



SIFT SPECIAL PUBLICATION

A Web Services Security Testing Framework

Colin Wong
Daniel Grzelak
[research@sift.com.au]

Version: 1.00

SIFT [ABN 42 094 359 743]
Date: 10/11/2006

ABSTRACT

The broad genre of web applications has the Open Web Application Security Project (OWASP) testing framework as a standard process for building and operating a security test program. However, no such framework exists that is specifically tailored for the security testing of web services. Web services are becoming increasingly important in the enterprise sector and as a result, the security of web services will become paramount. This paper identifies and details an approach to web services security testing including tools, techniques and processes. It aims to provide a guide for organisations wishing to adopt a standardised process for evaluating security mechanisms in web services.

Contents

Abstract	2
Introduction	5
Web Services	5
Related Work	6
The Framework	7
Document Structure	8
Toolkit	9
Proxy	9
Scriptable SOAP Generator	11
Decoding / Encoding Library	11
SSL/TLS Cipher Specification Enumerator	12
Web Method Enumerator	12
Threat Modelling	14
Threat Modelling Process	14
Application Overview	15
System Decomposition	15
Threat Enumeration	20
Scoping	27
Scoping Process	27
Scoping Drivers	28
Scope Definition	28
Planning	30
Test Planning Process	31
Test Strategy	31
Test Plan	32
Information Gathering	33
WSDL Scanning	33
SOAP Fault Error Messages	34
Web Method Enumeration	35
Fuzzing	37
Vulnerabilities	37
Injection	44
SQL Injection	45
Command Injection	46
LDAP Injection	47
XPath Injection	48
Code Injection	49
XML Injection	50
Other Injection	52
Confidentiality & Integrity	53
Confidentiality	53
Integrity	55
Web Services Security Extensions	56
Logging	59

Characterisation	59
Logging Issues	60
Logic Flaws	65
Authentication & Authorisation	67
Authentication	67
Authorisation	73
Availability	78
Reporting	85
References	86
Glossary	90
Appendix A: Threat Profile Template	92
Appendix B: Scope Definition Template	93
Appendix C: Test Strategy Template	94
Appendix D: Web Services Testing Checklist	95
Appendix E: Test Case Threat Mappings	100
About SIFT	103
Our Profile	103
Our Services	103

Figures

Figure 1: Web Service Deployment Tiers	6
Figure 2: Web Services Security Testing Process	7
Figure 3: Web Services Security Testing Toolkit	9
Figure 4: Web Services Architecture	19

Tables

Table 1: Forms of Message Manipulation.....	10
Table 2: Common Encoded Formats	11
Table 3: Generic Web Service Trust Levels	16
Table 4: Generic Web Service Assets	17
Table 5: Generic Web Service Entry Points.....	18

Copyright & Legal

This report is the intellectual property of SIFT Pty Limited.

This report may be reproduced, printed, stored and/or distributed provided full attribution to SIFT is provided, and no part of the document is altered or removed in any way.

In no event shall SIFT Pty Limited be liable to anyone for special, incidental, collateral, or consequential damages arising out of the use of this information.

SIFT® is a Registered Trademark of SIFT Pty Limited. Many designations used by manufacturers and sellers to distinguish their products are claimed as trademarks or other proprietary rights. Where designations appear, and when the editorial staff were aware of a claim, the designations have been shown. Other trademarks, registered trademarks, and service marks are the property of their respective owners.

Copyright © 2006 SIFT Pty Limited. All rights reserved.

Originated in Australia

INTRODUCTION

Web services are a widely touted technology that aim to provide tangible benefits to both business and IT. Their increasing use in the enterprise sector for the integration of distributed systems and business critical functions dictates the need for security assurance yet there is currently no security testing methodology specifically adapted to applications that implement web services. Although many application security testing principles can be generically applied to web services, particular aspects of the technology such as its reliance upon XML and web services specific standards require closer attention that is not provided by other testing methodologies [SUN 1]. Thus, a comprehensive framework for evaluating the security of web service implementations during all phases of the development cycle is required.

This paper is intended for web service developers and testers and due to the nature of the material, will necessarily contain a large amount of technical information. It assumes familiarity with common web service platforms and practices as well as an understanding of cryptography and information security principles.

This paper will describe a testing methodology for web services security and outline a process that can be adopted to evaluate web services security throughout the development lifecycle. The following material will be covered:

- The adaptation of security testing principles to the specifics of web services
- Threat modelling and profiling
- Test scoping
- An in-depth treatment of web services security testing techniques

When used, it is essential that this framework be part of the overall development lifecycle. Penetration style testing of web services may provide some degree of assurance that a web service is secure but is no replacement for designing the web service with security in mind from the ground up. This paper should be considered a testing supplement to an overall secure development process that considers the web service under test from an attacker's perspective in order to aggressively discover exploitable vulnerabilities.

Web Services

'Web services' is the name given to a technology for providing loosely coupled and interoperable communications between computer systems over a network. Applications of this technology include enterprise application integration, enabling communications between distributed systems and interoperability with partners or other businesses. The major benefits of using web services are platform independence, flexibility and ease of maintenance, which are the result of the open standards employed and the simplified underlying request-response concept that avoids the complexity of other distributed technologies such as CORBA.

Web services communications are primarily XML-based and essentially provide the ability to make RPC-like function calls between remote systems, regardless of their underlying platform. Specifically, SOAP over HTTP is the most common implementation of a web service. Supporting protocols such as UDDI [OASIS 1] and WSDL [W3C 1] provide discovery and description services that contain enough information for client applications to be built to utilise the service.

The ability to expose functionality through HTTP provides the benefit of minimal firewall configuration but also has the effect of circumventing firewall rules that may restrict unauthorised access to these functions. The addition of UDDI and WSDL allow remote users to easily locate and use the functionality provided. This exposure poses a far greater threat than was present in the previous generation of distributed technologies that typically used proprietary binary encoding schemes and restrictive firewall rule sets to limit access.

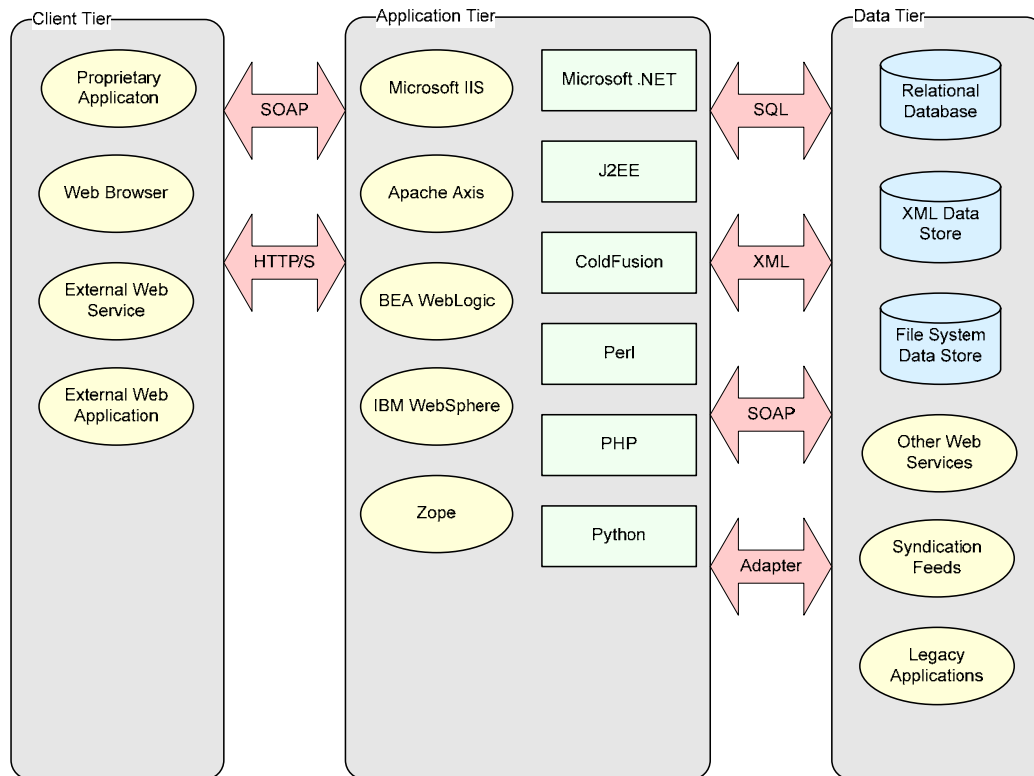


Figure 1: Web Service Deployment Tiers

Web services communication is based upon the Simple Object Access Protocol (SOAP). SOAP is an XML-based information packaging definition which can be used for exchanging structured and typed information between peers in a distributed environment, relying on Internet transport protocols such as HTTP. Because SOAP is standards based, it also provides interoperability in heterogeneous environments.

For more complete discussion refer to the official documentation from the W3C. The most relevant documents are "SOAP Version 1.2 Part 0: Primer", [W3C 2], "SOAP Version 1.2 Part 1: Messaging Framework" [W3C 3] and "SOAP Version 1.2 Part 2: Adjuncts" [W3C 4].

Related Work

At the time of writing, there has been no work on a formalised security testing methodology specifically targeted at web services. However, a testing framework aimed at standard web applications exists and provides a very informative starting point.

OWASP

Although the Open Web Application Security Project (OWASP) is aimed at the more general category of web applications, it covers application security issues highly relevant to web services. The OWASP Testing Project [OWASP 1] and the OWASP Web Application Penetration Checklist [OWASP 2] contain detailed information on web application security testing from start to finish. These can be thought of loosely as a model for the approach taken in this document.

The OWASP Testing Project is divided into four sections. The first two provide an introduction, background and overview of the project, from scoping to principles of testing. This is done to establish an overall contextual basis from which testing can begin. Such a basis is essential, and parts of the same introductory material are also covered in this document.

The final two sections describe processes that can be adopted to ensure a thorough review of a web application. Although not as relevant in this context, some material such as that covering threat modelling and test workflow descriptions is applicable.

The Framework

This web services security testing framework follows a similar progression to the majority of software testing projects with the addition of a threat modelling phase. When used as part of a security assessment, it is suggested that the major phases of the framework be executed linearly. In instances where this framework is adopted as part of the testing process within the development lifecycle, regression testing may require that the test execution phase be completed multiple times as required.

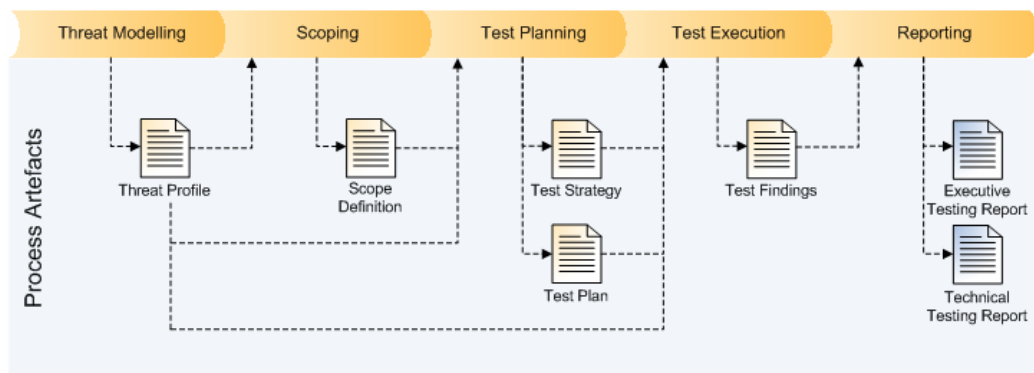


Figure 2: Web Services Security Testing Process

The framework begins with threat modelling to produce a threat profile necessary to understand potential system risks. The threat profile facilitates scoping of the required tests and later aids in planning and test execution. After a scope definition is created, planning takes place based on the defined scope and nature of potential threats.

Once a test strategy and a test plan are available, the core test execution phase can begin. This phase has been divided into a number of key test areas:

- Information Gathering
- Fuzzing
- Injection
- Confidentiality & Integrity

- Logging
- Logic Flaws
- Authentication & Authorisation
- Availability

Finally, the results of these tests are used to generate a report or reports relevant to both a business and technical audience.

Document Structure

This document follows the structure laid out by the framework illustrated in Figure 2 with each major step corresponding to a major section within this document. Each core test area is also documented as a major section of this document. Additionally, the document covers the Web Services Security Testing Toolkit and includes a number of appendices which can aid in applying a structured testing methodology and in creating consistent deliverables.

Throughout each testing section, each type of test is accompanied by a test case description table. These test cases can be used to ensure that each type of test is executed and relevant areas covered. Each test case includes the following information:

- Reference – A means of identifying the test case
- Test Case – The name of the issue being tested
- Objective – The aim of performing the tests described in the test case
- Description – A description of the issue being tested, and other background information required to execute the test case
- Threat – The threat (as per the Threat Modelling section) that corresponds to this test
- Impacted Asset – The asset (as per the Threat Modelling section) that is targeted by the test
- Test Method – The process used to perform the testing

The final section of the document consists of the following test aids:

- Threat Profile Template – Provides a structure for documenting the threat profile of an web service that will be tested
- Scope Definition Template – Provides a structure for creating a scope definition document
- Test Strategy Template – A starting point for defining a test strategy
- Web Services Testing Checklist – A checklist of test cases that must be undertaken throughout the testing project
- Test Case Threat Mappings – A table that provides an indication of which threats apply to which test cases

TOOLKIT

Prior to the discussion of testing processes, it is important for security professionals and developers alike to understand the tools necessary for testing. This section briefly outlines a number of tools that may be useful in conducting a web service penetration test, including:

- A web services proxy;
- A scriptable SOAP generator;
- A decoding/encoding library;
- An SSL/TLS cipher specification enumerator; and
- A web method blindcrawler.

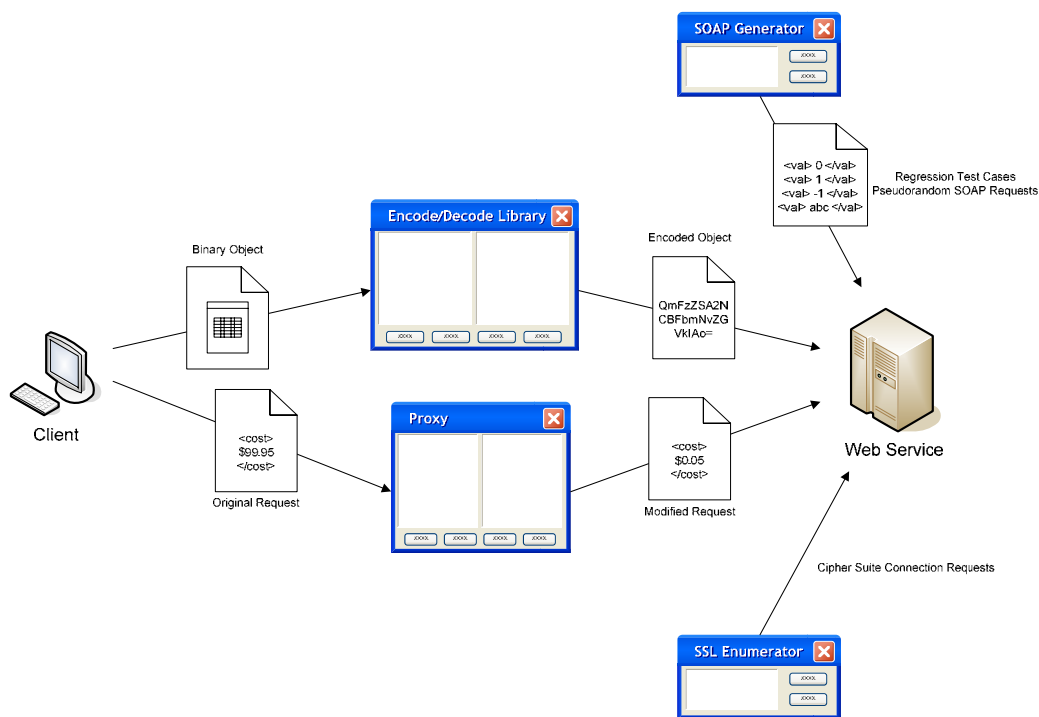


Figure 3: Web Services Security Testing Toolkit

Proxy

For the purposes of manipulating web services communications and verifying trust boundaries, a HTTP proxy with SOAP extensions is required. The proxy is inserted between client and server and should be capable of intercepting SOAP messages with the option of inserting, removing and modifying messages in the communications stream. This will allow the tester to observe the exchanges between client and server as well as manipulate the application protocol in order to exploit weaknesses. The proxy should also provide the ability to encode and decode common formats that may be used within SOAP communications to ensure its usefulness in differing environments.

Communications

Web services can use a variety of different underlying protocols as communication channels. The most common of these in use is SOAP over HTTP(S) [W3C 4]. Less popular communication channels available to web service implementations include SMTP, message queues and basic TCP or UDP. Support for SOAP over HTTP(S) should be considered a minimum requirement in the proxy and additional SOAP bindings should be supported via a plug-in interface.

Certain testing circumstances may require the proxy to translate between different protocols on each communication endpoint. For example, it is typically easier to perform testing with a plaintext tool but the server itself may require SSL connections. In this case, the proxy can accept HTTP traffic from the tester but initiate a HTTPS connection with the server endpoint. Another situation that may arise is the use of HTTP keep-alives and it may be necessary for one half of the proxied connection to employ keep-alives whilst the other half is constantly torn down and renewed. Additionally, support for translating between HTTP 1.0 [RFC 1945] and HTTP 1.1 [RFC 2616] may be a useful feature for identifying weaknesses in the handling of protocol versions.

Data Format Parsing and Display

The proxy tool should be capable of understanding the underlying protocols of web services communications and displaying them to the tester in a readable format. Of primary concern is the SOAP protocol with a HTTP binding. The ability to break down protocol elements and display them individually will prove highly useful to the tester and should cover elements such as HTTP headers, SOAP envelope headers and SOAP envelope body elements as well as support web services extensions such as WS-Security.

Message Manipulation

The principal purpose of the proxy is to enable the tester to manipulate messages sent between server and client in order to evaluate trust boundaries and input handling. There are many different actions that the proxy should allow:

ACTION	DESCRIPTION
Raw editing	Manipulate messages at the character or byte level.
Modify messages	Modify messages in accordance with the underlying protocol.
Inject messages	Construct an artificial message and insert it into the stream.
Remove messages	Remove a genuine message from the stream.
Delay messages	Introduce artificial delays in the message stream.
Reorder messages	Control the order of the messages in the stream.
Replay messages	Capture and re-insert messages into the stream.
Scripted modifications	Automatic changes to all messages in the stream.
Triggered modifications	Automatic changes to all messages in the stream that meet a certain criteria. The criteria may be based on the underlying protocol or the message contents. Regular expression support would be advantageous.

Table 1: Forms of Message Manipulation

Logging

Software testing in any form is about providing diagnostic information to those in a position to fix issues found whilst also providing a level of assurance to those responsible for software projects. To increase the repeatability of tests and provide an automatic audit trail of the tests performed, the proxy should log both the entire original and modified message streams as well as the modifications performed. This provides a form of automatic documentation that will allow the tests to be easily repeated and is also of use for diagnostic and problem resolution purposes.

Scriptable SOAP Generator

A scriptable tool that is able to generate and send web service requests is required for efficient web services testing. This tool can be used to test the strength of the web service against automated test cases as well as hand crafted custom inputs.

Such a tool must remove some of the burden of creating and executing mundane test cases such as testing for integer and buffer overflows. This can be achieved through automation of test case creation and execution, which typically involves progressively changing input sizes or values in either a predefined or pseudorandom manner. This approach is also well suited for evaluating boundary conditions.

The SOAP generator will be the primary tool for providing automated testing. A number of security tests cannot be automated but a large portion can. To increase the efficiency and timeliness of testing, the tool can be used to create a repeatable test suite that can quickly identify which vulnerabilities still remain in a web service in subsequent phases of testing. In this way tests could be scheduled whenever required and regression testing performed as part of a regular cycle.

