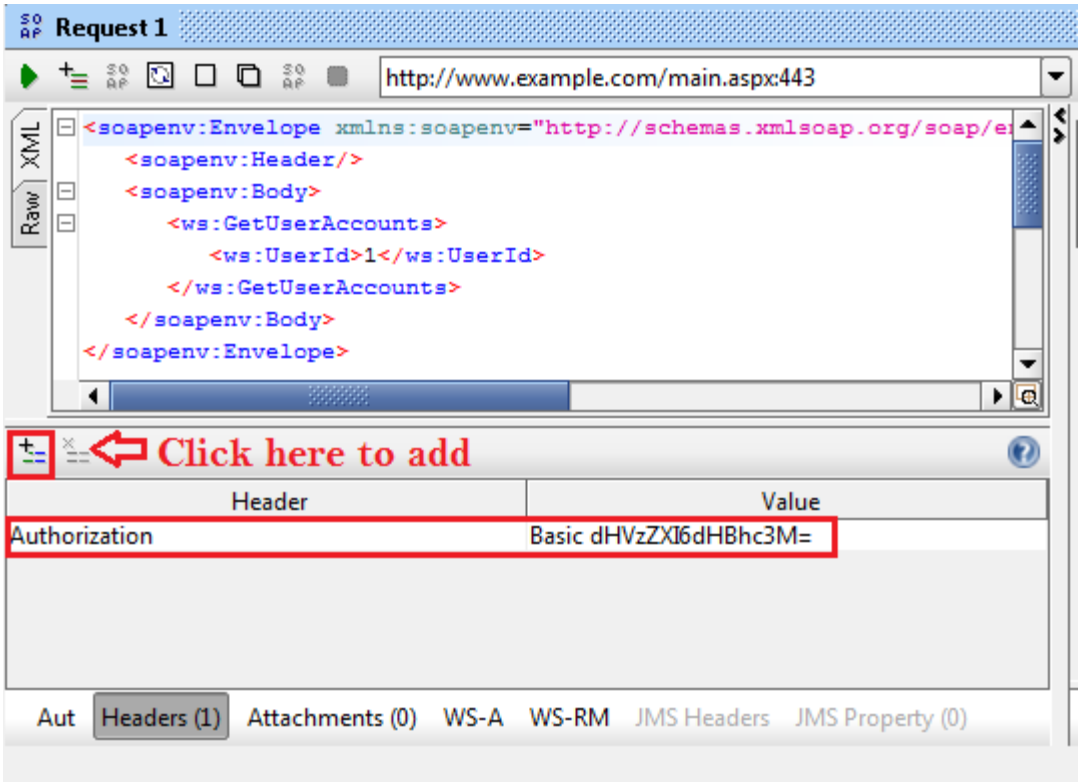
Let's assume that you created a project for a Web service which asks for Basic Authentication. Now when you are sending a request it generates an error that you are not authorized. There are basically two ways to solve this problem.

Solution 1:

When you open a request generated by SoapUI, at the bottom you will find some tabs named Aut, Headers, Attachments, WS-A, WS-RM, JMS Headers and JMS Properties as shown in Img15.

