THREAT MODELLING FOR WEB SERVICES BASED WEB APPLICATIONS

Lieven Desmet, Bart Jacobs, Frank Piessens, and Wouter Joosen
DistriNet Research Group, Katholieke Universiteit Leuven, Celestijnenlaan 200A, 3001 Leuven, Belgium

Abstract: Threat analysis of a web application can lead to a wide variety of identified threats. Some of these threats will be very specific to the application; others will be more related to the underlying infrastructural software, such as the web or application servers, the database, the directory server and so forth. This paper analyzes the threats that can be related to the use of web services technology in a web application. It is part of a series of papers, written by different academic teams, that each focus on one particular technological building block for web applications.

Key words: Threat analysis, web services, web applications

1. INTRODUCTION

Analyzing and modelling the potential threats that an application faces is an important step in the process of designing a secure application. Some of these threats are by nature very specific to the application, and one can only give quite general guidelines on how to identify such threats. But other threats are directly or indirectly related to the underlying platforms, technologies or programming languages. Hence, it makes sense to identify and document these technology-specific threats, and to provide guidelines to software vendors on how to mitigate the associated risks.

This paper reports on the results of such an analysis for the use of web services technology in web applications. It is part of a series of papers [1,2,3,4,5], written by different academic teams, that each focus on one particular technological building block for web applications. Each of these

papers (including this one) starts from the generic architecture for web applications presented in [1].

2. WEB SERVICES

Web services are a more and more common building block in modern web applications. This section gives a short introduction to web services, and describes how they can be used in web applications.

2.1 Web services

A web service is essentially an XML-messaging based interface to some computing resource. The web services protocol stack consists of:

- Some transport layer protocol, typically HTTP.
- An XML-based messaging layer protocol, typically SOAP [9]
- A service description layer protocol, typically WSDL [10]
- A service discovery layer protocol, typically UDDI [11]

In this document, the assumed web services communication model is SOAP over HTTP. Basic SOAP interactions are asynchronous and unidirectional, but can be combined to implement request/response processes, or even more sophisticated interactions.

SOAP messages are XML based messages for exchanging structured and typed information. SOAP can be used to implement RPC, but the focus shifts to document based information flow in recent web service development.

Next to the originating and receiving node of a web service, intermediate nodes can be defined, as shown in Figure 1. Those intermediate nodes can process the SOAP message, and add extra information to the message (such as a signature on a part of the message).

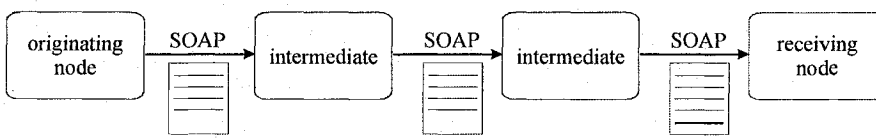

Figure 1. Process flow of a SOAP message.

This document also makes the assumption that WSDL is used to specify the public interface to a web service. A WSDL-based description of a web service can include information on available functions, typing information and address information. WSDL is usually generated by tools, not by hand.

The use of dynamic discovery of web services in web applications is not yet widely used. Hence this document does not consider the service discovery layer.

2.2 Web services in web applications

A generic architecture for web applications is presented in [1]. Within this architecture for web applications, the technology of web services can be used for a variety of purposes. Some examples include:

1. *Wrapping legacy applications:* Incorporating legacy application functionality within a web application is often done by giving the legacy application a web service façade, which can be used by the application server.
2. *Better web server – application server separation:* If the web server communicates with the application server by SOAP/HTTP instead of RPC, the firewall between the DMZ (containing the web server) and the middle tier only needs to open port 80.
3. *Rich Clients:* The browser can download client-side application components (such as Java Applets or .NET assemblies) from the web server. These components can then interact with the web server using web services.
4. *Integration of building block services:* Reusable application services such as authentication or storage can be made available as web services and be used in a variety of web applications.
5. *Multistage processing:* Web services support an asynchronous messaging model. A single request can traverse multiple intermediaries before reaching its final destination. For example, an authentication server as intermediary can authenticate the SOAP message before its arrival at the application server.
6. *Virtual organizations:* Web services can be used for business-to-business integration, creating useful federations of autonomous entities.

Since this paper intends to provide guidelines for Independent Software Vendors building web applications, we assume that the last scenario will be less common. Instead we focus on the most important threats in the other scenarios in the remaining of this paper. These scenarios do not use some of the more advanced features of web services, such as dynamic discovery of services and UDDI. Hence, our threat modelling does not consider these features either. This assumption seems to be in line with the Microsoft Threats and Countermeasures guide [8].