Finally, the tool must be able to distinguish between success and failure. Web services, especially with a HTTP binding can flag errors in many different ways and support for these needs to be available. Some examples include the 400 series of HTTP error codes, 200 series pages with custom messages, server disconnection and the SOAP Fault block.

Decoding / Encoding Library

SOAP messages will typically represent complex data structures that contain both binary and non-binary data. As these data structures tend to be custom-made for the application, it is important to build a general-purpose library of encoders and decoders for popular formats. These components should be built as a plugin library that will allow use by both the proxy and the SOAP generator. These encoders and decoders can then be called to perform their actions on particular fields in SOAP messages that contain a particular data format. Some common formats are described in the following table.

FORMAT	DESCRIPTION
Zip	Popular file compression algorithm.
Base64/Unicode	Common formats for representing binary data.
DES/3DES/AES/Blowfish	Common data encryption standards.
MD5/SHA-1/SHA-256	Common message digest standards.

Table 2: Common Encoded Formats

In addition to simply performing the encoding or decoding functions, the encoder and decoder plugins should also provide the tester with additional functionality including the ability to:

- View the decoded data stream
- Modify the decoded data stream before re-encoding it
- Automatically decode specified fields in SOAP messages

This provides the facility to test important aspects of the web service that would otherwise be a time-consuming manual process. Copies of the original encoded data and the modified versions should be logged to an appropriate location to ensure traceability.

SSL/TLS Cipher Specification Enumerator

The common deployment of Secure Sockets Layer (SSL) [FRE 1] or Transport Layer Security (TLS) [RFC 2246] to protect web services communications has resulted in the need for an efficient method of discovering the supported cipher specifications. A tool is required that can repeatedly establish connections with a server, selecting a different cipher specification on each connection. The tool can then evaluate the server response received for each connection attempt and determine the cipher specifications supported by the server.

SSL/TLS cipher specifications include:

- AES128-SHA
- AES256-SHA
- DES-CBC3-MD5
- DES-CBC3-SHA
- DES-CBC-MD5
- DES-CBC-SHA
- DHE-DSS-AES128-SHA
- DHE-DSS-AES256-SHA
- DHE-DSS-RC4-SHA
- DHE-RSA-AES128-SHA
- DHE-RSA-AES256-SHA
- EDH-DSS-DES-CBC3-SHA
- EDH-DSS-DES-CBC-SHA
- EDH-RSA-DES-CBC3-SHA
- EDH-RSA-DES-CBC-SHA
- EXP1024-DES-CBC-SHA
- EXP1024-DHE-DSS-DES-CBC-SHA
- EXP1024-DHE-DSS-RC4-SHA
- EXP1024-RC2-CBC-MD5
- EXP1024-RC4-MD5
- EXP1024-RC4-SHA
- EXP-DES-CBC-SHA
- EXP-EDH-DSS-DES-CBC-SHA
- EXP-EDH-RSA-DES-CBC-SHA
- EXP-RC2-CBC-MD5
- EXP-RC4-MD5
- RC2-CBC-MD5
- RC4-64-MD5
- RC4-MD5
- RC4-SHA

An assessment of the classes of cipher specifications supported should be performed and weak cipher support should be noted. At the time of writing, weak cipher specifications are considered to include all specifications that use no encryption, DES, RC2, anonymous Diffie-Hellman or have symmetric key lengths of less than 128-bit.

Web Method Enumerator

A dictionary attack tool can be used to brute force the web method names for a given web service under some circumstances. That is, SOAP requests can be submitted to a web service using probable combinations of words. This is possible because responses to requests for non-existent and existent web methods differ markedly under most platforms.

For example, searching for the “myWebMethod” web method can be achieved with the following request:

```
POST http://myWebService/uri HTTP/1.0
SOAPAction: myWebMethod
Content-Type: text/xml

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Body>
    <myWebMethod xmlns="http://myWebService/xmlns">x</myWebMethod>
  </soap:Body>
</soap:Envelope>
```

The following messages within a SOAP response might indicate the web method does not exist:

- “No method named”
- “Server did not recognize the value of HTTP Header SOAPAction”
- “No such operation”

While the following messages may indicate that the web method does exist:

- “Method exists”
- “The arguments do not match the signature”
- “IllegalArgumentException”

There are countless other variations of responses and anomalies within responses which can indicate the presence of a web method. These fingerprints vary by platform, platform version, and application implementation and are used by the SIFT Web Method Search tool [SIFT 3].

THREAT MODELLING



Prerequisites:

- None

Outcomes:

- Threat Profile Document

Threat modelling is the process of identifying potential threats and their impacts on a system. It is typically used in the software development life-cycle (SDLC) or risk management process to enable developers and testers to formulate a global view of potential risks posed to a system. Thus, threat modelling provides a foundation for the risk profile associated with the deployment of web services, and the potential exposure created with the availability of these services [MEI 1].

In an ideal situation, the threat modelling process acts as a driver for security testing efforts. It is recommended that a thorough threat modelling process be completed before undertaking security testing. Once a threat profile has been developed, testers can better focus their efforts on the planning and execution of tests across identified threat paths.

Threat Modelling Process

- 1) Create an application overview
 - a) Identify key functionality
 - b) Identify roles
 - c) Identify application architecture
 - d) Identify the technologies used
- 2) Decompose the system
 - a) Identify trust levels and trust boundaries
 - b) Identify assets
 - c) Identify entry points
 - d) Identify data flows
 - e) Identify use cases
 - f) Identify security mechanisms
- 3) Threat enumeration
 - a) Identify common threats and attacks
 - b) Identify threats in data flows
 - c) Identify threats in use cases
 - d) Identify threat entry points
 - e) Identify impacted assets
- 4) Document the threat profile

Application Overview

In order to determine the threats to a web service system, it is essential to grasp the basic architecture and structure of the application. At a minimum, testers should have a general appreciation of the key functionality of the application and a familiarity with the defined user roles. An understanding of the technical implementation and architecture is also useful and can be gained from design or technical documentation.

Having an understanding of the key functions of the application simplifies the system decomposition process and makes process and data flows more readily apparent to the tester. Additionally, knowledge of user roles within the system will assist analysis of access controls and technical details may narrow the area of testing required, depending on the implementation choices made with regard to platforms, frameworks and other factors.

The application overview is commonly captured in diagrammatic form such as a module or deployment diagram. It should be updated as the system is decomposed and new components are discovered. The identified threats can also be placed on the diagram with the affected components indicated.

System Decomposition

System decomposition involves breaking down the system into its component parts in order to gain a deeper understanding of it. By comprehending more of the inner workings of the system, the ease with which threats can be discovered improves and the completeness of the analysis will be increased.

The following elements of the web service application are to be broken out to assist threat enumeration:

- Trust levels
- Assets
- Entry points
- Data flows
- Use cases
- Security mechanisms

Trust Levels

Trust levels describe the degree of confidence and reliance between interacting elements in a system from the perspective of the application. To identify the trust levels at various points in the system, it is important to consider attributes such as the state of authentication, authorisation levels and validation performed. As trust levels are uncovered, it is also essential to identify trust boundaries – that is, where trust levels change – as these areas commonly contain security-specific processing and validation. A number of generic web service trust levels are presented below that can be tailored to the specific web service under test.

ID	NAME	DESCRIPTION
1	Server remote	Any incoming connection to a web service.
1.1	Anonymous server	Incoming connections to the web service that can be handled without

ID	NAME	DESCRIPTION
	remote	authentication.
1.2	Authenticated server remote	Incoming connections to the web service that must be authenticated before proceeding. This trust level should be broken down further to reflect the particular roles and access levels used in the web service being analysed.
2	Local process	Privilege levels relating to processes running on the client and server infrastructures.
3	Client remote	Any incoming connection to a web service client.
3.1	Server responses	Connections to a client that come from the web service server.
3.2	Intermediary responses	Connections to a client that come from a web service intermediary.
4	Network	Any other connection.

Table 3: Generic Web Service Trust Levels

Assets

The assets of a system are the elements that hold value and need to be protected from unauthorised access and damage. The identification of assets is a necessary step in threat modelling to determine the targets of potential attacks and also to assess the impacts of a successful attack. A number of generic web service assets are outlined in the table below that should be tailored to the specific web service under test. These assets will be referenced in later test cases to illustrate the likely targets of each attack scenario.

ID	NAME	DESCRIPTION	TRUST LEVELS
1	Web service functionality	The specific functionality that is provided by the web service. This asset must be broken down specifically for the particular application under test.	Server remote
2	Credentials	Information that identifies one entity to another.	Client remote Network Server remote Local process
2.1	Username	Unique user identifiers used to access the web service.	Client remote Network Server remote Local process
2.2	Password	Secret strings that verify the identity of users before granting access to the web service.	Client remote Network Server remote Local process
2.3	Tokens	Secret tokens used to maintain session state or provide authorisation between domains.	Client remote Network Server remote Local process
2.4	Certificates	Digital document that is a verified identity	Client remote

ID	NAME	DESCRIPTION	TRUST LEVELS
		claim.	Network Server remote Local process
3	Web service availability	The ability of all legitimate users to access the web service when required and receive a timely response.	Network Local process Server remote
4	Data	Information consumed or produced by the web service.	Network Local process
4.1	Log files	The audit data stored in system logs.	Local process
4.2	Stored data	Information stored on the web service backend infrastructure.	Local process Authenticated server remote
5	System access	Interactive access to the underlying system hosting the web service.	Local process
6	Network resources	Network elements and infrastructure that is required for the correct functioning of the web service.	Server remote Local process
7	System information	Information about the underlying platform of a web service and clients.	Network Local process
7.1	Client system information	Client platform information, versions and services.	Network Local process
7.2	Server system information	Server platform information, versions and services.	Network Local process
7.3	Intermediary system information	Intermediary platform information, versions and services.	Network Local process

Table 4: Generic Web Service Assets

Entry points

An entry point is an avenue for incoming data to enter the system. By definition, entry points must exist for a web service to receive input and perform its purpose. However, they also provide the ability for malicious inputs and attacks to enter the system and cause damage. Entry points can include exposed interfaces such as public web service methods as well as internal communications or administrative interfaces. The table below describes some common entry points employed by web services and their associated trust levels. This should be customised specifically for the web service under test.

ID	NAME	DESCRIPTION	TRUST LEVELS
1	Endpoint socket	The listening socket associated with the web service endpoint.	Anonymous server remote
1.1	Authentication web method	The method that authenticates a user as having higher privilege, usually providing some sort of corresponding session token if the supplied credentials are correct.	Authenticated server remote

ID	NAME	DESCRIPTION	TRUST LEVELS
1.2	Log off web method	The web method that disassociates a user from an active session with the web service.	Authenticated server remote
1.3	Web method	All web methods deployed by the web service that are specific to the application being analysed. This entry point should be decomposed into the functional groups of the web service under test.	Server remote
2	Client socket	The client socket which initiates a connection with the web service endpoint or an intermediary.	Client remote
2.1	Web method request	Any web service call originating from a client.	Network
2.2	Web method response	Any reply to a web method call.	Client remote
3	Intermediary sockets	The connections established from the client to an intermediary, an intermediary to any other intermediary, and an intermediary to the endpoint.	Client remote Server remote
3.1	Intermediary handled client request	A request handled by an intermediary where the last processing node was the client	Server remote
3.2	Intermediary handled endpoint response	A response handled by an intermediary where the last processing node was the web method endpoint.	Client remote
3.3	Intermediary handled intermediary request	A request handled by an intermediary where the last processing node was another intermediary.	Server remote
3.4	Intermediary handled intermediary response	A response handled by an intermediary where the last processing node was another intermediary.	Client remote
4	Network	Any device on the network path between client and endpoint.	Network

Table 5: Generic Web Service Entry Points

Below is a commonly used architecture for delivering web services to clients, with entry points of potential threats identified.

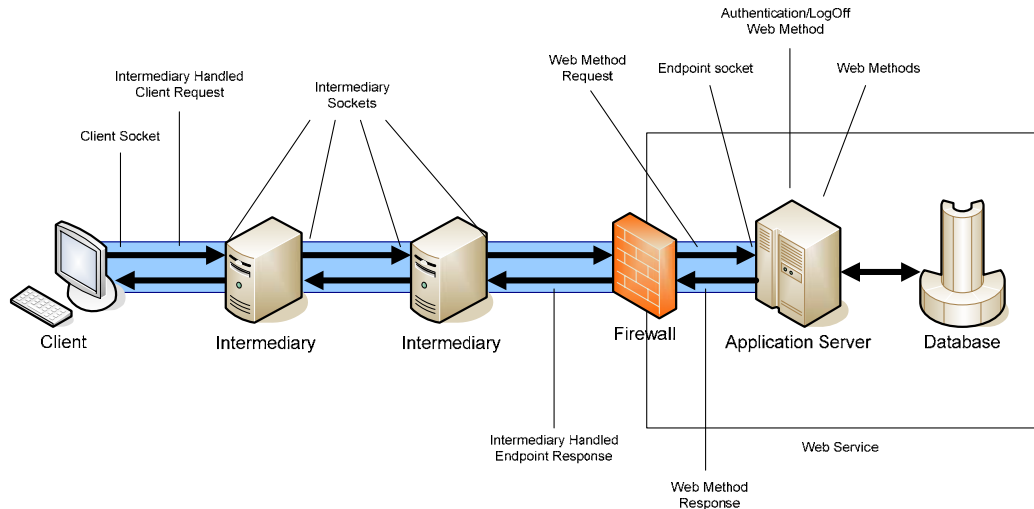


Figure 4: Web Services Architecture

Gaining access to web services entry points can be achieved through various components of web services communications. Specifically, testing should incorporate the following elements as appropriate:

- HTTP Parameters

While not always accessible by the web service, HTTP parameters are sometimes used to carry additional information not supplied by the web method caller.
- SOAP Header Content

The SOAP header often contains a wealth of information used by both the endpoint and any intermediaries. Information used as part of extensions is also often held in the header.
- SOAP Body Content

All SOAP body content should be tested as it is where the majority of useful information resides.
- SOAP Attributes

As with any other XML documents, SOAP messages have attributes which can be accessed and used by applications and are therefore usable in attacks.
- SOAP Attachments

SOAP Messages can contain attachments in a multipart MIME structure. SOAP attachments are being increasingly used and can hold dangerous data and therefore should be considered in testing.

Data Flows

Data flows represent the paths that web service requests and responses take between client and server. It is necessary to identify data flows to grasp the interactions of the web service with external entities. As data moves between nodes and across trust boundaries, it must be properly validated and protected. The type of information and the protections around it must be suitable for the trust level the data is passing through. Points at which data flows cross trust boundaries typically require greater attention as validation and processing is concentrated in these areas and is often targeted by threats.

Use Cases

Functional use cases are another source of information that can be used to derive potential threats to the web service. Use cases can be analysed for ways in which functionality can be invoked without authorisation or malicious data inserted. Attempts should be made to identify weaknesses present in the flow of events of each use case that may provide an avenue of attack due to a deficiency in the process or the implementation of the functionality.

Security Mechanisms

When modelling threats, it is also useful to determine the security mechanisms in place so that gaps in coverage can be identified. The identification of security mechanisms, their coverage of entry points and the extent of protection that they afford assets should be considered so that threats circumventing these mechanisms can be developed. Additionally, it will be important later in the testing process to verify the correct operation of identified security mechanisms in response to attempted attacks.

Threat Enumeration

In this section we present an overview of the threats faced by web services [HOL 1]. This is a generic high-level list of threats and is not intended to be exhaustive. Many threats will be application and implementation specific and must be determined by analysing the elements identified in the previous section on system decomposition. For each threat, it is necessary to identify the entry points or attack vectors and potentially impacted assets. It may also be useful to determine the trust levels required to make use of the entry point.

Threat enumeration and identification is typically performed by a small group of security specialists, developers and testers as a brainstorming exercise. However, it is important to capture the threats identified in a form that can be later used to produce the threat profile.

Threat Classification

Threats are often classified according to their type. Threat types are often associated with specific security mechanisms and can therefore be quickly mitigated. One form of classification is known as the STRIDE model [MEI 2], describing different categories of threats:

- Spoofing
- Tampering
- Repudiation
- Information Disclosure
- Denial of Service
- Elevation of Privilege

The STRIDE model has been adopted for the purpose of threat modelling in this paper. The following is a generic STRIDE threat classification for web services which should be tailored for the domain of the specific web service being tested. Each threat is identified with a reference number and contains a description, possible architecture entry points and a list of assets that may be impacted.

Spoofting

Spoofting covers the broad use of faked credentials to gain access to resources that the attacker is not entitled to access. If a web service inadequately verifies the credentials presented by its users, it may be susceptible to spoofting attacks.

In a web services environment, there are potentially several parties involved. The server, client and any number of intermediaries may be involved with the creation and processing of SOAP messages. These parties may reside between different trust boundaries and may be susceptible to spoofting attacks.

There are various ways to exploit credentials or spoof the source of messages. These include credential forgery, session hijacking and impersonation attacks. The consequences of these spoofting attacks can be severe and may result in the compromise of data and system integrity.

Reference	T-S01
Threat Description	Spoofed requests
Impacted Assets	Web service functionality Credentials
Entry Points	Web method request
Attack Techniques	Attacker spoofs a request from a legitimate client
Countermeasures	Strong authentication

Reference	T-S02
Threat Description	Spoofed responses
Impacted Assets	Credentials Data
Entry Points	Client socket – Web method response
Attack Techniques	Attacker spoofs a response from the server
Countermeasures	Strong authentication

Tampering

Tampering is the unauthorised modification of data in a system or as it flows between components of a system. Failure to adequately protect data stores or communication channels will make a system vulnerable to tampering attacks.

Web services are typically deployed as part of complex systems that involve several distributed parties. As a result, there may be multiple communication paths and data stores in a single system that will require protection. This is especially important if the communications channel crosses a trust boundary.

Tampering attacks include modifying corporate data, man-in-the-middle attacks and the insertion of malicious software such as viruses and Trojans. The effects can be serious and cause disruptions to service and loss of data integrity.

Reference	T-T01
Threat Description	Man-in-the-middle attack
Impacted Assets	Credentials Data
Entry Points	Network
Attack Techniques	Attacker inserts themselves into the communication path between communicating parties and impersonates each to the other
Countermeasures	Strong authentication

Reference	T-T02
Threat Description	Injection attack
Impacted Assets	Data System access
Entry Points	Endpoint socket - Web method request
Attack Techniques	Attacker supplies malicious input that is directly used in calls to an external or underlying system
Countermeasures	Input validation and sanitisation

Reference	T-T03
Threat Description	Insertion of malicious software
Impacted Assets	System access Network resources
Entry Points	Endpoint socket - Web method request
Attack Techniques	Attacker manages to execute or place malicious software on the system
Countermeasures	Input validation and sanitisation

Repudiation

Repudiation is the denial by a party of having performed an action, despite the action having actually been completed by that party. This threat exists when other parties have no method of proving that something occurred. This can arise if the web service does not bind the caller to their actions or if auditing and logging controls have been bypassed.

The threat of repudiation can lead to data inconsistencies and the inability to prove or disprove transactions have occurred. Auditing is made more difficult as the audit trail may be incomplete or tampered with.

Reference	T-R01
Threat Description	Denying a web service transaction
Impacted Assets	Credentials Data Log files
Entry Points	Client socket Endpoint socket
Attack Techniques	Attacker exploits inadequate logging to deny the occurrence of a transaction
Countermeasures	Robust logging controls Consistent identity management

Reference	T-R02
Threat Description	Replay attack
Impacted Assets	Web service functionality Data
Entry Points	Endpoint socket - Web method request
Attack Techniques	Attacker captures and replays a valid web method request to the server
Countermeasures	Mechanism for detecting duplication

Information Disclosure

Information disclosure is the ability for users to access data that they are not authorised to access. If a web service does not adequately verify the identity of its users it may be unnecessarily disclosing confidential data or web services implementation details to unauthorised or malicious users.

As web services are typically implemented as part of a complex system and have access to a large amount of potentially sensitive information, it is important to ensure that access to the information is restricted. The transfer of the data should also be secured to prevent eavesdropping and sniffing threats.