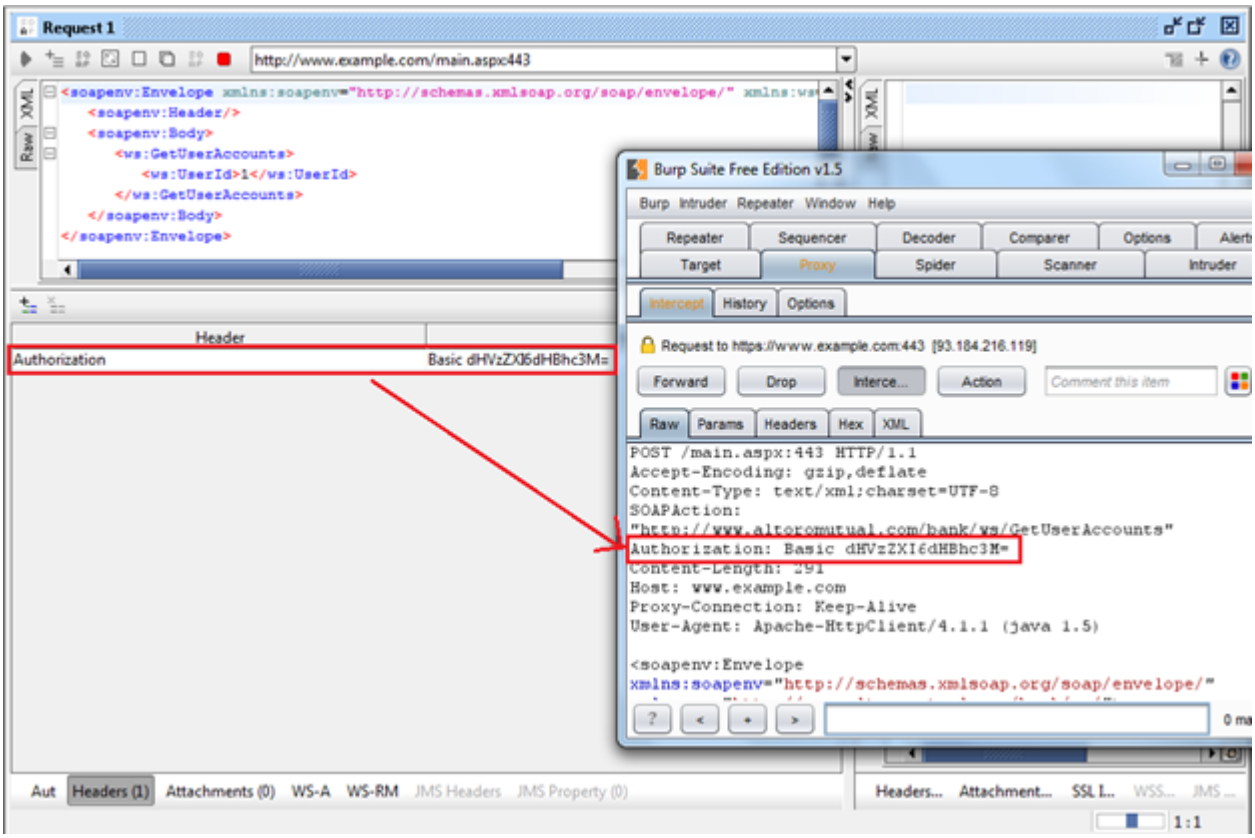
Img15: Different tabs of the Request window

When you click on the Headers tab, it will open a small window, which allows you to add any headers to the message. There you can add a header name "Authorization" and its values should be "Basic <Base64 encoded username:password>". If your username and password are tuser:tpass, then when you encode it, your header values will look like "Basic dHVzZXI6dHBhc3M= " as shown in Img16.



Img16: Adding Authorization Header

After setting this, if you send a request in SoapUI, you will be able to see the Authorization header in the request. So, let's send the request and check in Burp whether we are getting the Authorization header or not as shown in Img17.

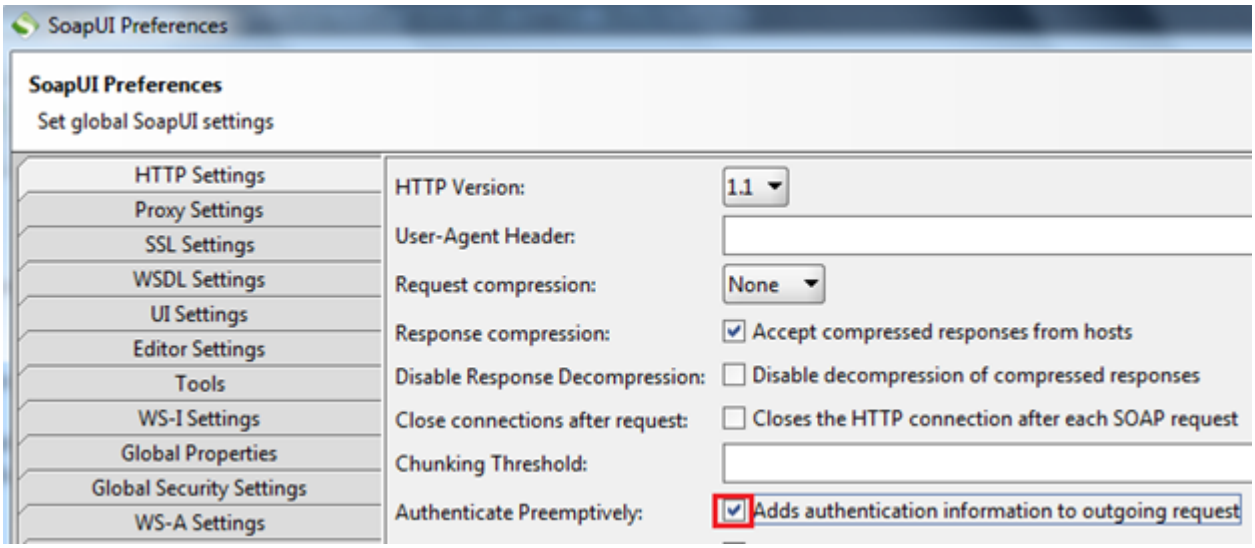


Img17: Request with Authorization Header

Solution 2:

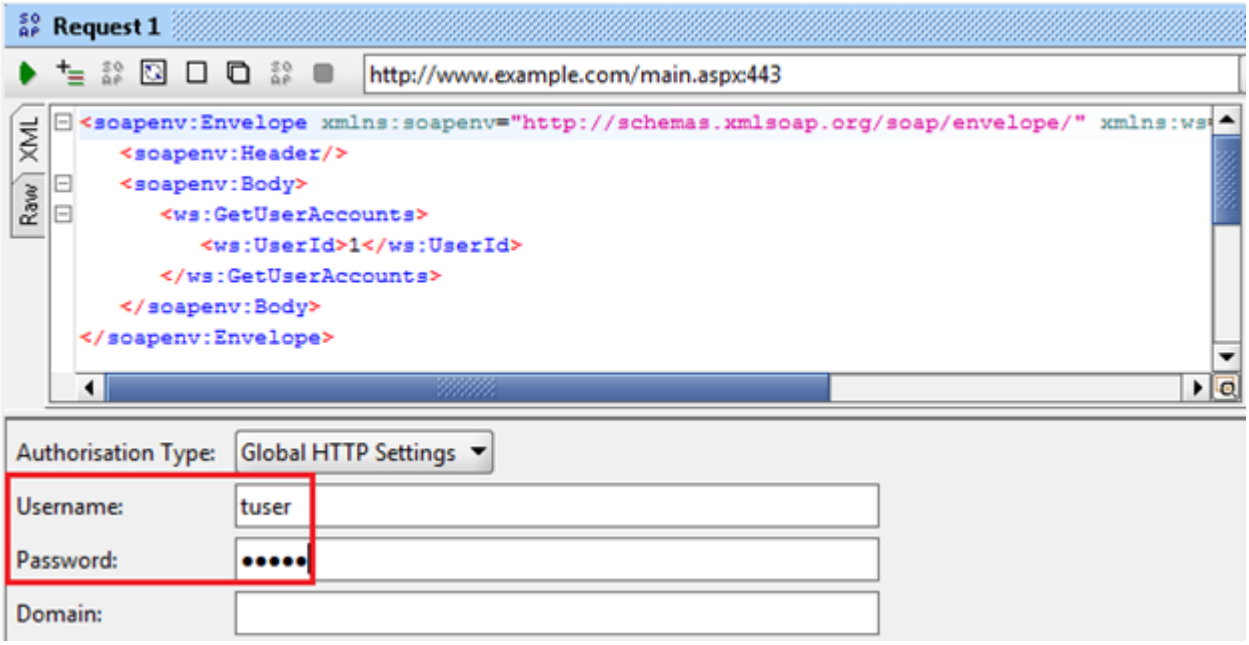
In this case we will use the Aut tab and define the username and password. But, this information will not get sent as a header. To send them as a header, you need to do the following.

Click on the settings icon, which is present on the third from the right side on top of the SoapUI window. Then select "HTTP settings tab". From that settings list, check the "Authenticate Preemptively" option as shown in Img18.