3. OVERVIEW OF ASSETS AND THREAT CATEGORIZATION

This section starts with an overview of the important assets within a web services based web application. Next, a generic classification of threats associated with web services is presented. The section ends with an overview of the attack entry points of web services within a web application.

3.1 Assets

The assets to be protected are subdivided into:

- *Application specific assets*: The data and procedures in the server systems are the main assets, possibly spread over all three tiers. Since we make an abstraction of the application, these cannot be detailed further. For a specific application, further analysis is necessary.
- *Web service specific technology artifacts*. These include elements such as WSDL files, assemblies implementing client and server calls, SOAP messages and so forth. The threats to these assets are the web services technology specific threats. Threats to these assets usually lead indirectly to threats to application specific assets (e.g. leaking of an assembly might give an attacker the necessary information on how to attack a back-end system, a SOAP message will usually include application specific information, tampering with a WSDL file may enable service spoofing, and so forth.)
- *Private information* on the client machine
- *Availability* of the various machines, connections and services in the architectural picture.

3.2 Overview of possible threats

We follow the STRIDE threat categorization [13] for systematically enumerating the threats. In this section, we discuss in a generic way the threats present in a scenario with a single web service consumer and a single web service provider.

1. *Spoofing*: Whenever the communication line between the web service consumer and provider crosses a trust boundary, there is a threat of spoofing. Both the provider and the consumer can be spoofed.
2. *Tampering*: Tampering can be done while data is on the communication channel, while data resides on the consumer

machine, or while it resides on the provider machine. For web services in particular, targets for tampering are the SOAP and WSDL files, the executing code at both consumer's and provider's side, and application specific data on the consumer or provider.

3. *Repudiation*: Repudiation threats are by nature application-specific and are not further detailed here. Web services do provide countermeasure technologies here, such as XML signatures.
4. *Information disclosure*: Information can leak during communication, or while being stored on consumer or provider machine. Similar to the tampering threats, targets for information disclosure are the SOAP and WSDL files, the executing code at both the consumer's and provider's side, and application specific data on the consumer or provider.
5. *Denial of service*: Denial-of-service attacks try to disturb the services by overloading the communication line, or by enforcing a crash or ungraceful degradation of the consumer or provider.
6. *Elevation of privilege*: An elevation of privilege can occur on both the consumer's and producer's machine.

3.3 Attack entry points

On the architectural overview in Figure 2, the possible places where there can be a web service consumer and provider combination are indicated. For each of these web-service instances, each of the generic threats discussed in the previous section is potentially relevant, thus leading to a very high number of potential threats.

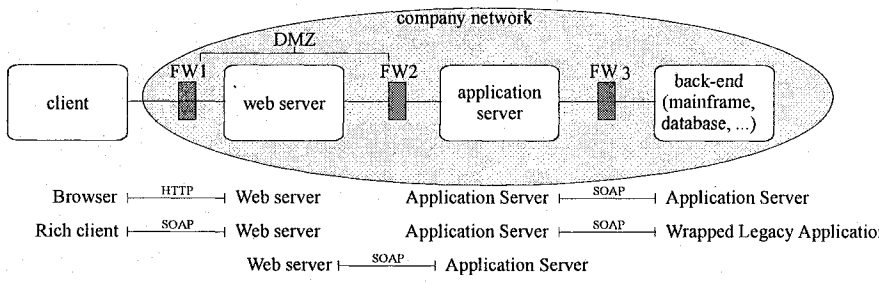

Figure 2. Attack entry points.

4. LIST OF THREATS

In order to keep the list of identified threats reasonable in size, we present only the most relevant threats in this section. For those threats, only a short overview is given here. More details can be found in [14].

To be able to identify the most relevant threats, we make two assumptions. Firstly, we assume that the company network and the servers are secured according to best practices. We do take into account that an internal attacker might get company network access, but with no privileges on any of the server systems. As a consequence, we consider it unlikely that an attacker can get direct access to state kept on any of the server machines. (Of course, indirect access is still possible, e.g. an application exception can leak information to clients.)

Secondly, we assume that attacks will be directed to the server. We do not consider attacks to the client. The rationale for this is that the web application designer/architect typically is concerned with protecting server assets, and does not have much control over the client software anyway.