Information disclosure can have serious consequences for an organisation. Attackers may exploit information disclosure vulnerabilities to steal confidential data or gain details about the system that may allow further system penetration.

Reference	T-I01
Threat Description	Eavesdropping on communication paths
Impacted Assets	Data System information Credentials
Entry Points	Network
Attack Techniques	An attacker on the communication path captures passing network traffic
Countermeasures	Strong encryption

Reference	T-I02
Threat Description	Information leaked by verbose error messages
Impacted Assets	System information
Entry Points	Endpoint socket - Web method request
Attack Techniques	Attacker induces error conditions to extract platform and implementation details
Countermeasures	Consistent exception handling that presents generic error messages

Reference	T-I03
Threat Description	Unauthorised access to services or data
Impacted Assets	Web service functionality Data Credentials
Entry Points	Endpoint socket - Web method request
Attack Techniques	Attacker bypasses access control mechanisms or uses a service that was not meant to be running
Countermeasures	Strong authorisation and access controls

Denial of Service

A denial of service threat aims to deplete the computing or network resources of a system or exploit an implementation flaw in order to prevent the system from providing services to its legitimate users. Web services are typically used in high availability and critical business systems and thus, a disruption to normal service can result in material productivity and financial losses.

Denial of service attacks can range from traffic floods that seek to overwhelm the network capacity of the system or supporting systems, to attacks that directly exploit some flaw in the implementation that halts the operation of the system.

Reference	T-D01
Threat Description	Deplete computer or network resources
Impacted Assets	Network resources Web service availability
Entry Points	Endpoint socket
Attack Techniques	Attacker floods the system with junk traffic or valid requests to overwhelm computers, routers, capacity etc.
Countermeasures	Detection and filtering of malicious traffic Business continuity and disaster recovery planning

Reference	T-D02
Threat Description	Exploit a programming or implementation flaw
Impacted Assets	Web service functionality Web service availability
Entry Points	Endpoint socket - Web method request
Attack Techniques	Attacker finds and exploits a flaw in implementation that causes the system to hang or crash
Countermeasures	Code review Comprehensive testing

Reference	T-D03
Threat Description	Corruption of data to prevent normal operation of the web service
Impacted Assets	Data Web service availability
Entry Points	Endpoint socket
Attack Techniques	An attacker corrupts application data by exploiting a vulnerability or using unauthorised functionality
Countermeasures	Strong authorisation and access controls Input validation and sanitisation

Elevation of Privilege

An elevation of privilege attack involves an attacker gaining access to data or functionality of systems to which they are not authorised. These vulnerabilities can arise as a result of many factors but are often due to flawed authentication and authorisation mechanisms. Another avenue of attack used for the elevation of privileges is the exploitation of coding vulnerabilities that allow remote code execution or database manipulation.

The consequences of an elevation of privilege attack are severe and may result in total compromise of a system. If an attacker manages to gain privileged access to a host then they have complete control over it and may steal data or leverage the compromised system to attack other hosts.

Reference	T-E01
Threat Description	Remote execution of code or software
Impacted Assets	System access
Entry Points	Endpoint socket - Web method request
Attack Techniques	An attacker exploits a buffer overflow vulnerability to access or execute commands on the target host with higher privileges than authorised
Countermeasures	Input validation and sanitisation Lowest privilege execution

Reference	T-E02
Threat Description	Administrative interfaces or functions available
Impacted Assets	Web service functionality System access
Entry Points	Endpoint socket - Web method request
Attack Techniques/Vectors	Attacker is able to access administrative interfaces or functions by bypassing access control mechanisms
Countermeasures	Strong authorisation and access control Out-of-band administration interfaces

A sample threat profile template is provided in Appendix A.

SCOPING

Threat Modelling

Scoping

Test Planning

Test Execution

Reporting

Prerequisites:

- Threat Profile Document

Outcomes:

- Scope Definition Document

Scoping is the process of defining project elements that are included in the exercise as well as those that are to be specifically excluded. Although some general scoping issues are covered in this section, guidance is primarily provided for the scoping of web services security testing based on security requirements and assessed threats and risks.

Test scoping should be undertaken by the security specialist in co-ordination with other stakeholders such as testers, developers and management. As different stakeholder groups have different and potentially conflicting interests, inputs from each group are needed during the scoping phase in order to ensure that the final test scope appropriately reflects the business and technical requirements for security assurance. Developers are important resources in the scoping process however their involvement with the creation of the system subject to testing may pose a conflict of interest or be biased towards functional rather than security requirements. For this reason it is suggested that they not be responsible for scoping but instead take a significant 'subject-matter expert' role in scope definition.

Scoping Process

The scoping process consists of four key elements, outlined below:

- 1) Identify security requirements
 - a) Review requirements specification
 - b) Identify security services
- 2) Assess threat risks
 - a) Choose risk rating scheme
 - b) Assign threat risks
 - c) Prioritise and rank threats
- 3) Define scope
 - a) Identify objectives
 - b) Identify boundaries
 - c) Determine elements out-of-scope
 - d) Determine elements in-scope
 - e) Correlate in-scope elements with security services and assets
 - f) Identify testing assumptions
 - g) Generate timing estimates
- 4) Document scope

Scoping Drivers

Security test scoping is most commonly expressed in terms of security requirements to be validated, threats to be tested or a combination of both. A web services system may have a particular security requirement and must therefore be assessed for compliance with that requirement. Conversely, given that a web services system is subject to a particular threat, the vulnerability to that threat might be analysed.

Security requirements are typically expressed in terms of the security service that must be provided to a particular asset or set of assets. A security service is defined to be a "...processing or communication service that is provided by a system to give a specific kind of protection to system resources" [RFC 2828]. Security services are implemented by security mechanisms to protect assets. An example of this is the use of the Advanced Encryption Standard (AES) cipher to protect information within a customer database. The security service being offered is data confidentiality, the asset being protected is customer information and the security mechanism used to achieve this is the AES cipher.

It is useful to employ security requirements as a driver for scoping as requirements contain objectives that can be verified as being met. Security requirements, and consequently security services, can typically be derived from the requirements specification documents of a system. Where requirements are not available or are unclear, some analysis may need to be performed with relevant stakeholders.

The alternative approach of using the threat model as a basis for testing requires additional effort but can provide greater coverage of tests where requirements may contain critical gaps. In order to use this approach, a risk assessment of the threats must be conducted to prioritise the threats and use those as a basis for testing. That is, testing will seek to verify that each threat is not manifested as an actual vulnerability with priority given to the highest risk items.

Security requirements are often a favourable alternative to the threat-risk approach because they tend to be created at a high level and can be easily extracted from previous work however an integrated approach may be the most practical. The most suitable approach for a web services security testing project should balance the need of management to fulfil requirements with the coverage and additional assurance provided by threat-risk driven scoping. The testing will then form the logical link between the necessary security requirements and assurance that the web service is not vulnerable to the identified threats. Note that for the purposes of this paper, scoping of the target objects will be defined at a generic high level that can be used to more specifically elicit scoping parameters for a particular system.

Scope Definition

The scope definition document provides a high level description of the elements of work to be undertaken and any additional information required to support the execution of Web Services testing. This may include the identified security requirements and/or threats & risks and how they relate to the testing. The scope definition document must also outline testing objectives, prioritisations, assumptions, exclusions, an expected timeline and success criteria for completion.

The objective of the scope definition document is to identify the elements that will be covered by the testing effort. To this end, it is necessary to prioritise the elements being tested by some agreed method, enabling allocation of testing resources in the most productive fashion. Security requirements can be prioritised on their business significance or may already have been prioritised through earlier activities.

Various schemes are available for prioritising threats but most typically focus on prioritisation according to likelihood, potential impact and asset importance. Some frequently used risk rating schemes are the Damage potential, Reproducibility, Exploitability, Affected users, Discoverability (DREAD) model [MEI 2] and the likelihood-severity model [AS 4360]. The choice of model and other important factors influencing prioritisation should be documented clearly.

Similarly, any specific exclusions to the testing effort should be verified and recorded. These exclusions may encompass items that have been explicitly defined as out-of-scope by management or not to be tested such as certain IP ranges, intermediaries or callable web services.

All testing necessarily involves making some assumptions and it is important that these are recorded. This is particularly important if the assumptions influence the scoping of the testing effort. Such assumptions may include the proper functioning of trusted intermediaries or backend servers. These assumptions may have a significant impact on the ability to effectively plan and execute the web services testing in a timely fashion, as well as the integrity or completeness of the test results.

Assumptions introduced during the scoping phase should be elicited from both technical and management sources, and agreed prior to test case development and execution.

The scope definition document should provide some timing estimates regarding the testing effort. These timing estimates are typically made by the testers and will necessarily take into account the scale of the web service under test as well as resourcing, access restrictions and other project constraints. It is expected that an estimated timeline for the testing of high level elements will be further refined during the planning phase. Timing estimates for non-technical tasks are usually the responsibility of project managers and not the testers. It is difficult to generalise execution times because they are highly environment dependant.

A more detailed analysis of the security services, mechanisms, assets and threats should be included within the scope document. The security services and mechanisms, protected assets and how the identified threats interact with these elements should be documented. Testing elements are to be assigned priorities, with the appropriate justification documented for future reference. Estimates should be made of the effort and resources required for testing each particular security service.

Prioritisation of testing elements is followed by selection of the specific elements of testing that are to be undertaken. This is the principle purpose of the scope definition document and will determine the focus of testing efforts. Depending on the specific web service itself and the resources available, it may not be possible to adequately cover all security requirements or threats and thus prioritisation can be used to refine the scope to fit within these constraints.

Finally, the document should contain a set of exit criteria for each scope item, ensuring that testing objectives are met. These criteria will be used to ensure suitable testing has been executed.

Proving the web service is secure to a defined checklist of tasks or failing to identify any vulnerabilities should not be considered as critical success factors. Exit criteria should be identified for each security service and should consider the coverage and completeness of testing.

A sample scope definition template is provided in Appendix B.

PLANNING

Threat Modelling

Scoping

Test Planning

Test Execution

Reporting

Prerequisites:

- Threat Profile Document
- Scope Definition Document

Outcomes:

- Test Strategy Document
- Test Plan Document

Effective web services security testing requires the development of both Test Strategy and Test Plan documents. The Test Strategy document provides a high-level overview of the approach to be taken for testing, whilst the Test Plan provides further detail about how the testing should be executed.

The development of test planning documents is a vital component of the testing process. Test planning provides assurance and direction to the test execution phase and helps to ensure that all testing objectives are met. Furthermore, this documentation clearly identifies the testing procedures which are to be followed, ensuring objective results are obtained and consistency is maintained for any regression efforts.

Test planning documents provide the framework for which the web services are to be assessed, including testing assumptions, criteria for test case inclusion, test environment and execution requirements and any limitations of the test strategy. These documents are used as a basis for resource allocation and enable effective project management throughout the testing process.

The test planning documents should be created during the design phase of the Software Development Life Cycle (SDLC), prior to any testing taking place. They should be updated as appropriate when new threats and test cases are developed. Hence, the documents should be stored electronically in an accessible location with a standard versioning system in place to provide easy access for developers, testers and security analysts.

The importance of developing test planning documentation early in the SDLC stems from the importance of following security by design practices as well as the exponentially increasing costs of identifying and fixing security issues late in the SDLC. Upon starting any implementation, regular testing should be conducted rather than being postponed until the latest possible stage.

Retrospective security testing, however, is frequently required given the nature of modern rapid software development methodologies which are often extremely functionality driven. Similarly, it is also needed in the event of an audit-like situation where beta or production systems exist and require testing. Such testing also requires both a test plan and test strategy document for identical reasons to those discussed above.

Test Planning Process

The test planning process comprises two major streams and is outlined below:

- 1) Document Test Strategy
 - a) Create system architecture overview
 - b) Outline high-level testing approach
 - c) Identify high-level testing objectives
 - d) Perform and document business impact analysis
 - e) Describe resource requirements and allocation
 - f) Identify management controls
 - g) Identify defect tracking controls
- 2) Document Test Plans
 - a) Identify test objectives
 - b) Document test-specific dependencies and requirements
 - c) Outline threats and security services being assessed
 - d) Describe test approach and create test cases
 - e) During execution, update test cases to reflect actual testing conducted

Test Strategy

The strategy document should provide an overview of the approach to be adopted for the security testing of the web services components. This strategy should be used to guide project management, develop detailed test plans and to ensure that testing efforts meet business requirements.

General information about the test environment should be included, so as to provide an understanding of testing approach, test case selection and the assumptions that underlie or constrain any testing results. The strategy document should clearly identify high-level test objectives and provide a summary of how priority items impact the business. When defining a test strategy, the analyst should be careful to not include low level/technical details, as it should remain a management-oriented document.

In addition, a test strategy should identify likely resource allocation, required facilities and other requirements for the testing process. The test strategy should include information on the following items, where relevant:

- Web Services overview (systems and environments)
- Architecture diagram
- Trust diagram
- Ownership of testing components, roles and responsibilities
- Business impact analysis
- In / out of scope items
- Assumptions and risks
- Test entry and exit criteria
- Reporting requirements and issue escalation
- Management controls

- Defect tracking method
- Approach to testing for each component or phase (high-level)

Test Plan

Test plans are documents outlining the processes for individual testing scenarios. A test plan is a document containing a logically grouped set of test cases to be executed. The grouping of test cases may be defined by the security service being tested, web service component being assessed or by type of test being conducted. Test cases will typically be conducted in a designated order, as each test case may have a number of environment dependencies or setup requirements. Test areas that should be considered within the plan include:

- Information Gathering
- Fuzzing
- Injection
- Confidentiality & Integrity
- Logging
- Logic Flaws
- Authentication & Authorisation
- Availability
- Ad-hoc Testing

In normal software development testing, test cases are very detailed. Although this is often necessary for providing appropriate software assurance, security testing often does not lend itself to such an approach. Where detail is available it may be provided, yet identification of test objectives is more critical. Greater judgement is required on behalf of the security analyst, than is expected of a software tester.

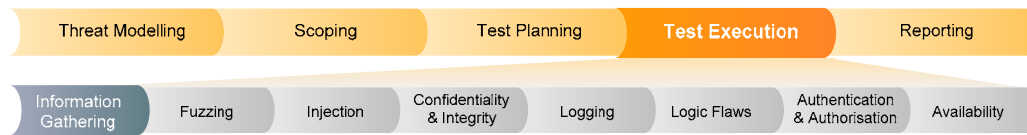
The Information Gathering section of this framework provides further detail on these test areas, and may be used in the development of specific test cases. The identified test areas are not exhaustive, and it is expected that additional test cases will be required to meet specific testing objectives for a given system. Suggestions are made however, to assist in the creation of test cases for unique or proprietary web services.

The test plan should document test cases, enumerated from the Scope Definition document and the Test Strategy document. Test cases can be logically derived from the threat profile and security services to be tested and should describe the aspects of the security service under test.

An overview of the threat being addressed and some example inputs and procedures for assessing the vulnerability of the web service should also be provided. During the actual testing, it is expected that a more rigorous set of test inputs and actions will be developed, documented and used. Test cases should be updated accordingly, to reflect these changes.

A sample test plan template is provided in Appendix C.

INFORMATION GATHERING



Prerequisites:

- Test Strategy Document
- Test Plan Document
- Threat Profile Document

Outcomes:

- Areas of information disclosure identified

The information gathering process aims to identify available web services and obtain detailed information about their operating environment and infrastructure. Information obtained during this stage of testing is required for execution of later test cases, yet it also enables the security analyst to identify other potential weaknesses within the web services environment, requiring further investigation.

The techniques adopted within this section mimic methods used by potential attackers. Typically, scanning techniques are used to expose as much information as is possible about the environment, as grounds for further investigation and/or system compromise. Although the methods described should not interfere with normal business operations, suitable arrangements should be made to minimise any impact on production systems and/or business functions.

WSDL Scanning

Web Services Description Language [W3C 1] is an XML format designed to provide details of exposed web services. The services available and the information required to make use of these services is described in a machine consumable fashion. This allows web services to be dynamically bound to and called by client applications.

The detailed information present in a WSDL document can be an extremely useful source of information for a web services attacker as it provides information about the services themselves as well as required parameter types. On most implementation platforms, the default behaviour is to publish a WSDL document that describes *all* the services available. This can include administrative services as well as test or debug services created by developers which have not been removed and were never meant for public use. Other services of interest may include authentication services or transaction services. The presence of these services in the WSDL document is a clear invitation for attackers to attempt to take advantage of or exploit weaknesses in these services.

When assessing web service security, it is essential to determine the degree of exposure provided by the associated WSDL documents and to ensure that sensitive services or methods are not disclosed. Although there is no guarantee that a web service or method excluded from

a WSDL document will not be called by a knowledgeable attacker, it reduces the likelihood that an attacker can easily identify the existence of these services.

Reference	TE-IG01
Test Case	WSDL Retrieval
Objective	Identify web method call mechanics.
Description	The WSDL document contains information describing the functionality offered by the web service and the parameters required to use it. This information can be invaluable for an attacker as it defines message formats and reveals details about exposed services.
Threat	Information disclosure
Impacted Asset	Server system information
Test Method	<p>Common web service platforms have well known naming conventions for WSDL documents. Attempt to retrieve WSDL documents from URLs including variations on the following:</p> <ul style="list-style-type: none"> ▪ http://webservice.company.com/ServiceName?WSDL ▪ http://webservice.company.com/ServiceName.asmx?WSDL ▪ http://webservice.company.com/ServiceName.cfc?WSDL ▪ http://webservice.company.com/ServiceName.dll?WSDL ▪ http://webservice.company.com/ServiceName.exe?WSDL ▪ http://webservice.company.com/ServiceName.php?WSDL ▪ http://webservice.company.com/ServiceName.pl?WSDL ▪ http://webservice.company.com/ServiceName.m?WSDL ▪ http://webservice.company.com/ServiceName.wsdl ▪ http://webservice.company.com/wsdl/ServiceName.wsdl ▪ http://webservice.company.com/axis/services/ServiceName.wsdl ▪ http://webservice.company.com/axis/services/ServiceName.jws?wsdl

SOAP Fault Error Messages

The SOAP standard has an inbuilt mechanism for describing errors that occur during processing [W3C 3] which is largely analogous to application exceptions. The SOAP Fault element is used to transport error information back to the calling endpoint and contains:

- a code identifying where the error occurred;
- the actor that caused the fault to occur;
- a human readable explanation; and
- application specific details of the error.

Error details are supplied by the application itself and there is a danger that overly verbose error details may provide valuable information for an attacker seeking to learn about the web service. Error details should be limited to the minimum required for support personnel to diagnose and fix problems and should not provide implementation details, platform information or low-level details such as application exception stack traces.

Web services testing should include an assessment of the information leaked via SOAP Fault error messages. The tester should undertake activities that induce error conditions in the application and identify error messages that disclose potentially sensitive information. This can often be performed during fuzzing, as described in a later section. Once certain error conditions are identified, the tester can further refine the error-causing SOAP requests in order to extract further information.

Reference	TE-IG02
Test Case	Error Message Information Leakage
Objective	Assess the extent of information provided by error messages.
Description	Error messages within SOAP Faults can contain detailed platform information and implementation details such as code fragments or stack traces. These error messages can provide an attacker with feedback about system responses to malicious inputs or reveal sensitive information such as hostnames and database connection strings.
Threat	Information disclosure
Impacted Asset	Server system information
Test Method	Attempt to induce errors in the web service application by supplying invalid inputs and observe the web service response. Try to interpret error messages that are returned for information of use to an attacker such as versions, hostnames, application errors, etc.

Web Method Enumeration

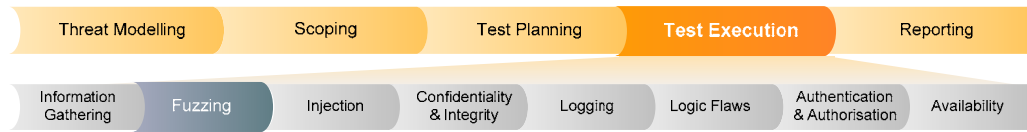
As with any application, developers may leave hidden functions or backdoors in their software. Often the secrecy employed is used as a form of security for administrative tools which are assumed to be known to only those who have the right to use them.