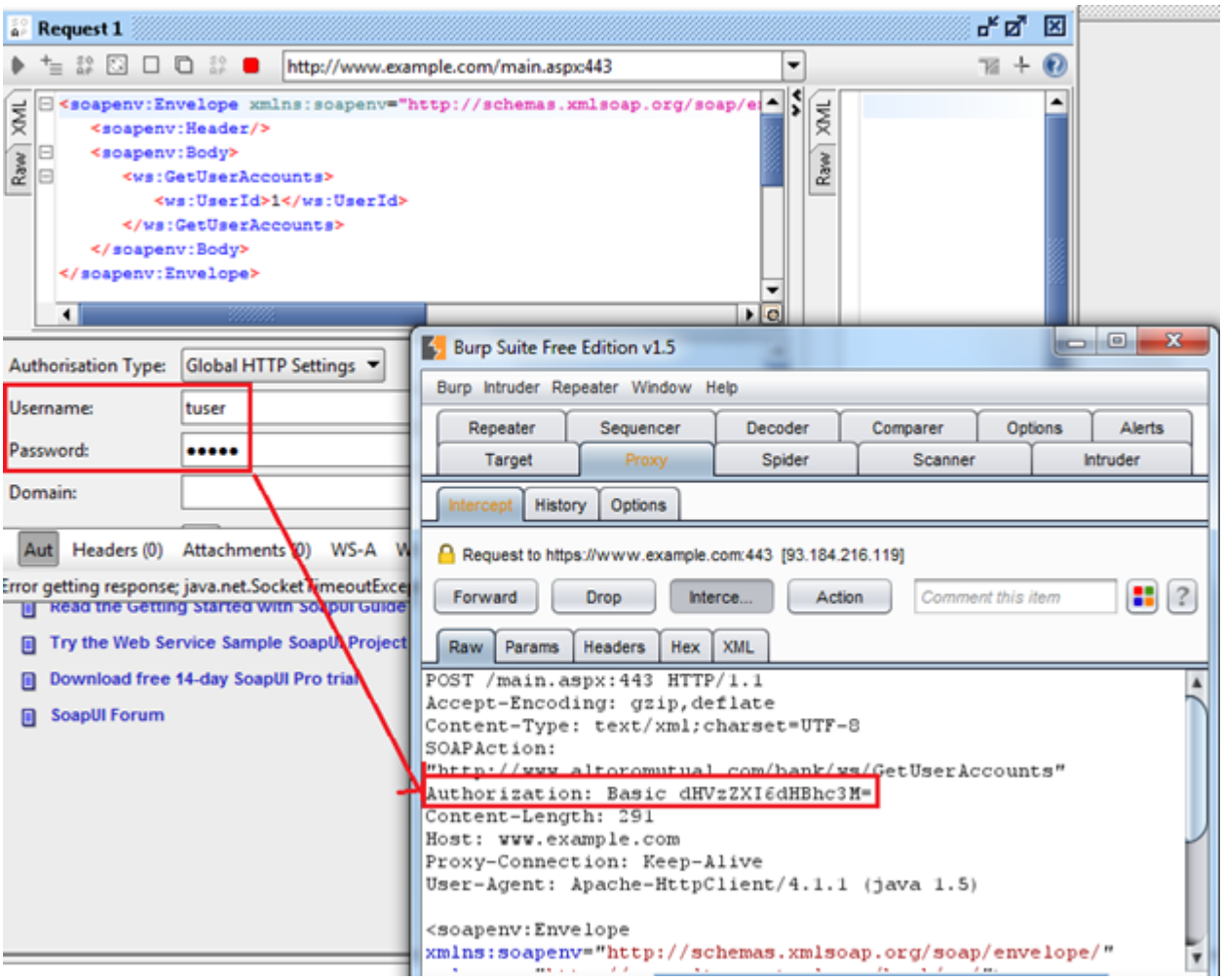
Img18: Changing SoapUI settings

Then, set the username and password in the Auth tab as shown in Img19.



Img19: Adding username and password in the Aut tab

Now you will see the Authorization header sent with the request as shown in Img20.



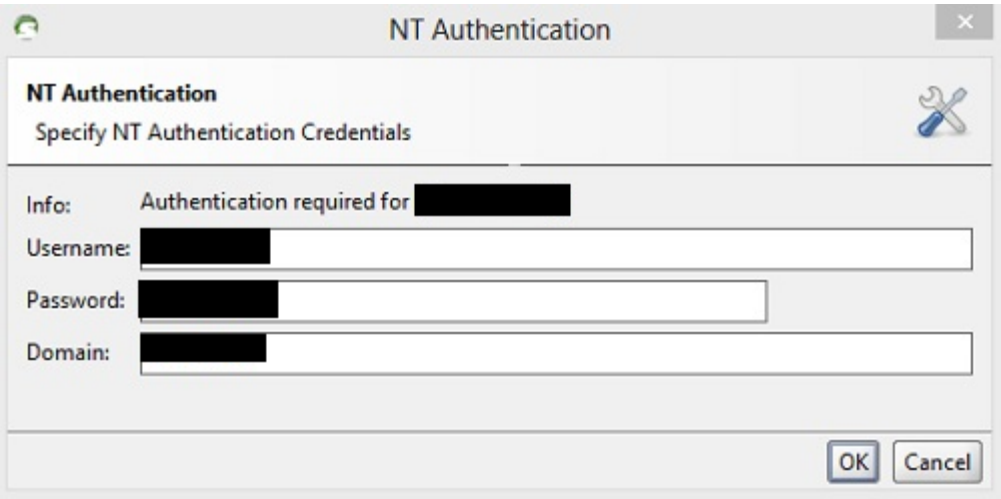
Img20: Request with an Authorization Header

Challenge 3:

What if you are testing a web service with NTLM Authentication? How do you configure SoapUI for that?

Solution:

In the case of NTLM Authentication, while creating a project, SoapUI auto detects the authentication type and ask for credentials as shown in Img21.