The threat analysis is done on web applications, running on a Microsoft platform, as introduced in Section 4 of [1].

4.1 Spoofing

Given the possible instances of web services within the web application, the scenario where the client is spoofed in its communication with the web server is considered the most relevant. Weak or no authentication of the client can lead to unauthorized access to the web service.

Two other relevant spoofing threats can occur if the web service crosses a trust boundary. If the DMZ cannot be trusted, there could be a spoofing threat between the web server and the application server. If the application server communicates with a remote application server, there is a considerable spoofing threat in both directions (see [12] for further information).

4.2 Tampering

The highest risk for tampering exists at the client side. An attacker can tamper with all assets residing on the client machine or traveling over the HTTP channel. This leads to the following threats that are considered most relevant in this category.

- A SOAP message is replayed, leading to the unintended duplication of a server action or to inconsistencies on the server.

- A SOAP message is tampered with or maliciously constructed, leading to a whole variety of problems on the server side, such as information disclosure due to thrown exceptions or violations due to malicious input (e.g. SQL injection attacks to the database).
- The WSDL-file sent to the client, containing essential contact information (such as URLs) is tampered with. Changing this information can mislead the client.
- The rich client is reverse engineered and modified. In a rich client scenario, an attacker can gain valuable information by analyzing the browser extension sent to the client. Modifying this extension can enable an attacker to bypass input validation checks, or to construct malicious SOAP calls.

Depending on the context of a particular application, the threat of modifying state information on the servers could be important, but is not further detailed in this document. In particular in the scenario where the web application allows remote upload (or modification) of the content or functionality of the web application, modification of the state information on the server could be an important threat.

4.3 Repudiation

Repudiation threats are by nature application-specific and are not further detailed here. Web services do provide countermeasure technologies here, such as XML signatures.

4.4 Information disclosure

The highest risk for information disclosure exists again at the client side. An attacker can read all assets residing on the client machine or traveling over the HTTP channel. This leads to the following threats that are considered most relevant in this category:

- SOAP messages are disclosed, possibly leaking application specific information such as credit card numbers to an attacker.
- WSDL files are unnecessarily disclosed, giving the attacker information about the application structure.
- Web service implementation leaks information about application internals, for instance by sending stack trace information on errors.

Depending on the context, additional threats that are not detailed in this document could be relevant. In particular, weak host or network security

could lead to disclosure of web services specific information such as the files containing the web service code (the .asmx files).

4.5 Denial of service

We consider server denial of service the most relevant threat in this section, causing the server to crash or to degrade ungracefully because of a malicious SOAP call.

In addition, sending a client a malicious assembly in a rich client scenario could do denial of service on that client. Also communication overload could be a threat.

4.6 Elevation of privilege

Again, our focus is on elevation of privilege on any of the servers. We consider the most relevant threat to be the scenario where a web service wraps a legacy application. This can possibly expose legacy software vulnerabilities: the wrapping web service essentially provides a communication path from the Internet to an application written without this connectivity in mind.

5. DESIGN GUIDELINES FOR COUNTERMEASURE SELECTION

A multitude of security technologies is available to counter the threats identified. One of the key challenges for a designer is to make a sensible selection of such countermeasure technologies. In this section, we give an overview of countermeasure technologies, and we provide guidance on how to select appropriate technologies.

The guidance is structured as follows: for each kind of countermeasure we summarize the issues and questions a designer should keep in mind while selecting a technology for implementing that countermeasure, and we give a short overview of the available technologies with their properties.

5.1 Authentication

Authentication counters spoofing threats.

Questions/issues:

- Do you want to authenticate a user or a machine?

- Do you want entity authentication or message authentication? Entity authentication provides evidence that a given entity is actively participating in this communication session, while message authentication provides guarantees about the originator of a message.
- Do you need to propagate the authentication through delegation? If your service relies on other services, you may need to authenticate the client to the other services. Not all authentication technologies support this kind of delegation.
- What assumptions can you make about the authenticated party (e.g. can you install software on the authenticated party's machine)?
- What is the number of users? Some authentication technologies scale better than others.
- Does your application need access to authenticated identities? Some authentication technologies do not provide an API to retrieve authenticated identities at the application level.
- Do you need to integrate in an existing infrastructure? If an authentication infrastructure is already in place, it is probably a good idea to reuse it.
- Security versus ease-of-use? Security mechanisms that are not easy to use can cause the end users to either make mistakes or ignore them altogether.
- Related to data protection and authorization needs: authentication is often done as a precursor to authorization. So make sure authentication and authorization technologies work seamlessly together. Similarly, data protection is often combined with authentication.