If no directory of web methods such as a WSDL is available for a particular web service, the task of finding “secret” functionality may be reduced to guessing web method names. This is not as impractical as it may appear, as web method names will often be based on their functionality and thus easily deduced.

It should be noted that this tactic may be flagged by an IDS, XML firewall, or honey pot web method [SIFT 1] and result in the denial of further interaction with the web service. Such a response should be noted as part of the assessment.

Reference	TE-IG03
Test Case	Web Method Enumeration
Objective	Identify methods not published in the WSDL.
Description	Not all implemented methods may be published in the WSDL document. It may be possible to guess web method names that may have been deliberately or inadvertently omitted from the WSDL.
Threat	Information disclosure Elevation of privileges
Impacted Asset	Web service functionality Server system information
Test Method	<p>Identifying hidden methods can be done using a list of likely candidates. The following is a list of possibilities which should be expanded to meet specific testing needs:</p> <ul style="list-style-type: none"> ▪ admin, administration, administrator, administrative ▪ test, testing, testmethod, testservice ▪ demo, demonstration, demomethod, demoservice ▪ hidden, private ▪ dev, development, devmethod, devservice ▪ config, configuration, conf, settings <p>If a web method name from the list is called and returns a different response to that of an obviously non-existent method, it may be implemented and warrant further investigation.</p>

FUZZING



Prerequisites:

- Scriptable SOAP generator is available
- Test Plan Document
- Threat Profile Document

Outcomes:

- Common / obvious vulnerabilities are identified

Fuzzing is an automated testing technique for identifying vulnerabilities in data handling. It involves generating pseudorandom data and submitting it in defined fields or parameters in order to evaluate how the web service or supporting systems handle it. Fuzzing is useful in identifying the presence of common vulnerabilities in data handling and the results of fuzzing can lead to a more refined attack on a vulnerable data field.

Fuzzing is automated and it can be tempting to become over-reliant on it for identifying security weaknesses. Although fuzzing is very fast and is an efficient addition to the testing arsenal, it cannot replace the other testing methods identified in this document or other forms of thorough manual analysis. Fuzzing will quickly identify simple issues but it should always be used in conjunction with other security assessment techniques.

Vulnerabilities

The fundamental security inadequacies that fuzzing often uncovers usually stem from the components used by the caller and callee to generate valid messages. For example, a client developed under *C#* generates well formed SOAP structures when it calls some web service. The producers of the web service will thus often assume that all client requests will be well formed and not account for exceptional cases.

Fuzzing should be performed at two logically separate layers of web services: the SOAP layer and the actual application data layer. Additionally, if other standards such as web services security are employed, these can also be targeted.

The following are specific areas in which fuzzing can be used to identify web service security weaknesses. They are presented from most general to most specific and may overlap in objectives and test data. Fuzzing inputs can often be generated by programmatically analysing the WSDL or sample SOAP requests and making modifications to the structure and content of valid requests. Suggested fuzzing data sets are provided in the test cases below.

Reference	TE-FZ01
Test Case	Numerical Values
Objective	Identify mishandling of numerical fields.
Description	Any value that is only valid as a numerical value or is expected to be a numerical value. By supplying unexpected values it may be possible cause buffer overflows or perform other unauthorised actions.
Threat	Denial of Service Information disclosure Elevation of privileges Tampering
Impacted Asset	Web service availability System access System information
Test Method	Depending on the level of knowledge the test data set may be smaller. It is important to test all boundary conditions including those of common type sizes. Usually these would include such things as negative values, zero, powers of two, non numeric values etc.

Reference	TE-FZ02
Test Case	Base64 Encoded Values
Objective	Identify flaws in Base64 data handling.
Description	Base64 is used to encode binary data in order to conform to XML specifications. Depending on how the data is processed and used, unexpected forms can cause unwanted behaviour such as application crashes during decoding or the subsequent storage of invalid data.
Threat	Tampering Information disclosure Denial of Service
Impacted Asset	Web service availability Data System information
Test Method	Modify base64 values to contain characters other than digits, lowercase and uppercase letters, plus and minus. This will test if the decoding mechanism copes well with invalid inputs. Modify base64 values to contain the special character "=". This is a padding character and may cause unexpected behaviour.

Reference	TE-FZ03
Test Case	Character Strings
Objective	Identify mishandling of character strings.
Description	This very broad category provides general guidelines for any data that is not of any particularly classifiable form. It may thus be susceptible to classic problems such as buffer overflows and other parsing errors.
Threat	Tampering Information disclosure Denial of Service
Impacted Asset	Web service availability Data System access System information
Test Method	Excessively long strings can be used for testing for buffer overflows. Strings that have a null in the middle may create strange results under some environments.

Reference	TE-FZ04
Test Case	General Values
Objective	Test the handling of values that have an unspecified format.
Description	If it is not possible to identify the nature of the values being supplied this category provides a general overview of the types of inputs that should be tested. The outcomes of such tests are unpredictable due to the generality of values being used.
Threat	Tampering Information disclosure Denial of Service
Impacted Asset	Web service availability Data
Test Method	In general it is necessary to supply unexpected inputs. If a value appears to be in a certain format, fuzzing "outside" or on the "edge" of the expected format will produce the best results. Usually this will include excessive length, meta-characters etc.

Reference	TE-FZ05
Test Case	Sub-system Parameters
Objective	Identify possible injection flaws in sub-systems.
Description	Certain sub-systems such as database servers are often relied upon by a system. It may be possible to execute injection attacks against those sub-systems and take some form of control.
Threat	Spoofing Tampering Information disclosure Denial of Service Elevation of privileges
Impacted Asset	Web service availability Stored data System access
Test Method	Pinpoint any sub-systems being utilised by the web service. Identify any special characters or character sequences that may cause errors or obvious deviations in execution flow. Attempt to induce injection by supplying the documented special characters.

Reference	TE-FZ06
Test Case	Output Values
Objective	Assess the security of the data output mechanism.
Description	This category relates to any values that may be used to influence output on the client side of the application. It primarily seeks to identify cross-site scripting vulnerabilities but should still be analysed for applicability on a case by case basis if browsers are not used.
Threat	Spoofing Information disclosure Elevation of privileges
Impacted Asset	Credentials Data
Test Method	If a custom display format is used, any special characters used by the format should be tested. In the generic web browser case, the special html characters such as <, >, ' and " should be tried. If output is rendered incorrectly, then a client side attack may be possible.

Reference	TE-FZ07
Test Case	Addressing Parameters
Objective	Assess the possibility of accessing resources outside those intended.
Description	Systems often use addressing information to access information directories. Common types of addresses such as those employed by URIs, XPath and LDAP might be open to manipulation by supplying in-line addressing information in variables.
Threat	Tampering Information disclosure
Impacted Asset	Data System access
Test Method	Identify any data addressing schemes that may be implemented by the web service. Test any fields that may be used directly in address construction by providing addressing information inside those parameters. If the call returns an access violation or a message specifying the particular resource does not exist, addressing injection may be possible.

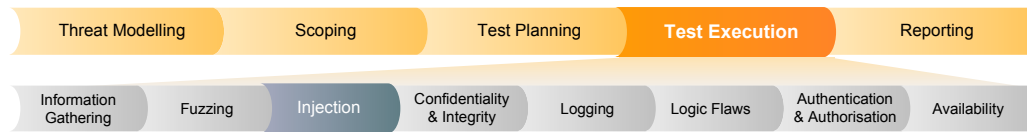
Reference	TE-FZ08
Test Case	Tokens
Objective	Identify simple session token weaknesses.
Description	Because of their importance as both an authentication and authorisation mechanism, fuzzing of session tokens is very important. Consequences include unauthorised access to functionality and the hijacking of accounts whose sessions have been compromised.
Threat	Spoofing Repudiation Elevation of privileges
Impacted Asset	Tokens
Test Method	If a valid session token is available it may be worthwhile to simply increment it continually for a period of time checking if it presents a valid connection. Some other useful values may include nulls and zeros. Useful malformed values may include shorter tokens, longer tokens, and white space.

Reference	TE-FZ09
Test Case	Format String Parameters
Objective	Ascertain the web service's susceptibility to format string attacks.
Description	This category only applies to applications written in languages such as C/C++ that use unvalidated format functions such as the printf family. With the ability to modify the format specifier it may be possible to execute arbitrary code.
Threat	Tampering Information disclosure Denial of Service Elevation of privileges
Impacted Asset	Web service availability Data System access
Test Method	A series of "%x" or "%s" specifiers will provide the best results as the application will crash or show portions of memory if a successful exploitation has taken place.

Reference	TE-FZ10
Test Case	Logging Values
Objective	Assess the logging mechanism's handling of simple attacks.
Description	Any value that is logged directly to some medium has the potential to somehow corrupt logs or provide an inaccurate view of what actually occurred. Fuzzing values that influence formatting are of particular importance.
Threat	Tampering Repudiation
Impacted Asset	Log files
Test Method	This test data is highly dependant on the logging mechanism being used. If unknown it may be useful to try inserting carriage returns, line feeds, null values, excessive spaces, and unprintable characters.

Reference	TE-FZ11
Test Case	File Names
Objective	Identify directory traversal vulnerabilities.
Description	Due to the common presence of canonicalisation issues in file access, fuzzing values that might be subsequently used for file access has become important. If no filtering is done, it may be possible to traverse the directory structure and gain unauthorised access to data.
Threat	Tampering Information disclosure
Impacted Asset	Stored data System information
Test Method	The most important value in this case is the "../" which moves the relative directory back one step and could be used to induce directory traversal. Certain file systems may allow null characters in filenames that can be used to bypass filename validation routines.

INJECTION



Prerequisites:

- Test Plan Document
- Threat Profile Document

Outcomes:

- A list of verified injection vulnerabilities present in the web service

Web services are typically used within the enterprise application integration and business to business sectors as they provide a heterogeneous, platform-independent mechanism for functionally integrating distributed applications. Individual web services can be extremely complex, crossing multiple application domains and interacting with a variety of systems. Thus, it is vital that these web services are resistant to injection attacks when external systems are accessed or invoked.

Injection attacks involve the subversion of a subsystem to perform actions beneficial to the attacker. By directly passing input parameters to a subsystem query or other construct that has some special structure, special characters may be used to change that construct. When the subsystem subsequently uses this as a basis of performing some action, execution may take an unintended path and have unexpected outcomes.

The most prevalent injection vulnerabilities include SQL injection, command injection, LDAP injection, XPath injection, and code injection. There are undoubtedly many other forms of injection and the tester should be aware of these for testing of different subsystems. The important point is that the subsystem input handling should be analysed and subjected to test cases that assess the effectiveness of input validation processing.

Most injection vulnerabilities can be easily and quickly identified by the process of fuzzing due to the presence of meta-characters in various language syntaxes. Additionally, verbose error messages are often produced if such vulnerabilities are present. Although automated fuzzing tools are available, results should be manually verified to reduce false positives. Subtle vulnerabilities where some validation or sanitisation occurs may be more difficult to identify. In this situation, a more thorough analysis is required and typically involves a more refined construction of inputs in an attempt to bypass validation and sanitisation mechanisms. The remainder of this section seeks to explain the general techniques that can be applied to most web services to identify these hidden vulnerabilities or verify results produced by the fuzzing phase.

Analysing Web Method Responses

The responses provided by the web methods being tested will be used to determine whether a vulnerability exists or if further investigation is required. In general if the method call returns a HTTP status code other than 200, further investigation will be required. The 500 error code is the predominant indicator of inadequate input validation. If a response contains a <soap:Fault>

element, it will likely provide an indication of the attempted injection and in some cases, the specific query or other construct that was tampered with will be revealed. If an injection condition exists but good exception handling is employed, the response may contain a 200 status code in addition to an error message or even a null (or nil) return value. It is vital to identify any deviation from normal or expected responses as it may indicate the presence of a vulnerability.

SQL Injection

SQL injection occurs when a database is queried with an SQL statement which contains some user influenced input that is outside the intended parameter range. If this occurs, an attacker may be able to gain control of the database and execute malicious SQL or scripts.

In the standard case, SQL injection is identifiable by supplying a single quote in every possible parameter via an attack proxy. In some cases this may not be sufficient – for instance, if the quote character is escaped but SQL injection is still possible in an integer field. The prudent tester should therefore test some SQL reserved words. If successful, this will create a syntactically invalid SQL statement and cause an error or exception to occur.

The tester should not assume consistency of parameter handling throughout the web service. Each parameter should be tested in isolation, ensuring other parameters have been assigned valid values. In this way, each parameter can be tested individually without interference from other bad parameters. Some parameters may be validated properly and halt execution whereas others may prove to be vulnerable.

It may also be possible that a combination of specific parameter values may be required for a successful SQL injection attack. If this is the case, the testing for SQL injection becomes very much implementation specific and will differ greatly between web services. The tester must take care so as to fully understand the web service and underlying logic structures, in order to test as many probable vectors as possible.

Once the existence of an SQL injection issue has been established, the tester may wish to explore the extent of the problem. Several discussions on such topics are available including [SQL 1], [ANL 1] and [CER 1].

Reference	TE-IN01
Test Case	SQL Injection
Objective	Identify SQL injection vulnerabilities.
Description	Any value that may be used as part of an SQL query should be tested for the ability to change SQL processing in some way, possibly causing data disclosure, modification, removal, or sometimes compromise of the SQL server.
Threat	Spoofing Tampering Information disclosure Denial of Service Elevation of privileges
Impacted Asset	Web service availability Stored data System access
Test Method	While this will not give an indication of the exact extent of vulnerabilities, using the single quote as a test value should identify most vulnerabilities.

Command Injection

Command injection is the manipulation of commands passed to the operating system or other external systems to perform a function as specified by an attacker. In general it occurs when an operating system utility is used by the web service to perform a certain function without validating the input to that function.

For instance, if the application uses the ping binary of the operating system and passes a user specified IP address directly to the binary it may be possible to take control of that execution. In this example the command passed to the operating system may be:

```
"ping <user_supplied_ip>"
```

and abused by specifying

```
"ping 127.0.0.1; useradd test -p GXX7d1LBXab6k"
```

which would cause the *test* user account to be added to the system.

Exhaustive testing of command injection may not always be practical. The tester should look for candidate parameters that are likely to be used as part of a command. It is therefore important to be aware of the types of functions that the operating system can provide.

Many modern operating systems support command line special characters that allow the execution flow of a command to be modified. The specific characters for the operating system underlying the web service implementation should be identified and used in testing.

The extent to which command injection will allow control over the external system is highly dependant on the configuration of that system. The tester may attempt to execute binaries available to enumerate file access control lists and other system settings. Depending on the level of popularity, further resources should be available on the system manufacturer's website and other online databases.

Reference	TE-IN02
Test Case	Command Injection
Objective	Discover any command injection vulnerabilities that may exist.
Description	If an external system is used to execute existing commands and input to these commands is not properly validated, it may be possible to run commands of the user's choosing.
Threat	Tampering Information disclosure Denial of Service Elevation of privileges
Impacted Asset	Web service availability Stored data System access
Test Method	The standard set of potentially dangerous special characters that could be used as a starting point for testing: <ul style="list-style-type: none"> ▪ ";" is a command separator that can be used to start a new command ▪ " " is the output stream piping character that can be used to start a new command ▪ "<" and ">" are the input and output redirectors that can be used to pipe input to a new command ▪ The carriage return and line feed can often be used as command separators ▪ ".", ",", "/" and "\" which can be used to induce directory traversal possibly enabling an attacker to execute a command other than that which is intended.

LDAP Injection

The Lightweight Directory Access Protocol (LDAP) uses queries to access data. In a similar fashion to SQL, if queries are composed from user input without adequate validation or sanitisation, the risk of an injection attack may exist. The difference between LDAP search queries and SQL queries is that LDAP strings are usually in the form of a URL [HOW 1].

Due to the fact that LDAP queries are in URL format and no quotation is required, all LDAP special characters should be tested for injection. It may also be possible to use the URL encoded equivalents of the characters.

As individual characters are tested, a list of allowed characters can be established. Once this is done, the tester may perform further exploitation and enumeration. Detailed exploitation techniques are documented in [FAU 1] which can be used in combination with the protocol specification [RFC 2251] to continue deeper testing.

Reference	TE-IN03
Test Case	LDAP Injection
Objective	Assess the web service's susceptibility to LDAP injection.
Description	If LDAP queries are constructed directly from user input, this may result in significant system compromise, particularly in the disclosure of user credentials. Testing LDAP special characters is therefore a critical process.
Threat	Spoofing Elevation of privileges
Impacted Asset	Credentials
Test Method	<p>Identifying vulnerable LDAP parameters involves sending LDAP special characters to the web service and observing if errors occur. The base list of test characters is provided below along with their common LDAP usages.</p> <ul style="list-style-type: none"> ▪ [%26] & = and ▪ [%7C] = or ▪ [%21] ! = not ▪ [%7E] ~ = approx equal ▪ [%3E] > = greater than ▪ [%3C] < = less than ▪ [%2A] * = any ▪ [%3F] ? = new parameter ▪ [%28] (= start grouping ▪ [%29]) = end grouping ▪ [%3D] = = equal

XPath Injection

The XML Path Language (XPath) is a language for accessing elements of an XML document [W3C 5]. XPath queries may be dynamically constructed from user inputs and if these inputs are not properly validated or sanitised, the resulting query may be vulnerable to injection and allow access to unauthorised parts of the XML document or database. Specifically, XPath special characters may be used to manipulate the nodes or attributes that are referenced by the application.

Similar to SQL injection, XPath injection is most easily detected by supplying the single quote and double quote character to each web service parameter one at a time, and assessing the responses obtained. Fields vulnerable to XPath injection will cause the application to construct a syntactically invalid XPath query, due to the addition of the quote character, and an error will occur when the query is processed.

As with other forms of injection testing, it may be necessary to supply valid values for other parameters whilst individually testing a specific parameter. Furthermore, particular processing may only occur if inputs meet certain conditions but this is largely implementation specific and must be individually assessed for each web service.

If an XPath injection is detected, it may be useful to determine the extent of the vulnerability. An excellent reference on this topic is [KLE 1] which covers blind XPath injection attacks.

Reference	TE-IN04
Test Case	XPath Injection
Objective	Identify XPath injection vulnerabilities.
Description	The use of user supplied input in an XPath query may provide an attacker with the ability to modify the query. All inputs to XPath query strings must be tested with XPath special characters to determine if the application is susceptible.
Threat	Information disclosure
Impacted Asset	Stored data
Test Method	In most cases the single quote and forward slash characters are sufficient to identify XPath injection vulnerabilities as errors or empty results will occur if XPath injection is possible.

Code Injection

Code injection is an attack on a web service which allows the attacker to fool the application into executing code of the attacker's choosing. This form of vulnerability is more prevalent in standard web applications but remains a threat for web services. Code injection attacks exploit the processing of evaluation constructs that include unsanitised user input and can be used to execute commands on the server system.

An example of flawed code may be:

```
eval("for( $i = 1; $i < $userinput; $i++) ... ");
```

which could allow a user to supply their own statements to be executed by the language interpreter.

It is possible to take an (almost) blanket approach to testing for code injection flaws. However, some programming languages which do not have a scripting-based history will not be vulnerable at all. Some common languages that have *eval* constructs that allow string data to be interpreted as code include:

- PHP
- Perl
- Python

Supplying random symbols that create an invalid statement may cause the web method to respond with an error or unusual result. If this occurs further investigation may be warranted to differentiate code injection from other forms of injection. This may require the tester to have knowledge of the language in use, in order to construct valid statements.