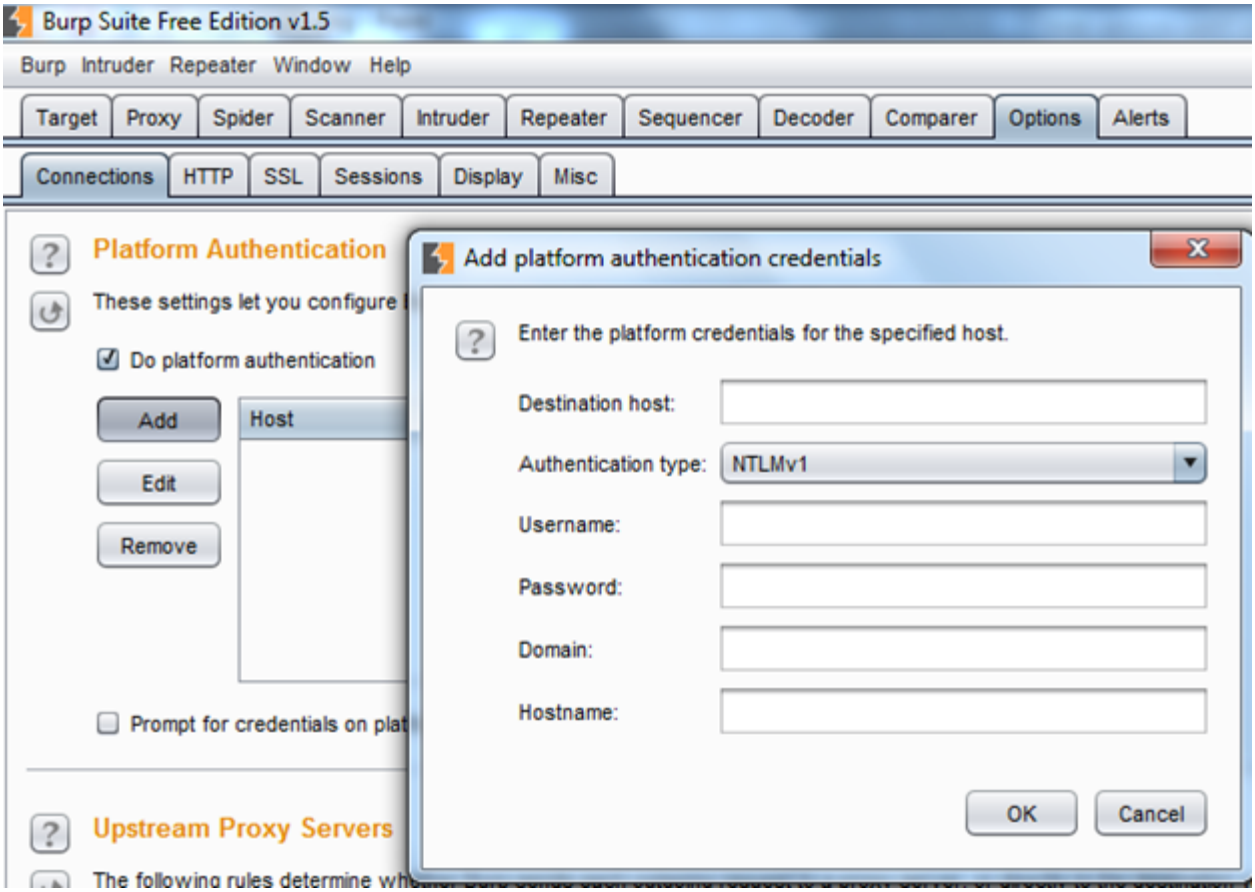


Img21: NTLM Authentication window

Sometimes even when you give correct credentials, SoapUI will fail to fetch the requests from WSDL. Though SoapUI 4.6.3 comes with better NTLM Authentication feature, I have not used it yet. So the problem is that if after providing correct credentials, SoapUI will fail to generate requests. What to do?

Solution:

Integrate your burp with the browser. Download and install SOA Client Mozilla add-on. Then go to the Options tab in your Burp, and under the Authentication platform, add new authentication type, enter the credentials in the window, select Authentication type as NTLM V1/V2 as shown in Img22 and start testing with the SOA Client.



Img22: NTLM Authentication configuration

Conclusion:

In this article we went through how fuzzing is an essential part of black box testing, how we can use information disclosed from one vulnerability in another, and what are the challenges you might face in testing web services, and what are their solutions.

In the next installment, we will focus on how Burp Suite Professional adds some extra features in Web services penetration testing apart from fuzzing, and mostly on the use of Burp Scanner and Extender while performing Web services penetration testing.