Available technologies:

- At the network level, use IPsec. IPsec authenticates machines, but does not provide an API for passing identities to applications. IPsec requires OS support (available from Windows 2000 and up).
- At the transport level, use any of the HTTP authentication mechanisms (basic, digest, forms, passport, integrated windows, or SSL client certificate). For a discussion of the advantages and disadvantages of each of these authentication mechanisms, see [6].
- At the application level, use WS-Security, or XML digital signatures on SOAP messages. XML digital signatures can provide message authentication, but require an infrastructure to manage client certificates.
- Single-sign-on infrastructures such as Microsoft Passport can support the web application in authenticating the client.

- An intermediate authentication and/or authorization server can be used within the web service flow to check the user identity and to approve credentials.

Example designs:

- Basic HTTP authentication over an SSL protected channel is often used for client to web server authentication.
- IPsec is a good choice for mutual authentication between web server and application server.

5.2 Data protection

Data protection counters tampering and information disclosure threats for data in transit.

Questions/issues:

- Do you need selective encryption? Is it feasible to protect all content in the same way, or do some parts have different protection requirements than other parts?
- End-to-end or hop-by-hop? Are all intermediates that process the messages trusted, or do you need protection from potentially untrustworthy intermediates?
- Do you cross a Network Address Translation (NAT) device? Some data protection technologies cannot cross NAT boundaries.
- Related to authentication mechanism: often session keys for data protection are negotiated as part of the authentication process. Make sure you keep in mind these dependencies.

Available technologies:

- At the network level use IPsec/ESP. This is hop-by-hop, non-selective data protection. IPsec does not mix well with NAT.
- At the transport level use SSL or RPC Packet Privacy. Again hop-by-hop, non-selective data protection. SSL can cross NAT boundaries.
- At the message level, use XML encryption of (parts of) SOAP messages. This is the only technology providing selective protection and end-to-end protection.

Example designs:

- SSL is the typical choice for data protection between client and web server.
- Message level protection is needed in some multistage processing scenarios.

5.3 Authorization

Authorization counters tampering and information disclosure on data residing on servers. Authorization can also counter elevation of privilege or denial of service.

5.3.1 Questions/issues:

- What information do you need to make authorization decisions? Do you base access control decisions only on authentication information, or also on application state information?
- What is the granularity of the assets you are protecting access to? Do you need to control access to the application, or to specific functionalities within the application, or to specific objects in the application?
- Do these objects that need protection map naturally on operating system, web server or database resources?
- Do you need to integrate in an existing infrastructure?
- How will the access control policy be managed?
- Authorization technology is related to the authentication mechanism (and identity flow), as discussed in section 5.1.

5.3.2 Available technologies:

- At the machine level, by restricting access to a set of IP addresses (using IPsec, IIS or a firewall). This is a very coarse-grained access control. Keep also in mind that IP addresses can be spoofed to fool IIS access control.
- At the URL level, by configuring IIS. IIS can leverage the Windows access control mechanisms for restricting access to web server files.
- At the application server level, by using .NET or COM+ mechanisms for role-based access control.
- In the application code itself: application code performs the necessary authorization checks, possibly calling a centrally managed authorization engine. See [7] for a detailed discussion of application managed authorization.

- An intermediate authorization server can do access control or prove the client's authority.

5.3.3 Example designs:

- Each of the server machines could use IP-based access control to make sure the server machines are only accessible from expected machines.
- Role-based access control for protecting web application functionality from clients.

5.4 Input validation

Input validation potentially counters any of the STRIDE threats.

Questions/issues:

- As data flows from client to back-end or from client to other client, who will sanitize the data? Consider all data flows originating from an untrustworthy source and make sure they are validated somewhere.
- Is there a strict XML schema describing allowable input? If so, this can be used as a basis for validation. If not, provide a description of allowable input using other means such as regular expressions.
- Where does untrustworthy data go? If it goes to the database, SQL injection is a possible threat. If it is echoed to clients, cross-site scripting could be an issue. If it goes to a wrapped legacy application, there is a threat of buffer overflows.

Available technologies:

- Validating XML parser.
- Regular expression API's.

5.5 Other countermeasure technologies

We briefly summarize other countermeasure technologies. For more detail, we refer to [14].