Reference	TE-IN05
Test Case	Code Injection
Objective	Identify code injection vulnerabilities.
Description	If unvalidated user supplied input is supplied to calls to <i>eval</i> -type functions, malicious commands may be inadvertently executed by the web service. It is vital to ensure that all inputs to <i>eval</i> -type functions are properly validated and sanitised.
Threat	Tampering Information disclosure Elevation of privileges
Impacted Asset	Data System access Network resources
Test Method	As each language has its own specific syntax, the language used in the web service implementation must be identified. Once identified, language constructs and special characters should be supplied to the set of potential fields. Some examples of these include: <ul style="list-style-type: none"> ▪ String specifier characters – commonly the single quote (') or double quote (") characters ▪ Language reserved words – eg. for, while, if, else ▪ Brackets – regular brackets (), square brackets [] and curly braces {} ▪ Mathematical symbols – eg. +, -, *, /

XML Injection

XML injection is the process of inserting elements or attributes into an XML document as a result of improper input validation. The insertion of elements or attributes may allow an attacker to sufficiently alter the structure of the XML document such that subsequent processing is affected and application behaviour is modified.

Simple API for XML (SAX) parsers in particular may expose XML injection vulnerabilities in a web service as they treat an XML document as a unidirectional stream and conduct validation at the same time as processing. XML injection may allow an attacker to effectively overwrite previous nodes in the document or re-initiate processing with arbitrary supplied values. Depending on the implementation and subsequent processing, this may lead to tampering issues that can result in privilege escalation or spoofing attacks.

XML injection is most easily identified by supplying angle bracket or quote characters to input fields. If the application is vulnerable to XML injection, these inputs will result in malformed XML that should manifest as an error. However, these malformed XML documents may not provide immediate feedback if they are stored for later asynchronous processing and it may be necessary to obtain access to internal data stores to verify the results of these test cases.

Another important test value is the `<![CDATA[...]]>` tag that directs the XML parser to ignore its contents [W3C 6]. This allows any character string to be contained within an element and may provide an avenue for XML injection as special characters will be left effectively untouched and may be inserted into a resulting XML document.

Reference	TE-IN06
Test Case	XML Special Characters
Objective	Identify weaknesses in the handling of special characters within inputs to XML documents.
Description	The use of user supplied input in the construction of an XML document may allow the document structure to be manipulated if inputs are not properly validated. Such changes to the XML structure may cause unexpected consequences when the document is processed.
Threat	Tampering Denial of service
Impacted Asset	Web service availability Data
Test Method	<p>XML injection is commonly identified by supplying special characters such as angle brackets and quotes to input fields that are embedded into XML documents. Entity reference values for these characters should also be tested to assess indirect XML injection and these include:</p> <ul style="list-style-type: none"> ▪ &lt; ▪ &gt; ▪ &amp; ▪ &quot; ▪ &apos; <p>Web services vulnerable to XML injection will typically respond with an error when they attempt to process the malformed XML document. In the case of indirect injection, it may be necessary to obtain access to internal data stores or logs to determine if higher order injections are possible.</p>

Reference	TE-IN07
Test Case	XML CDATA Sections
Objective	Determine vulnerabilities resulting from improper handling of CDATA sections.
Description	This test case is similar to TE-IN06 but utilises the XML CDATA section as an attack vector. CDATA sections in XML documents provide the ability to automatically escape special characters. This may allow XML special characters to pass through within input values. These values may then be used as inputs to subsequent XML documents potentially resulting in a higher order XML injection vulnerability.
Threat	Tampering Denial of service
Impacted Asset	Web service availability Data
Test Method	Supply XML special characters such as angle brackets and quote characters within CDATA fields and assess the response of the web service. Web services vulnerable to XML injection will typically respond with an error when they attempt to process the malformed XML document. It may be necessary to obtain access to internal data stores or logs to determine if the web service is vulnerable.

Other Injection

Other types of injections may be difficult to identify using a black box testing approach. The tester must first identify the subsystem being used and then proceed to further analyse injection possibilities. If numerous components are interacting and no information is provided about the system, such analysis may be near impossible.

However, in almost all cases an injection can be revealed by providing the subsystem special characters or combinations thereof to elicit erroneous or unusual responses. It will be up to the tester to identify the characters considered "special" by the subsystem.

CONFIDENTIALITY & INTEGRITY



Prerequisites:

- Test Plan Document
- Threat Profile Document

Outcomes:

- Issues with confidentiality and integrity mechanisms identified

Confidentiality in a web services context provides the ability for client-intermediary-server communications to occur without eavesdropping by unauthorised parties. Integrity provides the ability for the communications to be protected against tampering by unauthorised parties. These features are typically critical in the web services environment because communications commonly travel through the untrusted Internet zone. Furthermore, web service communications are often of a sensitive business nature and may involve financial transactions or information exchange between business partners. Thus it is essential that communications are protected and this is conventionally achieved through the use of encryption ciphers and message integrity codes.

Confidentiality and integrity mechanisms for web services are often implemented in different ways, often using custom cryptography which is likely to be flawed. However, standards for ensuring security properties in web services communications are now available and their use is encouraged. Specific web service implementations should be tested for adherence to these standards as well as other best practices involving secure data transfers. In addition, it is important to subject the communications protocols used by the web service to a rigorous cryptographic analysis.

Confidentiality

Several encryption ciphers are widely available including the Advanced Encryption Standard (AES), Triple Data Encryption Standard (Triple DES) and Rivest Cipher 4 (RC4). General purpose secure communication protocols such as Secure Sockets Layer (SSL) and Transport Layer Security (TLS) also exist which utilise encryption ciphers like AES and RC4 in accordance with best practices. These encryption ciphers and secure protocols are open and have undergone extensive analysis by the security community. On the other hand, proprietary encryption algorithms and protocols are often flawed and should not be relied upon as they have not been thoroughly examined and tested by the wider community. Thus, the choice of encryption technology for the application is an important element and the strength of the chosen cipher must be appropriate for the information being protected.

The security of an encryption cipher resides in properties of the secret key used for decryption and the key management process. As a result, the key generation, key storage and key distribution processes must be scrutinised for weaknesses that may yield secret keys to an

attacker. Furthermore, the choice of a strong cipher or secure protocol is no guarantee of security and it is essential that the method in which it is utilised is considered. It is especially important to assess the coverage of the encryption used to protect the communication channel - specifically, what items are encrypted - as seemingly innocuous unencrypted items often provide information that can aid an attacker. Additionally, encryption technologies only protect against external eavesdroppers and provide no protection against application level threats such as SQL injection.

Although cryptanalysis techniques are beyond the scope of this paper, they may be required if a proprietary encryption cipher is being tested. Implementations that make use of established protocols like SSL and TLS are also susceptible to problems if used incorrectly. SSL/TLS parameters such as key lengths and algorithms must be chosen appropriately for the specific web service under test and the latest version of the SSL/TLS protocol should be chosen. At the time of writing the current versions are SSLv3 and TLSv1 and the commonly accepted settings are 128-bit keys and 3DES, RC4 or AES for the encryption algorithm but this is subject to future change. A common list of confidentiality test cases applying to web services are presented below:

Reference	TE-CI01
Test Case	Cipher Choice
Objective	Identify the encryption cipher used in the application.
Description	The choice of encryption cipher will influence the strength of the encryption and the ability for an attacker to successfully crack the encryption and recover plaintext data.
Threat	Information Disclosure
Impacted Asset	Data
Test Method	Determine the cipher used from the code, design or discussion with developers. This procedure may be difficult in a black-box testing environment. Assess the suitability of the chosen cipher to the data being protected.

Reference	TE-CI02
Test Case	Encryption Coverage
Objective	Determine items within web service communications that are encrypted.
Description	Encryption should be applied over all sensitive portions of messages to ensure they are protected against unauthorised eavesdropping. Failure to provide adequate coverage over the message contents may allow an eavesdropper to read parts of the message or infer its contents.
Threat	Information Disclosure
Impacted Asset	Data
Test Method	Attempt to extract usable information from unprotected portions of the communications stream and analyse what is disclosed. Significant items to look for include message integrity check fields, signatures, timestamps and nonces as these can disclose information about the data contained or other

protections in place.

Specifically, if the XML Encryption standard [W3C 7] is utilised, identify the data included within the <xmlenc:EncryptedData> tags and the remaining unencrypted data within the SOAP body.

In addition to these test cases, a number of confidentiality issues may require consideration under certain implementations. The questions below can be used as a starting point of investigation if the implementation being assessed demands it:

- How are encryption and decryption keys generated?
- How are encryption and decryption keys stored?
- How are encryption and decryption keys negotiated or distributed?
- Is there any statistical bias in the keyspace?
- How large is the keyspace?
- Are encryption and decryption keys cycled regularly?
- Is the cipher vulnerable to timing attacks?

Integrity

Message integrity checks are normally based on digital signatures or cryptographic hash functions or both. Many open schemes are available and have undergone intense scrutiny by the wider security community. The choice of message integrity check code should be evaluated to ensure that it provides the required level of assurance that the message has not been tampered with.

Integrity checks are typically keyed and the secret key issues noted in the confidentiality section also apply to integrity. The use of integrity checking mechanisms in the web service must also be examined to ensure that they meet the requirements of the application.

Test cases include:

Reference	TE-CI03
Test Case	Replay Attacks
Objective	Determine the susceptibility of the communication protocol to attacks on the 'freshness' of messages.
Description	A replay attack involves the malicious use of a valid message or set of messages that has already been accepted by the web service previously. Although the messages may be protected against eavesdropping and tampering, the attacker is not interested in determining the contents of the message, but only in re-initiating the action that the message represents.
Threat	Spoofing Repudiation
Impacted Asset	Credentials Data
Test Method	Capture and replay previous messages to the server to test whether or not the application accepts them. Attempt to repeatedly perform important actions such as financial transactions or authentication exchanges. Check for the presence of nonces, timestamps or other measures of 'freshness' in SOAP

	messages and assess the ability to manipulate or bypass them. Confirm that 'freshness' measures are protected against tampering.
--	--

Reference	TE-CI04
Test Case	Integrity Check Coverage
Objective	Determine the items within web service communications that are protected by message integrity checks.
Description	Integrity checks should be used to protect important data against unauthorised modification. Failure to provide adequate coverage over message data may allow messages to be sufficiently manipulated by an attacker such that the meaning of the message is changed.
Threat	Tampering
Impacted Asset	Data
Test Method	Determine the elements contained within integrity mechanisms and the remaining unprotected elements within the SOAP body. Attempt to alter communications or application flow by tampering with unprotected parts of messages. Specifically, if the XML Signature standard [W3C 8] is employed by the web service implementation, determine the contents of the <xmldsig:Signature> tags.

Further to the above, the following Integrity issues may require further consideration:

- What items remain susceptible to tampering?
- Is the integrity check code protected?
- How are the integrity check keys generated?
- How are the integrity check keys stored?
- How are the integrity check keys distributed?
- Is the integrity check code susceptible to collisions?

Web Services Security Extensions

The Organisation for the Advancement of Structured Information Standards (OASIS) has been continuously developing and updating e-business standards since 1993. Among the standards that have been recently developed is the Web Services Security (WSS) suite that includes the WSS: SOAP Message Security (WSSSMS) standard [OASIS 2]. The WSSSMS specification was designed to be used for securing web services communications – specifically SOAP messages – and to support a wide array of security primitives and technologies.

WSSSMS provides three main features:

- A means to transfer security tokens
- A means to ensure message integrity
- A means to ensure message confidentiality

These features allow web services to exchange SOAP messages in a secure fashion but do not address other security issues such as authentication and non-repudiation. These issues must be addressed using other security standards or application level mechanisms. Furthermore, the WSSMS specification does not describe explicit protocols but focuses on defining a standard interoperable syntax for implementing tokens, integrity and confidentiality in SOAP. Thus, it remains necessary to assess the implementation of a web service that utilises WSSMS to ensure that the cryptographic elements utilised provide the confidentiality and integrity required by the application.

Additional extensions to the WSS standards have been developed and include WS-SecureConversation [ACT 1], WS-SecurityPolicy [IBM 1] and WS-Trust [ACT 2]. These standards define additional security constructs to enable the expression of security policy and the establishment of secure sessions for web service communications. Once again, these standards do not dictate explicit protocols so implementations must be evaluated for security weaknesses.

To assess the cryptographic strength of a WSS communications protocol, it may be required to decompose the verbose XML SOAP messages to basic protocol elements. These can then be analysed in an identical fashion to that described above. However, WSS introduces a new area of testing that is required – compliance with the specification.

Compliance with the OASIS WSS specification must be assessed to ensure that the application will interoperate with other WSS compliant applications. Compliance testing involves two main areas: XML validation and application behaviour. XML validation is straightforward - the SOAP messages produced by the application must conform to the WSS standard and this can be verified by validating the SOAP message output against the appropriate XML schemas. It is important to validate all possible message outputs from the application to ensure that all communications comply.

The second area of testing WSS compliance is to verify the behaviour of the application against the behaviour described in the WSS specification document. This typically involves testing the communications component with various messages and assessing the application's behaviour in response to it. Some message test cases include:

Reference	TE-CI05
Test Case	Invalid XML
Objective	Assess the response of the web service to malformed XML inputs.
Description	WS-Security and other web service security standards are XML-based and their implementations require properly formed XML to function properly. The ability of these implementations to handle malformed XML needs to be determined to ensure that the malformed messages do not induce behaviour that does not comply with the specifications of the standard.
Threat	Tampering Information Disclosure Denial of Service
Impacted Asset	Web service availability Server system information
Test Method	Supply the web service with valid SOAP requests that do not comply with the WS-Security schemas to determine the response of the application. Verify that WS-Security compliant SOAP requests are required for the application to service requests.

Reference	TE-CI06
Test Case	XML Canonicalisation
Objective	Determine the effectiveness of XML canonicalisation in the web service.
Description	XML documents must be canonicalised before operations such as encryption or signing are performed because information can exist as an XML document in multiple equivalent forms [W3C 9].
Threat	Tampering
Impacted Asset	Data
Test Method	Generate XML documents that are logically equivalent and verify that they are treated identically by the web service. Test XML differences such as: <ul style="list-style-type: none"> ▪ Attribute ordering ▪ Encoding schemes ▪ Line break characters ▪ White space ▪ Special characters ▪ Entity references ▪ Empty elements

Reference	TE-CI07
Test Case	Unsupported Algorithms
Objective	Assess the response of the web service to requests for unsupported algorithms within WS-Security, WS-SecurityPolicy and other security standards.
Description	Verify that if unsupported algorithms are requested or the client claims to not support required algorithms, access is denied and processing of the request does not continue.
Threat	Tampering Information Disclosure
Impacted Asset	Data Server system information
Test Method	Generate valid XML SOAP messages that request or use unsupported algorithms for encryption or signatures. Attempt to force the web service to accept unsupported algorithms or revert to an open mode of operation.

Reference	TE-CI08
Test Case	Failed Policy Requirements
Objective	Assess the web service response to SOAP messages that do not meet security policy requirements.
Description	WS-SecurityPolicy provides messaging formats for exchanging additional security policy information such as security token types, visibility requirements

	and message age requirements.
Threat	Tampering Information Disclosure
Impacted Asset	Data Server system information
Test Method	Generate valid XML SOAP requests that do not satisfy security policy requirements. Attempt to force the web service to accept requests that are below minimal policy requirements.

LOGGING



Prerequisites:

- Test Plan Document
- Threat Profile Document

Outcomes:

- A list of logging issues identified

Logging functionality is a widely overlooked area of application testing but from a business perspective it is critical. If a system is compromised or some other failure occurs, logs can be the only way to track down the root cause and identify the perpetrator.

Testing logging in web services is quite similar to testing the same in any other form of application. Firstly the logging mechanism must be characterised, followed by testing and analysis. Often the testing will involve injecting special inputs, overflowing logs, and enumerating and analysing actions logged [SIFT 2].

Characterisation

In order to conduct a security analysis of logging mechanisms, they must be well understood. Several questions must be answered to produce a thorough assessment.

What medium is being logged to?

The most common medium to log to is a file on the local hard disk however this is not the most secure option and will often be replaced by some other medium. This might include write once media such as a CD, networked file or even a printer. Each medium has different inherent characteristics that could mean the data is analysed in different ways.

What format will the logs be in?

The formatting of logs can often be abused by attackers and therefore a great deal of information is required by the tester to properly analyse the security of logging mechanisms. How detailed is the information being logged? What individual pieces of information are being stored? Are log entries being written in any specialised format such as XML, HTML, syslog, eventlog? Is there any special formatting applied to the logs?

What is the standard method of viewing the logs?

In the end logs are only as effective and useful as how they are viewed by an analyst. If an analyst cannot gain any useful information from a log then it is a frivolous function. Understanding the usual method for examining log entries could provide an insight on how to make the analyst's job difficult or even impossible through log manipulation.

Logging Issues

Given the wide variety of logging methods available, a number of test cases are presented below. The web service being tested should be characterised and the relevant test cases executed, with appropriate modifications to account for implementation specifics if necessary.

Reference	TE-LG01
Test Case	Separator Injection
Objective	Assess the ability of the web service to log messages that contain special separator characters.
Description	Log entries are commonly delimited using a particular separator character. If an attacker can insert these special characters into elements that make up part of a log entry, it may be possible to create false or misleading entries.
Threat	Tampering Repudiation
Impacted Asset	Log files
Test Method	The tester should aim to poison the log with separator characters in order to place logged values under sections where they do not belong. Common separator characters include: <ul style="list-style-type: none"> ▪ Tabs ▪ Pipes ▪ Spaces ▪ Commas

Reference	TE-LG02
Test Case	White Space Injection
Objective	Determine the logging mechanism's susceptibility to white space injection.
Description	White space characters can be used to modify the appearance of log entries when they are viewed. If white space can be inserted into log entries, important information may be shifted past the right hand edge of the viewing application or new lines created if word wrap is enabled. This may allow an attacker to hide or obscure log entries that contain information about their activities. In the case where word-wrap is enabled in the viewing application, additional log entries may be created via an overly long entry and may contain information that negates the suspicious nature of previous entries.
Threat	Tampering Repudiation
Impacted Asset	Log files
Test Method	Using white space characters such as the tab and space, supply overly long messages which will be used in logging. After the messages have been processed, examine the generated logs to identify how the white space was handled.

Reference	TE-LG03
Test Case	XML Injection
Objective	Assess the possibility of inserting XML elements and/or attributes into an XML based log.
Description	XML elements or attributes included within data fields that are logged may affect the resulting structure of the XML document. This could be manipulated by an attacker to hide or obscure log entries that provide details of their activities. It may also allow an attacker to create bogus log entries to complicate investigations involving the manipulated log.
Threat	Tampering Repudiation
Impacted Asset	Log files
Test Method	The tester should attempt to pass XML special values such as the less than "<" and greater than ">" signs to the web service. If these values are written directly, it will likely be possible to change the structure of the log file. If this is the case, an attacker could almost totally control the logged events.

Reference	TE-LG04
Test Case	HTML Injection
Objective	Assess the possibility of inserting HTML tags into a HTML based log.
Description	The insertion of HTML tags into a HTML log that is rendered provides an attacker with multiple opportunities. It may be possible to modify the structure of the HTML document to hide or obscure incriminating log entries. Additionally, it may be possible to actively exploit the rendering application through a browser vulnerability, resulting in the compromise of the log monitoring terminal.
Threat	Tampering Repudiation
Impacted Asset	Log files
Test Method	Supply HTML special characters to web methods that are likely to log the input. This test is similar to cross-site scripting attacks and thus the same inputs should be tried. These would include supplying quotes and the less than and greater than symbols. Follow this with an inspection of the logs to determine if the attack was successful.

Reference	TE-LG05
Test Case	New Line Injection
Objective	Determine whether the logging mechanism is vulnerable to arbitrary entry creation via carriage return and line feed injection.
Description	In systems where log entries are based on single lines there is a threat of an attacker injecting new lines and creating a new entry. The insertion of new lines into a log entry may allow the creation of forged log entries which nullifies the ability for administrators to rely on the manipulated log.
Threat	Tampering Repudiation
Impacted Asset	Log files
Test Method	Use the carriage return, line feed, and combination of both as input to a web method that is logged. This should be followed by a review of the log files to determine if new lines were created.

Reference	TE-LG06
Test Case	Size Overflow
Objective	Assess the handling of log data after reaching the upper log size limit.
Description	There is always an upper bound on how much information can be logged at any one time. This limit may be set by an administrator or may be a direct result of the capacity of the medium being used to store information. When a log reaches its maximum capacity, old log entries may be overwritten, processing may cease or new entries may be lost. This may cause logs to be incomplete and unreliable.
Threat	Tampering Repudiation
Impacted Asset	Log files
Test Method	Repeatedly supply data to web methods that log their inputs until the log has reached its capacity. Once past capacity, analyse the resulting behaviour of the logging mechanism.

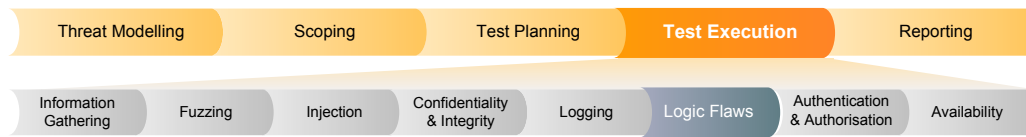
Reference	TE-LG07
Test Case	Information Disclosure
Objective	Determine the possibility of retrieving or otherwise revealing information contained within log entries.
Description	Logs often contain detailed information regarding implementation details, user accounts and backend system commands which are extremely valuable to an attacker. If log contents can somehow be disclosed then the sensitive information disclosed can assist attackers in finding weaknesses in the web service application.
Threat	Information Disclosure
Impacted Asset	Log files Server system information
Test Method	No standard method is available for this test case. The tester should attempt to analyse the processes employed in logging functionality and use them to gain access to log entries where possible. Logs and log entries should be tested in transit and in storage and attempts made to determine their contents.

Reference	TE-LG08
Test Case	Alternate Data Streams
Objective	Assess the ability to manipulate alternate data streams in logging facilities.
Description	Alternate data streams are a Windows NT File System (NTFS) feature introduced for compatibility with the Macintosh Hierarchical File System (HFS) and can be used to hide files and file contents from the user.
Threat	Tampering Repudiation
Impacted Asset	Log files
Test Method	If the log file naming is dependant on user input, attempt to supply the colon character to the web method used to create log files and determine if log entries are written to an alternate data stream file.

Reference	TE-LG09
Test Case	Not Logged Actions
Objective	Verify that the level of logging in place is sufficient.
Description	Different web services require different levels of logging to be implemented and these different levels of logging should be tested to ensure that the implementation satisfies security requirements. Inadequate levels of logging may allow attacks to go undetected or be difficult to investigate.
Threat	Repudiation
Impacted Asset	Log files
Test Method	Attempt to use web method calls to cause events which are deemed important and may require logging. Subsequently inspect logged data in order to identify if sufficient information was retained. Determine which actions are logged by the web service and the extent of information written to the log.

Reference	TE-LG10
Test Case	Logic Flaws
Objective	Identify any logic flaws that could cause some actions to be logged incompletely or not at all.
Description	Verbosely logging all web method calls is often not practical and therefore logic is often implemented to log only in certain circumstances. It may be possible to abuse this logic to obstruct full logging of malicious activity.
Threat	Repudiation
Impacted Asset	Log files
Test Method	Attempt to identify the logic used to filter logged events and attempt to abuse this to the tester's advantage. Try to exploit logically flawed implementations to bypass or subvert the logging of potentially malicious activity.

LOGIC FLAWS



Prerequisites:

- Test Plan Document
- Threat Profile Document

Outcomes:

- Logic issues identified

This section is devoted to testing that cannot be grouped under any other section. It is extremely application specific and may be considered as ad-hoc testing. The aim is to identify web service logic that compromises security in some way. Only some brief guidelines are provided as each engagement will be different. There is no formal process or set of universally applicable tests.

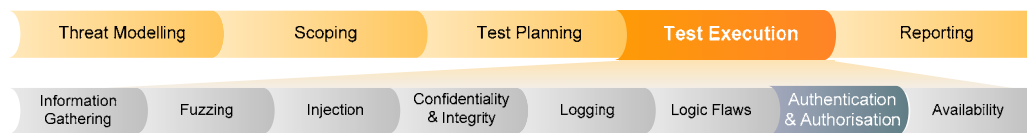
Detecting logic flaws in a black-box testing environment is difficult, time consuming and should not be relied upon. It is recommended that the tester be on the lookout for logic flaws during other forms of testing but not dedicate any significant portion of time specifically to discovering them.

The detection of logic flaws usually requires careful analysis rather than just pure input testing. Any odd or unexpected behaviour of the system should be analysed to determine the cause and impacts of the behaviour. Logic flaws are also difficult to detect in a black-box environment and access to source code and program logic will greatly aid the ability of the tester to identify and verify these issues.

Reference	TE-LF01
Test Case	Logic Flaws
Objective	Identify any logic flaws that could cause unintended application behaviour.
Description	It may be possible to abuse application logic to initiate actions which are not authorised or otherwise intended.
Threat	Tampering Repudiation Information Disclosure Denial of Service Elevation of Privilege
Impacted Asset	Web Service
Test Method	Identify and evaluate core business logic present in the web service. Attempt to execute available logic paths to perform actions in unintended ways. This

often includes manipulating the order in which actions are executed or using an unexpected method for performing actions.

AUTHENTICATION & AUTHORISATION



Prerequisites:

- Test Plan Document
- Threat Profile Document

Outcomes:

- A list of authentication and authorisation issues identified

Authentication and authorisation are critical elements of network exposed systems such as web services. Assessing these two complex functions is necessary to ascertain their compliance with application requirements and determine their ability to be manipulated or exploited.

Authentication and authorisation in web services is typically performed using a custom implementation or a standards-based implementation based on OASIS and XML standards. Custom implementations may utilise established cryptographic primitives or may contain proprietary algorithms that require a more rigorous assessment than standards-based implementations as also discussed in the Confidentiality & Integrity section. Testing an authentication and authorisation solution typically involves a detailed cryptographic analysis of the authentication protocol and access control system as well as an assessment of the implementation.

Authentication

Authentication is typically provided by means of a username and password. Depending on the requirements of the application, additional factors of authentication such as smartcards, tokens or biometrics may also be utilised to provide greater assurance of identity. Implementations range from home-grown proprietary solutions to highly structured and tightly integrated solutions made up of standardised products and protocols. Thus the approach taken to testing will depend on the solution under test. However, it will always be necessary to determine how user identity is established and to evaluate the authentication protocol used.

Security testing of the authentication component should be performed with the objective of finding areas in which the implementation does not meet specified requirements. Test cases should include forged or modified credentials, missing credentials and other inputs to induce error handling in the application. The specifics will vary depending on the technologies used but the aim is to generate fraudulent credentials that will either be accepted as genuine or cause the application to fail in some way that will allow access without authentication. The other aspect of testing the authentication component is to attack the authentication exchange. This typically consists of capturing a legitimate exchange and attempting to extract the credentials from it necessary for accessing the application. The captured exchange (or parts thereof) can also be replayed to the application in an attempt to impersonate the legitimate user. An attempt to hijack the session after a legitimate user has authenticated should be attempted in order to

test the session authentication mechanisms. It is also possible to attempt to extract authentication material hard coded into client software or deceive the server into exposing it. Brute-force and dictionary attacks against password-based systems should also be attempted.

Reference	TE-AA01
Test Case	Brute-force and Dictionary Attacks
Objective	Assess the susceptibility of the authentication exchange to brute-force or dictionary attacks.
Description	<p>These types of attacks are typically used against password authentication systems and rely on the ability to repeatedly test potential passwords against the authentication service.</p> <p>Brute-force attacks involve progressively testing all valid combinations of password characters in order to determine the real password for a given user account. This method will always succeed given enough time as all possible passwords will eventually be tested.</p> <p>Dictionary attacks are a more refined approach and involve testing a subset of the possible passwords, typically a dictionary of language words. Additional tests may include appending numbers to dictionary words, replacing letters with defined symbols and applying lowercase or uppercase modifications.</p>
Threat	Spoofing Elevation of Privileges
Impacted Asset	Credentials
Test Method	Identify the protocol elements in the authentication exchange. Determine whether mechanisms such as account lockouts and password salting are in place to increase the difficulty of these attacks. Attempt to initiate a brute-force or dictionary attack on the web service with appropriate wordlists.

Reference	TE-AA02
Test Case	Forged Credentials
Objective	Identify the system response to forged or modified credentials such as self-generated certificates or arbitrary tokens.
Description	Credentials should be issued by an authorised party and verified by the application when presented. Failure to appropriately verify the validity of presented credentials may allow an attacker to impersonate legitimate users or gain access to unauthorised functionality.
Threat	Spoofing Elevation of Privileges
Impacted Asset	Credentials
Test Method	Examine message formats to determine how credentials are presented to the web service and attempt to create similar but unauthorised credentials. This may include supplying self-generated certificates, forged tokens or random data as a credential to the web service and determining whether they are accepted as valid.

Reference	TE-AA03
Test Case	Missing Credentials
Objective	Identify the system response to missing credentials.
Description	A user that fails to present credentials should not be allowed access and the application should discard their request. The presence of credentials should not be assumed by the application processing logic and there should not be any form of default or backup account with application privileges.
Threat	Spoofing
Impacted Asset	Credentials
Test Method	Determine the element of the web service request containing user credentials and omit it from a valid request. Ensure that such requests are discarded and not processed under some generic user account.

Reference	TE-AA04
Test Case	Replay Attacks
Objective	Assess the susceptibility of the authentication exchange to replay attacks.
Description	A replay attack occurs when an attacker captures an authentication exchange and plays back the client portions of the exchange to the web service. If no mechanisms are present to detect this form of attack, it may be possible for the attacker to authenticate as the previously authenticated user.
Threat	Spoofing Elevation of Privileges
Impacted Asset	Credentials
Test Method	Identify the protocol elements in the authentication exchange. Determine whether mechanisms such as nonces, session tokens or timestamps are used within the authentication exchange to prevent re-use and guarantee 'freshness'. Capture a valid authentication exchange and attempt to use the information contained to impersonate the legitimate user.

Reference	TE-AA05
Test Case	Authentication Exchange Tampering
Objective	Assess the susceptibility of the authentication exchange to tampering by a malicious user.
Description	If elements of the authentication exchange can be modified by an attacker, they may be able manipulate the messages in order to subvert the authentication protocol. This may allow the tester to log in as a different user, gain elevated privileges or facilitate another form of attack.
Threat	Spoofing Tampering Elevation of Privileges

Impacted Asset	Credentials
Test Method	Identify the protocol elements in the authentication exchange. Determine whether mechanisms are in place to protect the integrity of the protocol elements within the authentication exchange. Attempt to compromise the authentication exchange by modifying unprotected elements.

Reference	TE-AA06
Test Case	Man-in-the-Middle Attacks
Objective	Determine the ability of the web service to resist man-in-the-middle attacks.
Description	<p>A man-in-the-middle attack occurs when a malicious attacker inserts themselves into the communications stream between two parties without their knowledge. This is possible if the two communicating parties are not mutually authenticated as this allows the attacker to impersonate each party to the other.</p> <p>Once this occurs, communications between the two parties will be completely compromised despite the use of encryption or integrity mechanisms because each party establishes a "secure" connection with the attacker whilst believing it to be with the other party. This provides the attacker with complete control over communications as they can read or modify data whilst passing it between the two parties.</p>
Threat	<p>Spoofing</p> <p>Tampering</p> <p>Information Disclosure</p>
Impacted Asset	<p>Credentials</p> <p>Data</p>
Test Method	Identify the protocol elements in the authentication exchange. Determine if authentication is bidirectional (i.e. mutual) as this prevents man-in-the-middle attacks by authenticating both parties to each other. If authentication is unidirectional, attempt to impersonate the server in order to initiate a man-in-the-middle attack and allow a legitimate user to connect to you and establish a "secure" session. This will allow the tester to read all traffic, modify requests and forward them to the real server for processing.

Reference	TE-AA07
Test Case	Factors of Authentication
Objective	Determine the factors of authentication used to identify users to the system.
Description	<p>It is widely accepted that authentication can take three forms:</p> <ul style="list-style-type: none"> ▪ Something you know (e.g. password) ▪ Something you have (e.g. token) ▪ Something you are (e.g. biometric) <p>For sensitive systems, it is often a requirement to authenticate users with at least two factors of authentication.</p>
Threat	Spoofing
Impacted Asset	Credentials
Test Method	Determine the factors of authentication used in the application and how they are bound to one another to provide assurance of identity. Test if one factor of authentication can be independently used with another. For example, supply Alice's digital certificate but log in as Bob.

Reference	TE-AA08
Test Case	Authentication Session Manipulation
Objective	Assess the ability to tamper with or subvert authentication session mechanisms.
Description	Authentication typically only occurs once per user session to avoid inconvenience. This is made difficult because SOAP is a stateless message-based communications protocol. Common methods for implementing this login process include the generation of a session token that is bound to the user or having the client cache the authentication credentials and automatically supply them with all subsequent messages.
Threat	Spoofing Tampering Elevation of Privileges
Impacted Asset	Credentials
Test Method	Identify if authentication is maintained between messages and the mechanisms used to implement this. Attempt to modify issued tokens or credential caches in order to manipulate the server's perception of user identity.

Implementation issues also exist but will vary depending on the technologies used. The products and technologies used should be assessed to ensure that they provide the features required for the application. It is important to verify that the solution satisfies the design and correctly implements the authentication protocol.

Reference	TE-AA09
Test Case	Storage of Authentication Credentials
Objective	Assess the strength of authentication credential stores.
Description	In order for a web service to identify a user, it must have some record of that user and their credentials. These user credentials must be stored securely to avoid credential theft by malicious users seeking to impersonate others in order to gain unauthorised access. Furthermore, client applications often cache or store user credentials for convenience and ease of use and these temporary copies must also be adequately protected.
Threat	Spoofing Elevation of Privileges
Impacted Asset	Credentials Stored data
Test Method	<p>The storage of authentication credentials on the server and client side must be assessed. Server side storage commonly takes the form of a directory service, database or configuration file. It is vital to test that passwords are not stored in the clear but instead are stored in salted and hashed form. Strong access control or other protections should be in place to safeguard the credential store. The tester should evaluate these protections by attempting to circumvent them and recover authentication credentials that can be fraudulently used.</p> <p>Client side storage of credentials should also be assessed and this may involve analysing the client application to determine where credentials are stored. Common locations are the Windows registry and configuration files. These credential stores are often protected with encryption and an effort should be made to determine the scheme used and whether any hard-coded keys are stored in the application.</p>

Reference	TE-AA10
Test Case	Confidentiality of Authentication Exchange
Objective	Determine the ability to eavesdrop on the authentication exchange.
Description	The authentication exchange should be kept private between the two parties and this is typically achieved by encrypting the communication. Failure to do this may disclose information that assists an attacker in credential theft.
Threat	Information Disclosure
Impacted Asset	Credentials
Test Method	The tester should capture the authentication exchange and attempt to derive any information that may be of use to an attacker. Determine if the exchange is encrypted or performed in the clear and attempt to recover usable credentials.

Reference	TE-AA11
Test Case	Certificate Verification
Objective	Identify if the web service sufficiently verifies digital certificates presented to it.
Description	The correct usage of digital certificates requires them to be verified with the certifying authority. Failure to adequately verify certificates renders the authentication they provide untrustworthy.
Threat	Spoofing
Impacted Asset	Credentials
Test Method	Determine if digital certificates are utilised and whether they are verified with the issuer during the authentication process. Attempt to supply a valid certificate from another issuer, self-generated certificate in place of an authentic certificate, or an expired (but valid) certificate.

Several standards have been ratified to aid the deployment of authentication systems in web services. These include XML Signature [W3C 8] which specifies rules for creating and representing digital signatures, WS-Security [OASIS 2] which specifies the syntax for embedding signatures within SOAP messages and WS-Federation [IBM 2] which defines mechanisms for sharing identity, authentication and authorisation information. However, these standards do not explicitly define protocols and only define the syntax for expressing and transmitting identity and authentication information in a web services context. Thus compliance with these standards does not preclude the authentication solution from having vulnerabilities and the design and implementation must be evaluated for security weaknesses as described above.

Finally, it is critical to have an enforceable policy regarding authentication credentials. The development of such policy is beyond the scope of this paper but should, at a minimum, cover the following:

- Password or passphrase strength requirements
- Storage of authentication credentials
- Disclosure of authentication credentials
- Responses to and awareness of social engineering techniques

Authorisation

Authorisation and access control occur subsequent to authentication and are maybe the commonly implemented using access control lists or roles. These implementations must be assessed to ensure that the correct access or role is assigned to an authenticated user and that their access or role cannot be tampered with to enable access to unauthorised information or functionality. There are several approaches to test this functionality but the design and mechanisms used to determine roles first need to be understood.

In a white-box testing environment, the design and mechanisms used should be readily available from documentation or discussions with stakeholders. However, in a black-box environment this will need to be determined through interactions with the web service. Two common methods for implementing authorisation or access controls in web services are the

use of security tokens and address filtering. SOAP request parameters can be identified by observing the messages sent between client and server and filtering may be detected by invoking the web service from multiple points in the network or from external sites.

The majority of conventional web services will utilise some form of security token within SOAP requests for authorisation and access control purposes. Standards such as Security Assertion Markup Language (SAML) [OASIS 3] and eXtensible Access Control Markup Language (XACML) [OASIS 4] have been developed to define a common syntax for expressing authorisation and access control information. Testing implementations that utilise and comply with these standards involves assessing the token generation and distribution processes and attempting to find weaknesses that may allow forgery or unauthorised modification of tokens.

Neither SAML nor XACML explicitly define protocols or implementations that provide authorisation or access control for web services so it is necessary to verify that tokens are generated from a suitably pseudorandom source. The tokens themselves should be analysed for predictability and opacity. In testing this component, it is useful to proxy the communications between client and server in order to try and discover fraudulent tokens that, when injected, are accepted by the application and provide a level of unauthorised access.

Additionally, it is important to verify that integrity and authenticity mechanisms exist to protect tokens during transit to prevent tampering. Tokens should be protected by some form of digital signature that is verified by the server before attempting to process the token. This allows the server to determine whether the token was tampered with and who it was issued by. To verify these protections during testing, a proxy should be used and tokens and signatures should be tampered with to check that modifications are detected. One can also attempt to substitute tokens and forged signatures to test the trust boundaries of the server.

The federation of web services also presents areas that need to be assessed. WS-Federation and WS-Trust [ACT 2] define methods for establishing trust and exchanging authentication and authorisation information between different domains. It is essential to evaluate the implementation of these standards to determine if the trust placed in third party web services, such as identity services, can be manipulated. Additionally, the policy behind trust decisions should be reviewed to ensure that it complies with specified requirements.

In the case of IP address filtering being employed for access control purposes, it must be established that additional mechanisms are in place as address filtering alone does not provide adequate protection. Addresses can typically be spoofed in order to bypass these ACLs or communications may be able to be proxied or tunnelled through a valid address. These methods should be tested in attempts to bypass this form of access control.

Temporary files created during the course of the application's uptime can leak valuable information to an attacker and it is important that these are protected. Temporary files should be created in restricted access directories and have permissions set to deny access from guests and regular users. These directories should not be Internet accessible and the files should be removed as soon as they are no longer required.

Reference	TE-AA12
Test Case	ACL and Role Consistency
Objective	Assess the consistency of ACLs and role definitions to business rules and policy.
Description	Implemented access control lists and user roles must be consistent with those defined by policy. Inconsistencies may provide an attack vector for malicious users to gain access to unauthorised functionality.
Threat	Elevation of Privileges
Impacted Asset	Web service functionality
Test Method	Attempt to enumerate access control lists and determine the consistency of the implemented ACLs and roles with those defined by policy. Identify any deviations and map the extent to which access to unauthorised data or functionality is granted. Attempt to use unauthorised functionality such as user administration facilities to gain further access to the system.