- *Non-repudiation*: Non-repudiation counters the repudiation threat. This can only be done meaningfully at the application level. Possible technologies include XML signatures and application-level auditing.

- *Sandboxing*: Sandboxing counters elevation of privilege threats, and can be provided by the operating system (process separation) or by .NET Code Access Security.
- *Secure coding*: Secure coding counters all kinds of threats. It is not further discussed here, since it is not a design time countermeasure. See [13] for more information.
- *Intrusion/fraud detection*: Intrusion or fraud detection counters all kinds of threats. As a designer, the process of detecting intrusions or fraud can be made easier by providing good, application-level audit data.
- *Availability related countermeasures*: These countermeasures counter denial-of-service related threats. Available technologies include filtering (rejecting unacceptable requests as quickly as possible, e.g. by using firewall rules) and throttling (limiting the number of unauthenticated requests to your application).

6. CONCLUSION

Threat modelling and countermeasure selection are important steps in an engineering process for building secure software. Documenting the threats inherent in the use of specific technologies and guiding designers in the selection of countermeasures to these threats can make these steps significantly easier. This paper reports on the results of an analysis of the use of web service technologies for web applications from this perspective. The most relevant threats are identified, and rough guidelines on how to mitigate the associated risks are provided. Threats, vulnerabilities and risks are described informally. A potential direction for future work is a more formal description, for instance in a UML profile for risk analysis, such as CORAS [15,16].

7. ACKNOWLEDGEMENTS

This work reported in this paper was developed as part of the *Designing Secure Applications* (DeSecA) project, funded by Microsoft. Partners within this project are the Universita' degli Studi di Milano, the Technical University of Ilmenau, the University of Salford, and the COSIC and DistriNet research groups of the Katholieke Universiteit Leuven.

8. REFERENCES

[1] L. Desmet, B. Jacobs, F. Piessens, and W. Joosen. A generic architecture for web applications to support threat analysis of infrastructural components, Eighth IFIP TC-6 TC-11 Conference on Communications and Multimedia Security (CMS 2004), September 2004, UK, pp155-160

[2] D. De Cock, K. Wouters, D. Schellekens, D. Singelee, and B. Preneel. Threat modelling for security tokens in web applications, Eighth IFIP TC-6 TC-11 Conference on Communications and Multimedia Security (CMS 2004), September 2004, UK, pp 213-223

[3] R. Grimm and H. Eichstädt. Threat Modelling for ASP.NET – Designing Secure Applications, Eighth IFIP TC-6 TC-11 Conference on Communications and Multimedia Security (CMS 2004), September 2004, UK, pp175-187

[4] E. Bertino, D. Bruschi, S. Franzoni, I. Nai-Fovino, and S. Valtolina. Threat modelling for SQL Server, Eighth IFIP TC-6 TC-11 Conference on Communications and Multimedia Security (CMS 2004), September 2004, UK , pp189-201

[5] D. W. Chadwick. Threat Modelling for Active Directory. Eighth IFIP TC-6 TC-11 Conference on Communications and Multimedia Security (CMS 2004), September 2004, UK, pp203-212

[6] Microsoft Patterns and Practices: Building Secure ASP.NET Applications, Microsoft Press, January 2003.

[7] Microsoft Patterns and Practices: Designing Application Managed Authorization, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/DAMAZ.asp

[8] Microsoft Patterns and Practices: Improving Web application security: Threats and Countermeasures, Microsoft Press, June 2003.

[9] W3C Note, SOAP: Simple Object Access Protocol 1.1, May 2000, http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

[10] W3C Note, Web Services Description Language (WSDL) 1.1, 15 March 2001, http://www.w3.org/TR/2001/NOTE-wsdl-20010315/

[11] UDDI.org white paper, UDDI Technical White Paper, 6 September 2000, http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf

[12] Hartman, Flinn, Beznosov, Kawamoto. Mastering Web Services Security. Wiley Publishing 2003.

[13] Howard, LeBlanc. Writing Secure Code 2nd edition, Microsoft Press, 2003.

[14] Designing Secure Application project (DeSecA), final report, May 2004.

[15] M. Lund, I. Hogganvik, F. Seehusen, and K. Stolen. UML profile for security assessment, Technical report STF40 A03066, SINTEF Telecom and Informatics, December 2003.

[16] M. Lund, F. den Braber, K. Stolen, and F. Vraalsen. A UML profile for the identification and analysis of security risks during structured brainstorming, Technical report STF40 A03067, SINTEF ICT, May 2004.