Reference	TE-AA13
Test Case	Token Forgery
Objective	Determine the extent to which tokens can be manipulated or forged.
Description	As SOAP is a stateless message-based protocol, some mechanism must be implemented to provide authorisation between SOAP requests or maintain session state. These mechanisms commonly use a token to identify a session or provide proof of authorisation and it is necessary to assess the ability to successfully forge or guess these tokens.
Threat	Tampering Elevation of Privileges
Impacted Asset	Tokens
Test Method	Identify the protective mechanisms surrounding security tokens such as integrity checks and signatures. Attempt to tamper with or forge tokens in order to manipulate the level of access granted. Additionally, attempt to manipulate integrity checks or signatures in order to pass server verification checks or attempt to bypass token verification altogether.

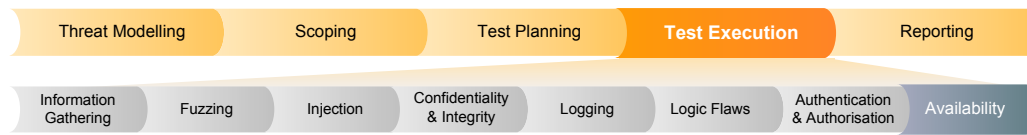
Reference	TE-AA14
Test Case	Hijacking Attacks
Objective	Assess the resilience to session hijacking attacks.
Description	As SOAP is a stateless message-based protocol, some mechanism must be implemented to provide authorisation between SOAP requests or maintain session state. These mechanisms commonly use a token to identify a session or provide proof of authorisation and it is necessary to assess the ability to steal and misuse these tokens.
Threat	Spoofing Tampering
Impacted Asset	Tokens
Test Method	Determine the token handling mechanism and attempt to subvert it. Identify where tokens are stored, how they are transmitted and how they are protected. Attempt to force a client to divulge a token in order to hijack its session and assume the identity of the authenticated user or steal their granted permissions.

Reference	TE-AA15
Test Case	Manipulation Across Trust Boundaries
Objective	Determine the ability to manipulate trust and authorisation between different trust domains.
Description	The federation of web services presents a new issue where trust and authorisation are concerned. Identity federation between web services is becoming more common and brings about the situation where one web service authenticates a user to another web service. SAML 2.0 supports the concept of federated identity and provides a standardised syntax for conveying security assertions such as authorisation information in token format. If trusts between web services are needlessly strong or assertions are not verified with the issuer, a risk is introduced where a malicious user may exploit a web service's trust in another to gain higher privileges than assigned.
Threat	Tampering Elevation of Privileges
Impacted Asset	Credentials
Test Method	Identify trust relationships between the target web service and others. Assess the ability to utilise a trusted or semi-trusted intermediary web service to provide authorisation or access control information to the target web service. Attempt to use weaknesses in the intermediary web service to gain elevated privileges that will carry over to the target web service. Try to tamper with assertion tokens that are carried over trust boundaries and verify that tampering is detected and that the tokens are validated with the issuer.

Reference	TE-AA16
Test Case	Attacks on Address Filtering
Objective	Identify susceptibility to address spoofing attacks.
Description	Address filtering is an unsophisticated method of authenticating or authorising users and is often susceptible to address spoofing. This involves forging addressing information to make requests appear to be coming from a different source. Address filtering may also be circumvented by proxying requests through a valid node such that the requests appear to be coming from a valid source address.
Threat	Spoofing
Impacted Asset	Credentials Network resources
Test Method	Determine if any form of address filtering is in place by testing from different points in the internal and external networks. MAC and IP address filtering can be spoofed in order to access the service from unauthorised nodes. Attempt to proxy or tunnel requests through a valid node for an unauthorised node.

Reference	TE-AA17
Test Case	Temporary Files
Objective	Assess the level of information disclosure possible from unauthorised access to temporary files.
Description	Temporary files created by the web service application that are accessible to users may reveal important information if not adequately protected with access controls or encryption.
Threat	Information Disclosure
Impacted Asset	Stored data
Test Method	Determine the presence of temporary files and attempt to access them via the web service or some other means. Identify the access controls around the temporary files and any other protections used to prevent unauthorised access to transient data.

AVAILABILITY



Prerequisites:

- Test Plan Document
- Threat Profile Document

Outcomes:

- A list of availability issues identified

In today's world of interconnectivity and e-commerce, businesses often require 99.9% (or higher – sometimes stretching to five or even six “nines”, i.e. 99.9999%) uptime of their applications. This requirement is exacerbated by the nature of web services which are being increasingly used for B2B integration between organisations. Service Level Agreements (SLAs) dictate strict availability levels with large the potential for monetary penalties for breaches. SLAs together with customer goodwill make capacity planning and availability testing critically important.

Assessing a web service in an availability context typically involves a code review to identify denial of service conditions and executing test cases in an attempt to induce them. Other activities such as network architecture assessments to identify single points of failure or network capacity evaluations may also be performed but are beyond the scope of this paper. The assessment of a web service's vulnerability to network denial of service attacks is also out of scope of this paper.

Test cases to consider in this stage of testing include malformed SOAP requests, well-formed SOAP requests that contain both legal and illegal values, modified parameters that may satisfy SOAP type constraints but may be unexpected by the application and SOAP requests containing recursive or deeply-nested elements. Load testing is also an activity that should be performed to gauge the maximum capacity of the web service and HTTP pipelining can be employed in order to increase the rate with which requests can be generated from a single node.

Reference	TE-AV01
Test Case	Parameter Tampering
Objective	Assess the resilience to parameter tampering attacks.
Description	This broad class of attacks refers to the modification of SOAP request parameters in transit between client and server [LIN 1]. This is typically achieved through the use of a proxy component that intercepts client generated requests and allows the attacker to insert values that may have been disallowed at the client level. These tests attempt to determine and exploit the trust placed in the client and the reliance upon the assumption that the end-user is using a legitimate client application to generate SOAP requests. Certain unexpected parameters may cause the application to fail and result in a denial of service condition.
Threat	Tampering Denial of Service
Impacted Asset	Web service functionality Web service availability Data
Test Method	Proxy communications between client and server, and systematically alter SOAP elements in requests to determine server responses. Attempt to insert values not permitted by the client application such as out-of-range or random values and assess the server responses returned.

Reference	TE-AV02
Test Case	Coercive Parsing
Objective	Determine the ability to disrupt services through malformed requests.
Description	Coercive parsing is the name given to the class of attacks that involve supplying illegal or malformed SOAP requests to the web service in order to cause undesirable behaviour [NEG 1].
Threat	Denial of Service
Impacted Asset	Web service availability
Test Method	Submit malformed requests such as SOAP messages with: <ul style="list-style-type: none"> ▪ Additional or missing angle brackets ▪ Additional or missing closing tags ▪ Missing required attributes ▪ Additional or missing quotes

Reference	TE-AV03
Test Case	Recursive SOAP
Objective	Assess the ability for the web service to handle recursive XML.
Description	This attack refers to the excessive nesting of elements within an XML SOAP request [LAY 1]. Although the request may be valid and well-formed, the level of nesting may cause the application or XML parser to fail and cause a denial of service condition. This is typically a result of inefficient DOM parsing and the inability to completely validate the XML prior to parsing.
Threat	Denial of Service
Impacted Asset	Web service availability
Test Method	<p>Deeply nested SOAP request such as:</p> <pre> <?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> <Order ... > <SubOrders ... > <Order ... > <SubOrders ... > <Order ... > <SubOrders ... > <Order ... > <SubOrders ... > <Order ... > ... </Order> </SubOrders> </Order> </SubOrders> </Order> </SubOrders> </Order> </SubOrders> </Order> </soap:Body> </soap:Envelope> </pre>

Reference	TE-AV04
Test Case	Overly Large SOAP
Objective	Assess the ability for the web service to handle large XML messages.
Description	Similar to the Recursive SOAP test case, this attack attempts to overwhelm the application or XML parser with a valid and well-formed SOAP request that contains an excessive number of elements or elements of unexpectedly large size [LIN 1]. This may cause the application to fail and cause a denial of service condition. This is typically a result of inefficient DOM parsing and the inability to completely validate the XML prior to parsing.
Threat	Denial of Service
Impacted Asset	Web service availability
Test Method	Excessively large SOAP request such as: <pre><?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> <Order ... >...</Order> <Order ... >...</Order> ... <Order ... >...</Order> </soap:Body> </soap:Envelope></pre>

Reference	TE-AV05
Test Case	Entity References
Objective	Determine the susceptibility of the web service to entity reference attacks.
Description	This class of attack refers to the manipulation of the ability for XML documents to reference other sources and include them for further processing or within the resulting output [NEG 1]. Although the SOAP specification prohibits DTD elements within a SOAP message, these attacks can be leveraged against XML documents contained within SOAP body fields where they are used. If the application can be coerced into referencing a source specified by the attacker, it may inadvertently process arbitrary documents and files. This may have several effects including the disclosure of file contents, initiation of network connections or a denial of service condition. It may also be possible to cause a denial of service if entity references are created with overly large reference strings. This can be obfuscated by constructing the reference in a recursive manner using a series of entity references.
Threat	Information Disclosure Denial of Service
Impacted Asset	Web service availability Stored data

	Server system information
Test Method	<p>Reference a source that contains a link to include file:///dev/random for processing.</p> <p>e.g.</p> <pre><!ENTITY random SYSTEM "file:///dev/random"></pre> <p>This approach may also be used to disclose the existence of files as a file-not-found exception will typically occur if the referenced file does not exist.</p> <p>File contents may also be disclosed if a reference to the file can be dereferenced and returned to the caller.</p> <p>e.g.</p> <pre><!ENTITY passwd SYSTEM "file:///etc/passwd"> <!ENTITY shadow SYSTEM "file:///etc/shadow"></pre> <p>Overly large reference built recursively.</p> <p>e.g.</p> <pre><!ENTITY junk "abcdefghijklmnopqrstuvwxy1234567890"> <!ENTITY junk2 "&junk; &junk;"> <!ENTITY junk3 "&junk2; &junk2;"> <!ENTITY junk4 "&junk3; &junk3;"> ... <!ENTITY junk1000 "&junk999; &junk999;"></pre>

Reference	TE-AV06
Test Case	Schema Poisoning
Objective	Assess the ability to manipulate XML validation schemas.
Description	Schema poisoning involves tampering with an XML schema used for validating SOAP requests in order to bypass or subvert the XML validation routines [LAY 1]. This may allow the manipulation of accepted data types, allowed elements, validation routines or the concealment of malicious data within requests that may cause unexpected behaviour in the application.
Threat	Tampering Denial of Service
Impacted Asset	Web service functionality Web service availability Data
Test Method	Attempt to access and tamper with schema documents to remove type checking, add/remove request elements or delete schemas altogether. If this can be achieved then generate SOAP requests that comply with the new schema that contains malicious elements and gauge server response.

Reference	TE-AV07
Test Case	Routing Detours
Objective	Identify if routing instructions can be manipulated to modify message routes.
Description	If the web service utilises WS-Routing to direct SOAP traffic, it may be possible to tamper with routing instructions contained in SOAP messages to alter the path it takes [NEG 1]. This may allow an attacker to view the contents of confidential messages or cause a denial of service by routing to non-existent nodes.
Threat	Information Disclosure Denial of Service
Impacted Asset	Web service availability Data
Test Method	<p>Routing to a non-existent node such as:</p> <pre><?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:Header> <wsroute:path xmlns:wsroute="http://schemas.xmlsoap.org/rp/"> <wsroute:action> http://www.corporation.com/transact </wsroute:action> <wsroute:to> soap://soap.corporation.com/transactions/initiate </wsroute:to> <wsroute:id> uuid:00000000-1234-1234-1234-123456789 </wsroute:id> <wsroute: fwd> <wsroute:via> soap://server.partner.com </wsroute:via> <wsroute:via> soap://non-existent.null.com </wsroute:via> </wsroute: fwd> <wsroute:rev> <wsroute:via> soap://non-existent.null.com </wsroute:via> </wsroute:rev> <wsroute:from></pre>

	mailto:user@corporation.com </wsroute:from> </wsroute:path> </soap:Header> <soap:Body> ... </soap:Body> </soap:Envelope>
--	---

Reference	TE-AV08
Test Case	Transform Attacks
Objective	Determine if it is possible to obfuscate malicious data through transformation rules.
Description	XML transforms may be utilised by the web service to convert between data formats or for internal preprocessing. If transforms are used, it may be possible for an attacker to generate an obfuscated request that only becomes malicious once transformed according to the application's transformation rules [LAY 1].
Threat	Tampering Denial of Service
Impacted Asset	Web service availability Data
Test Method	Identify the transforms applied to requests and analyse transformations to determine if malicious inputs can be hidden. This test case will be extremely implementation specific and may be difficult in a black box testing environment.

Reference	TE-AV09
Test Case	Authentication Flooding
Objective	Assess the resilience of the web service to authentication flooding.
Description	The authentication component is a common target to attack in order to induce denial of service conditions because of its criticality and centralised nature. Furthermore, initial login functions are unauthenticated but typically require the application server to initiate connections to backend databases or authentication servers and these connection pools can be exhausted.
Threat	Denial of Service
Impacted Asset	Web service availability
Test Method	A denial of service attack on an authentication component commonly occurs in one of two ways: flooding the component with authentication requests in order to prevent legitimate requests from being processed or attacking policy decisions such as account lockouts. Both methods should be employed by the tester to assess the effort and resources required to induce these conditions.

REPORTING

Threat Modelling

Scoping

Test Planning

Test Execution

Reporting

Prerequisites:

- Test Plan Document
- Threat Profile Document
- Test Case Execution Results

Outcomes:

- Executive Testing Report
- Technical Testing Report

There is no purpose in performing extensive testing if the results cannot be communicated to the people with the power to facilitate the remediation of identified vulnerabilities. This section outlines some considerations for web services security reporting however a detailed discussion is considered out of scope.

Prior to the production of a final report, some of the reporting effort will already have been completed. Test cases and test plans are an important part of the documentation of the testing process. They can be used for later regression testing, verification that identified issues have been resolved and auditing of testing procedures.

It is always important to understand the target audience and report accordingly. The two key target groups that are generally most relevant and hence should be given the most attention are management and technical personnel. The groups may be addressed in a single report or separate reports.

Management require high-level information that will allow them to make the best possible decisions on how to proceed in handling identified flaws in web services. However, the most valuable output for management is the risk that a vulnerability presents and the associated recommendations required to minimise or mitigate that risk. As there are many competing approaches to "rating" a given risk, it is recommended a recognised standard such as AS/NZS 4360 be extended to produce a suitable system for the organisation receiving the reports.

Technical staff generally require a lower-level perspective on the identified problems, although in some cases providing in depth information may not be practical. When it is, it may be provided in the form of a proof of concept, step-by-step recommendations, clear avoidance strategies or any other pertinent fashion. Vulnerabilities should be associated with their relevant web method or other logical component. Furthermore, it may be helpful to reference the test cases that were used to discover a particular flaw so they may be prioritised for regression testing and fine tuning the security testing methodology as a whole.

Web services security reporting is not only about stepping through each flaw, one by one, but identifying the root cause of a group of related issues. That is, separating implementation flaws from design flaws. It is crucial that the two categories are reported in their own right to provide architects a clear risk profile of the web service.

REFERENCES

[ACT 1] "Web Services Secure Conversation Language", Actional Corporation et al, February 2005

[ACT 2] "Web Services Trust Language", Actional Corporation et al, February 2005

[ANL 1] "Advanced SQL Injection In SQL Server Applications", Chris Anley, NGS Software, http://www.nextgenss.com/papers/advanced_sql_injection.pdf

[AS/NZS 4360] "AS/NZS 4360:2004 Risk Management", Risk Management, Standards Australia, 2004

[CER 1] "Manipulating Microsoft SQL Server Using SQL Injection", Cesar Cerrudo, Application Security, Inc., http://www.appsecinc.com/presentations/Manipulating_SQL_Server_Using_SQL_Injection.pdf

[FAU 1] "LDAP Injection – Are your web applications vulnerable", Sacha Faust, SPI Dynamics , <http://www.spidynamics.com/whitepapers/LDAPinjection.pdf>

[FRE 1] "The SSL Protocol Version 3.0", A. Freier, P. Karlton and P. Kocher, Transport Layer Security Working Group, November 1996, <http://wp.netscape.com/eng/ssl3/draft302.txt>

[HOL 1] "Web Service Security – Vulnerabilities and Threats Within the Context of WS-Security", Jesper Holgersson and Eva Soderstrom, University of Skovde, 2005.

[HOW 1] "RFC 2255: The LDAP URL Format", T. Howes, M. Smith, Netscape Communications Corp, December 1997

[IBM 1] "Web Services Security Policy Language", IBM et al, 18 December 2002

[IBM 2] "Web Services Federation Language", IBM et al, Verisign, 8 July 2003

[KLE 1] "Blind XPath Injection", Amit Klein, Sanctum Inc., 2004, http://www.packetstormsecurity.org/papers/bypass/Blind_XPath_Injection_20040518.pdf

[LAY 1] "XML Threats and Web Services Vulnerabilities", Layer 7 Technologies Inc., 2004, <http://www.layer7tech.com/library/page.html?id=39#wp>

[LIN 1] "Attacking and Defending Web Services", Pete Lindstrom, Spire Security, January 2004, http://forumsystems.com/papers/Attacking_and_Defending_WS.pdf

[MEI 1] "Threat Modeling Web Applications", J.D. Meier et al, Microsoft Corporation, May 2005, <http://msdn.microsoft.com/security/securecode/threatmodeling/default.aspx?pull=/library/en-us/dnpag2/html/tmwa.asp>

[MEI 2] "Improving Web Application Security: Threats and Countermeasures Roadmap", J.D. Meier et al, Microsoft Press, September 2003

[NEG 1] "Anatomy of a Web Services Attack", Walid Negm, Forum Systems, Inc., 1 March 2004

[OASIS 1] "UDDI Version 3.0.2", OASIS, 19 October 2004, http://uddi.org/pubs/uddi_v3.htm

[OASIS 2] "Web Services Security: SOAP Message Security 1.0", OASIS, March 2004, <http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0>

[OASIS 3] "Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0", OASIS, 15 March 2005, <http://docs.oasis-open.org/security/saml/v2.0/>

[OASIS 4] "eXtensible Access Control Markup Language", OASIS, 1 February 2005, http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf

[OWASP 1] "The OWASP Testing Project", The OWASP Foundation, December 2004

[OWASP 2] "OWASP Web Application Penetration Checklist", The OWASP Foundation, July 2004

[RFC 1945] "Hypertext Transfer Protocol – HTTP/1.0", Berners-Lee et al, Network Working Group, May 1996, <http://www.w3.org/Protocols/rfc1945/rfc1945>

[RFC 2246] "The TLS Protocol Version 1.0", T. Dierks and C. Allen, Network Working Group, January 1999, <http://www.ietf.org/rfc/rfc2246.txt>

[RFC 2251] "Lightweight Directory Access Protocol (v3)", Wahl et al, Network Working Group, December 1997, <http://www.ietf.org/rfc/rfc2251.txt>

[RFC 2616] "Hypertext Transfer Protocol – HTTP/1.1", Fielding et al, Network Working Group, June 1999, <http://www.w3.org/Protocols/rfc2616/rfc2616.txt>

[RFC 2828] "RFC2828 - Internet Security Glossary", R. Shire, GTE / BBN Technologies, May 2000, <http://www.ietf.org/rfc/rfc2828.txt>

[SIFT 1] "SIFT Note 2005-05", SIFT, 8 June 2005, <http://www.sift.com.au/17/137/sift-note-200505.htm>

[SIFT 2] "SIFT Note 2005-06", SIFT, 18 July 2005, <http://www.sift.com.au/16/138/sift-note-200506.htm>

[SIFT 3] "SIFT Web Method Search Tool", SIFT, 5 September 2006, <http://www.sift.com.au/73/171/sift-web-method-search-tool.htm>

[SQL 1] "SQL Injection FAQ", SQL Security <http://www.sqlsecurity.com/DesktopDefault.aspx?tabid=23>

[STA 1] "Attacking Web Services", Alex Stamos and Scott Stender, Information Security Partners, 2005.

[SUN 1] "Securing Web Services – Concepts, Standards, and Requirements", Sun Microsystems, October 2003

[SWI 1] "Threat Modeling", Frank Swiderski and Window Snyder, Microsoft Press, July 2004

[W3C 1] "Web Services Description Language (WSDL) 1.1", W3C, 15 March 2001, <http://www.w3.org/TR/wsdl>

[W3C 2] "SOAP Version 1.2 Part 0: Primer", W3C, 24 June 2003

[W3C 3] "SOAP Version 1.2 Part 1: Messaging Framework", W3C, 24 June 2003

[W3C 4] "SOAP Version 1.2 Part 2: Adjuncts", W3C, 24 June 2003

[W3C 5] "XML Path Language (XPath) Version 1.0", W3C, 16 November 1999, <http://www.w3.org/TR/xpath>

[W3C 6] "Extensible Markup Language (XML) 1.0 (Third Edition)", W3C, February 2004, <http://www.w3.org/TR/2004/REC-xml-20040204/>

[W3C 7] "XML Encryption Syntax and Processing", W3C, December 2002, <http://www.w3.org/TR/xmlenc-core/>

[W3C 8] "XML-Signature Syntax and Processing", W3C, February 2002, <http://www.w3.org/TR/xmlsig-core/>

[W3C 9] "Canonical XML Version 1.0", W3C, March 2001, <http://www.w3.org/TR/xml-c14n>

GLOSSARY

Fuzzer

A security testing tool that generates semi-structured pseudo random input into an application in order to solicit unexpected behaviour.

HTTP

The Hypertext Transfer Protocol (HTTP) is a TCP-based, application-layer, client-server, Internet protocol [R2616] used to carry data requests and responses in the World Wide Web.

LDAP

Lightweight Data Access Protocol (LDAP) is a protocol used for querying and modifying information in an LDAP store, typically directory services.

OASIS

OASIS is a not-for-profit global consortium that drives the development, convergence and adoption of e-business standards.

OWASP

The Open Web Application Security Project (OWASP) is a non-profit organisation which provides standardised tools and information relating to web application security.

Security Mechanism

A process (or a device incorporating such a process) that can be used in a system to implement a security service that is provided by or within the system.

Security Service

A processing or communication service that is provided by a system to give a specific kind of protection to system resources.

SSL

Secure Sockets Layer (SSL) is an Internet protocol that uses connection-oriented end-to-end encryption to provide data confidentiality service and data integrity service for traffic between a client and a server, and that can optionally provide peer entity authentication between the client and the server.

SOAP

Web services communication is based upon the Simple Object Access Protocol (SOAP). SOAP is an XML-based information packaging definition which can be used for exchanging structured and typed information between peers in a distributed environment.

TLS

Transport Layer Security (TLS) is a secure communications protocol based-on and very similar to SSL Version 3.

UDDI

Universal Description, Discovery and Integration (UDDI) is an XML based registry for used for listing web services.

W3C

The World Wide Web Consortium (W3C) develops interoperable technologies (specifications, guidelines, software, and tools) to lead the Web to its full potential. W3C is a forum for information, commerce, communication, and collective understanding.

WSDL

The Web Services Description Language (WSDL) is an XML format for describing web services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information.

XML

Extensible Markup Language (XML) is a simple, very flexible text format derived from SGML. It is most often used for creating special-purpose markup languages and storing data in a standardised format.

APPENDIX A: THREAT PROFILE TEMPLATE

Application Overview

- <What does the web service do?>*
- <Major components and their roles>*
- <Application architecture>*
- <Platforms and technologies used>*

System Decomposition

- <Trust levels and trust boundaries>*
- <Assets>*
- <Entry points>*
- <Data flows>*
- <Security mechanisms>*
- <System diagrams>*

Threats

- <Threat A>*
 - <Entry points used>*
 - <Data flows affected>*
 - <Trust boundaries crossed>*
 - <Assets impacted>*
- <Threat B>*
 - <Entry points used>*
 - <Data flows affected>*
 - <Trust boundaries crossed>*
 - <Assets impacted>*

APPENDIX B: SCOPE DEFINITION TEMPLATE

Executive summary

<High level summary of entire document>

Critical success factors

<What needs to occur for testing be considered successful?>

Assumptions

<Any assumptions that may impact the project>

In scope

Requirement	Priority
<i><Security requirement A></i>	
<i><Web service A></i>	<i><high/moderate/low></i>
<i><Web method 1></i>	
<i><Web method 2></i>	
<i><Web method 3></i>	
<i><Web service B></i>	<i><high/moderate/low></i>
<i><Web method 2></i>	
<i><Web method 4></i>	
<i><Security requirement B></i>	<i><high/moderate/low></i>
<i><Web service B></i>	
<i><Web method 1></i>	
<i><Security requirement C></i>	<i><high/moderate/low></i>
<i><Web service A></i>	
<i><Web method 3></i>	
<i><Web method 4></i>	

Out of scope

<Security requirement D>
<All web services>
<Security requirement B>
<Web method 1>

Schedule

<Estimated high level time line for major testing components>

Deliverables

<Test strategy>
<Test plan>
<Final report>
<Other deliverables>

APPENDIX C: TEST STRATEGY TEMPLATE

Executive summary

<High level summary of entire document>

Objectives

<What does this testing project aim to achieve?>

<High level testing objectives>

Deliverables

<Interim report(s)>

<Final report>

<Other deliverables>

Schedule & Milestones

<Estimated high level timeline for major testing phases>

<Delivery dates of project artefacts>

Resource Requirements

<WSDL documents>

<Source code>

<System documentation>

<Valid authentication credentials>

<System or network access>

Roles & Responsibilities

<Owners of components under test>

<Roles of people involved>

<Responsibilities of people involved>

Assumptions

<Any assumptions that may impact the project>

Testing Environment

<General understanding of the testing environment>

<Architecture overview>

Testing Approaches

<High level testing approaches>

Management Controls

<Issue notification and escalation>

<Issue tracking>

<Change control>

APPENDIX D: WEB SERVICES TESTING CHECKLIST

REF	TEST CASE	DESCRIPTION/OBJECTIVE	TESTED
Deliverables			
-	Threat Profile Document	The document produced by the threat modelling phase detailing the risks associated with the web service.	<input type="checkbox"/>
-	Scope Definition Document	The document produced by the scoping phase outlining the components and services to be assessed.	<input type="checkbox"/>
-	Test Strategy Document	The document produced by the planning phase that describes the overall testing methodology and approach.	<input type="checkbox"/>
-	Detailed Test Plan Document	The document produced by the planning phase that describes detailed testing information including test plans, sample inputs and expected behaviours.	<input type="checkbox"/>
-	Executive Testing Report	The document produced by the final reporting phase that provides a high-level overview of findings, risk analysis and recommendations for resolving issues or mitigating risk.	<input type="checkbox"/>
-	Technical Testing Report	The document produced by the final reporting phase that details the findings of testing at a technical level and recommendations for resolving or mitigating identified issues.	<input type="checkbox"/>
Information Gathering			
TE-IG01	WSDL Retrieval	Identify web method call mechanics.	<input type="checkbox"/>
TE-IG02	Error Message Information Leakage	Assess the extent of information provided by error messages.	<input type="checkbox"/>
TE-IG03	Web Method Enumeration	Identify methods not published in the WSDL.	<input type="checkbox"/>
Fuzzing			
TE-FZ01	Numerical Values	Identify mishandling of numerical fields.	<input type="checkbox"/>
TE-FZ02	Base64 Encoded Values	Identify flaws in Base64 data handling.	<input type="checkbox"/>

REF	TEST CASE	DESCRIPTION/OBJECTIVE	TESTED
TE-FZ03	Character Strings	Identify mishandling of character strings.	<input type="checkbox"/>
TE-FZ04	General Values	Test the handling of values that have an unspecified format.	<input type="checkbox"/>
TE-FZ05	Sub-system Parameters	Identify possible injection flaws in sub-systems.	<input type="checkbox"/>
TE-FZ06	Output Values	Assess the security of the data output mechanism.	<input type="checkbox"/>
TE-FZ07	Addressing Parameters	Assess the possibility of accessing resources outside those intended.	<input type="checkbox"/>
TE-FZ08	Tokens	Identify simple session token weaknesses.	<input type="checkbox"/>
TE-FZ09	Format String Parameters	Ascertain the web service's susceptibility to format string attacks.	<input type="checkbox"/>
TE-FZ10	Logging Values	Assess the logging mechanism's handling of simple attacks.	<input type="checkbox"/>
TE-FZ11	File Names	Identify directory traversal vulnerabilities.	<input type="checkbox"/>
Injection			
TE-IN01	SQL Injection	Identify SQL injection vulnerabilities.	<input type="checkbox"/>
TE-IN02	Command Injection	Discover any command injection vulnerabilities that may exist.	<input type="checkbox"/>
TE-IN03	LDAP Injection	Assess the web service's susceptibility to LDAP injection.	<input type="checkbox"/>
TE-IN04	XPath Injection	Identify XPath injection vulnerabilities.	<input type="checkbox"/>
TE-IN05	Code Injection	Identify code injection vulnerabilities.	<input type="checkbox"/>
TE-IN06	XML Special Characters	Identify weaknesses in the handling of special characters within inputs to XML documents.	<input type="checkbox"/>
TE-IN07	XML CDATA Sections	Determine vulnerabilities resulting from improper handling of CDATA sections.	<input type="checkbox"/>
Confidentiality & Integrity			
TE-CI01	Cipher Choice	Identify the encryption cipher used in the application.	<input type="checkbox"/>
TE-CI02	Encryption Coverage	Determine the items within web service communications that are encrypted.	<input type="checkbox"/>

REF	TEST CASE	DESCRIPTION/OBJECTIVE	TESTED
TE-CI03	Replay Attacks	Determine the susceptibility of the communication protocol to attacks on the 'freshness' of messages.	<input type="checkbox"/>
TE-CI04	Integrity Check Coverage	Determine the items within web service communications that are protected by message integrity checks.	<input type="checkbox"/>
TE-CI05	Invalid XML	Assess the response of the web service to malformed XML inputs.	<input type="checkbox"/>
TE-CI06	XML Canonicalisation	Determine the effectiveness of XML canonicalisation in the web service.	<input type="checkbox"/>
TE-CI07	Unsupported Algorithms	Assess the response of the web service to requests for unsupported algorithms within WS-Security, WS-SecurityPolicy and other security standards.	<input type="checkbox"/>
TE-CI08	Failed Policy Requirements	Assess the web service response to SOAP messages that do not meet security policy requirements.	<input type="checkbox"/>
Logging Issues			
TE-LG01	Separator Injection	Assess the ability of the web service to log messages that contain special separator characters.	<input type="checkbox"/>
TE-LG02	White Space Injection	Determine the logging mechanism's susceptibility to white space injection.	<input type="checkbox"/>
TE-LG03	XML Injection	Assess the possibility of inserting XML elements and/or attributes into an XML based log.	<input type="checkbox"/>
TE-LG04	HTML Injection	Assess the possibility of inserting HTML tags into a HTML based log.	<input type="checkbox"/>
TE-LG05	New Line Injection	Determine whether the logging mechanism is vulnerable to arbitrary entry creation via carriage return and line feed injection.	<input type="checkbox"/>
TE-LG06	Size Overflow	Assess the handling of log data after reaching the upper log size limit.	<input type="checkbox"/>
TE-LG07	Information Disclosure	Determine the possibility of retrieving or otherwise revealing information contained within log entries.	<input type="checkbox"/>
TE-LG08	Alternate Data Streams	Assess the ability to manipulate alternate data streams in logging facilities.	<input type="checkbox"/>
TE-LG09	Not Logged Actions	Verify the level of logging in place is sufficient.	<input type="checkbox"/>
TE-LG10	Logic Flaws	Identify any logic flaws that could cause some actions	<input type="checkbox"/>

REF	TEST CASE	DESCRIPTION/OBJECTIVE	TESTED
		to be logged incompletely or not at all.	
Logic Flaws			
TE-LF01	Logic Flaws	Identify any logic flaws that could cause unintended application behaviour.	<input type="checkbox"/>
Authentication & Authorisation			
TE-AA01	Brute-force and Dictionary Attacks	Assess the susceptibility of the authentication exchange to brute-force or dictionary attacks.	<input type="checkbox"/>
TE-AA02	Forged Credentials	Identify the system response to forged or modified credentials such as self-generated certificates or arbitrary tokens.	<input type="checkbox"/>
TE-AA03	Missing Credentials	Identify the system response to missing credentials.	<input type="checkbox"/>
TE-AA04	Replay Attacks	Assess the susceptibility of the authentication exchange to replay attacks.	<input type="checkbox"/>
TE-AA05	Authentication Exchange Tampering	Assess the susceptibility of the authentication exchange to tampering by a malicious user.	<input type="checkbox"/>
TE-AA06	Man-in-the-Middle Attacks	Determine the ability of the web service to resist man-in-the-middle attacks.	<input type="checkbox"/>
TE-AA07	Factors of Authentication	Determine the factors of authentication used to identify users to the system.	<input type="checkbox"/>
TE-AA08	Authentication Session Manipulation	Assess the ability to tamper with or subvert authentication session mechanisms.	<input type="checkbox"/>
TE-AA09	Storage of Authentication Credentials	Assess the strength of authentication credential stores.	<input type="checkbox"/>
TE-AA10	Confidentiality of Authentication Exchange	Determine the ability to eavesdrop on the authentication exchange.	<input type="checkbox"/>
TE-AA11	Certificate Verification	Identify if the web service sufficiently verifies digital certificates presented to it.	<input type="checkbox"/>
TE-AA12	ACL and Role Consistency	Assess the consistency of ACLs and role definitions to business rules and policy.	<input type="checkbox"/>
TE-AA13	Token Forgery	Determine the extent that tokens can be manipulated or forged.	<input type="checkbox"/>
TE-AA14	Hijacking Attacks	Assess the resilience to session hijacking attacks.	<input type="checkbox"/>

REF	TEST CASE	DESCRIPTION/OBJECTIVE	TESTED
TE-AA15	Manipulation Across Trust Boundaries	Determine the ability to manipulate trust and authorisation between different trust domains.	<input type="checkbox"/>
TE-AA16	Attacks on Address Filtering	Identify susceptibility to address spoofing attacks.	<input type="checkbox"/>
TE-AA17	Temporary Files	Assess the level of information disclosure from temporary files.	<input type="checkbox"/>
Availability			
TE-AV01	Parameter Tampering	Assess the resilience to parameter tampering attacks.	<input type="checkbox"/>
TE-AV02	Coercive Parsing	Determine the ability to disrupt services through malformed requests.	<input type="checkbox"/>
TE-AV03	Recursive SOAP	Assess the ability for the web service to handle recursive XML.	<input type="checkbox"/>
TE-AV04	Overly Large SOAP	Assess the ability for the web service to handle large XML messages.	<input type="checkbox"/>
TE-AV05	Entity References	Determine the susceptibility of the web service to entity reference attacks.	<input type="checkbox"/>
TE-AV06	Schema Poisoning	Assess the ability to manipulate XML validation schemas.	<input type="checkbox"/>
TE-AV07	Routing Detours	Identify if routing instructions can be manipulated to modify message routes.	<input type="checkbox"/>
TE-AV08	Transform Attacks	Determine if it is possible to obfuscate malicious data through transformation rules.	<input type="checkbox"/>
TE-AV09	Authentication Flooding	Assess the resilience of the web service to authentication flooding.	<input type="checkbox"/>

APPENDIX E: TEST CASE THREAT MAPPINGS

	Spoofing	Tampering	Reputation	Information Disclosure	Denial of Service	Elevation of Privileges
TE-IG01				•		
TE-IG02				•		
TE-IG03				•		•
TE-FZ01		•		•	•	•
TE-FZ02		•		•	•	
TE-FZ03		•		•	•	
TE-FZ04		•		•	•	
TE-FZ05	•	•		•	•	•
TE-FZ06	•			•		•
TE-FZ07		•		•		
TE-FZ08	•		•			•
TE-FZ09		•		•	•	•
TE-FZ10		•	•			
TE-FZ11		•		•		
TE-IN01	•	•		•	•	•
TE-IN02		•		•	•	•
TE-IN03	•					•
TE-IN04				•		
TE-IN05		•		•		•
TE-IN06		•			•	
TE-IN07		•			•	
TE-CI01				•		
TE-CI02				•		
TE-CI03	•		•			
TE-CI04		•				
TE-CI05		•		•	•	
TE-CI06		•				
TE-CI07		•		•		
TE-CI08		•		•		

TE-LG01		•	•			
TE-LG02		•	•			
TE-LG03		•	•			
TE-LG04		•	•			
TE-LG05		•	•			
TE-LG06		•	•			
TE-LG07				•		
TE-LG08		•	•			
TE-LG09			•			
TE-LG10			•			
TE-LF01		•	•	•	•	•
TE-AA01	•					•
TE-AA02	•					•
TE-AA03	•					
TE-AA04	•					•
TE-AA05	•	•				•
TE-AA06	•	•		•		
TE-AA07	•					
TE-AA08	•	•				•
TE-AA09	•					•
TE-AA10				•		
TE-AA11	•					
TE-AA12						•
TE-AA13		•				•
TE-AA14	•	•				
TE-AA15		•				•
TE-AA16	•					
TE-AA17				•		
TE-AV01		•			•	
TE-AV02					•	
TE-AV03					•	
TE-AV04					•	
TE-AV05				•	•	
TE-AV06		•			•	
TE-AV07				•	•	
TE-AV08		•			•	

TE-AV09



ABOUT SIFT

Our Profile

SIFT is one of Australia's premier information security consulting, intelligence and training firms. SIFT was founded in 2000 to support businesses in the important area of information security, through providing a focused and detailed understanding of policy, procedural, technical, and regulatory security risks and controls.

Uniquely focused on providing information security intelligence, consulting, and training within the context of industry and country specific regulatory requirements, SIFT is committed to providing concrete, specific and measured steps across the disciplines of information security, privacy and governance.

SIFT have built long-term relationships with major clients and information security stakeholders in both the public and private sectors, providing exceptional customer focus throughout our business units. Through our security intelligence and industry & regulatory connections, we are uniquely positioned to advise on security within the Australian context.

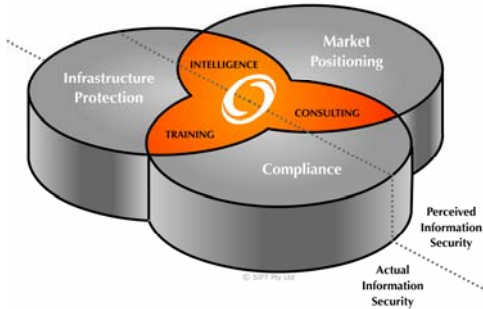
SIFT Pty Ltd

Head Office - Sydney
Level 6, 62 Pitt Street
Sydney NSW 2000

T: +61 2 9236 7276
F: +61 2 9251 6393

Melbourne
Level 40, 140 William Street
Melbourne VIC 3000

P: +61 3 9607 8274
F: +61 3 9607 8282



Our Services

Leveraging our unique perspective of information security issues in the Australian context, SIFT offers its clients a range of services:

Consulting

- Application Penetration Testing & Code Review
- Information Security Governance, Compliance & Reporting
- Board & Executive Level Support
- Security Reviews, Audits and Benchmarking
- Server & Network Level Penetration Testing
- Risk Assessment

Intelligence

- Policy & Procedure Development & Review
- Information Availability & Aggregation Reviews
- Product & Vendor Reviews/Recommendations
- Custom Research Reports

Training

- Foundations of Secure Web-Application Design
- Foundations of Information Security
- Introduction to Encryption & PKI
- Industry Based Training & Custom Training Programs



SIFT is a proud supporter of the work of the Inspire Foundation & Reach Out!