# THREAT MODELLING FOR ASP.NET
## Designing Secure Applications

Rüdiger Grimm and Henrik Eichstädt
*University of Technology, Ilmenau, Am Eichicht 1, D-98 693 Ilmenau*

**Abstract**: This paper gives a security analysis of Microsoft's ASP.NET technology. The main part of the paper is a list of threats which is structured according to an architecture of Web services and attack points. We also give a reverse table of threats against security requirements as well as a summary of security guidelines for IT developers. This paper has been worked out in collaboration with five University teams each of which is focussing on a different security problem area. We use the same architecture for Web services and attack points.

**Key words**: web services; asp.net; client-server; security; threats; web application; data storage; threat countermeasures.

## 1. INTRODUCTION

A Web service is a network of coordinated applications in the backend behind an http-governed Web server. The Web server is addressed by http-clients across the Internet. ASP.NET is one example for the coordination technology. However, the security analysis holds for Web services in general, not only for ASP.NET.

ASP.NET provides a set of components for developers to implement complex functionality in DLL. It is scalable, in that it provides state services to manage session variables (cookies, session ids, temporary URLs) across multiple Web servers in a server farm. It is stable, in that it can detect application failures and recover from them. It addresses both "managed code" (conformant to ASP.NET), as well as "unmanaged code" ("native code") to include "legacy" applications. It is performant, because ASP.NET pages are compiled whereas ASP pages are interpreted. When an ASP.NET page is first requested, it is compiled and cached, or saved in memory, by the

.NET Common Language Runtime (CLR). This cached copy can then be re-used for each subsequent request for the page. After the first request, the code can run from a much faster, compiled version, see Butler, Caudill [1] for details.

In this paper we will use an abstract Web services model which allows us to identify different sources and targets of attacks. On the basis of our attack analysis we will provide a structured view on security guidelines which help developers to avoid the most obvious security holes. The security holes derive mainly from the fact that any kind of Web service resides within the open world of Web usage. They are not specific to ASP.NET. However, ASP.NET is an obvious example of a Web services framework.

## 2.      ARCHITECTURE PREREQUISITES

We will base our security analysis on a rather abstract structure of ASP.NET technology which we will refer to as our Web services model. It consists of these four building blocks, which could reside either on the same or different hardware components:

1. a 'pure' ASP.NET component (which serves as a 'gate' between the web server and source code; external components can only be connected from this ASP.NET-component);
2. an 'external' component built with C#, VB or any other language using the Common Language Runtime (CLR); this is so-called 'managed code';
3. an 'old', external component being integrated into the Web service – possibly not integrated into the Common Language Runtime (CLR); this is so-called 'unmanaged code';
4. database(s).

Our Web service structure is a refined version of the architecture model in [2]. We have explicated the application server part by adding application details and communication relations between the components. The structure is shown in Fig. 1 below.

We will not analyse the internal functional structure of the four components any deeper. In this sense we will consider the Web services on the ASP.NET technology as a 'black box': it reacts on input data (both stored and communicated), and it creates some output data (both stored and communicated). Therefore it is inserting, updating, checking and/or deleting data of any kind.

In this Web services model, several assumptions are made which are to be respected by application security policies in the first place. First, our focus is on ASP.NET technology, therefore we address only the Web

service. Other services such as FTP, Sendmail, or Telnet are also security relevant, but out of scope of this paper. Furthermore, we assume that the Web Server is organised as follows.

1. A firewall protects the Web Server from the Internet which contains a positive list of ports and protocols to be accepted.
2. The Web Server accepts and responds to 'valid' http(s)-requests only;
   a) 'valid' are requests with correct http syntax, and the URLs of which are within an explicitly accepted name space;
   b) 'validity', however, does not refer to parameter content; on this level, parameter content is not checked and will therefore be addressed by our attack analysis below.
3. On an operating system level the Web service is configured according to these minimal security requirements:
   a) only a minimal set of components and applications is installed: e.g., if not explicitly needed, no ssh / sendmail / telnet / ftp etc. service is addressable through this Web Server; no client browser is available within the Web service;
   b) rights management within the Web service follows the least privileges principle for the relationship 'userid → application';
   c) a minimal set of users (potential attackers) has access to the internal network: with respect to the relationship 'persons → userid'.

## 3. ATTACK ENTRY POINTS

### 3.1 SOURCES AND TARGETS

In order to identify attack points, two aspects are to be addressed: sources ("who attacks"), and targets ("what is attacked?"). Attackers may reside inside or outside the server ("sources"). They may aim at assets of the server or of the client ("targets"). Servers do not only organise their own assets, but also assets of clients, for example account levels, private information, or an achieved status of a purchase. An attack on the server can, therefore, also be an attack on a resource inside the server which represents a client's asset. It is in the interest of the server to protect both its own assets (e.g. received payment), as well as the assets of its client as far as it is responsible for them. Otherwise, a server will lose reputation, or even be liable for losses of its clients.

## 3.2     ATTACK POINT SOURCES

Attacks can be pushed from outside as well as from inside the Web service-network. In a refined view of the Web services model, presented below in Fig. 1, the following six attack sources can be identified:

1. attack from an external aggressor via the standard http(s)-gate
2. Web service attacking the client (delivering malicious code, misuse of personal data)
3. attack from an external aggressor circumventing the ASP.NET gate (=> firewall and webserver are not secured properly)
4. attack from an internal aggressor via the internal network
5. security risks by connecting unmanaged code (native) applications
6. attack from an aggressor application nested inside the Web service structure. This could be any kind of application as database, (web)server, operating system program or any other application.
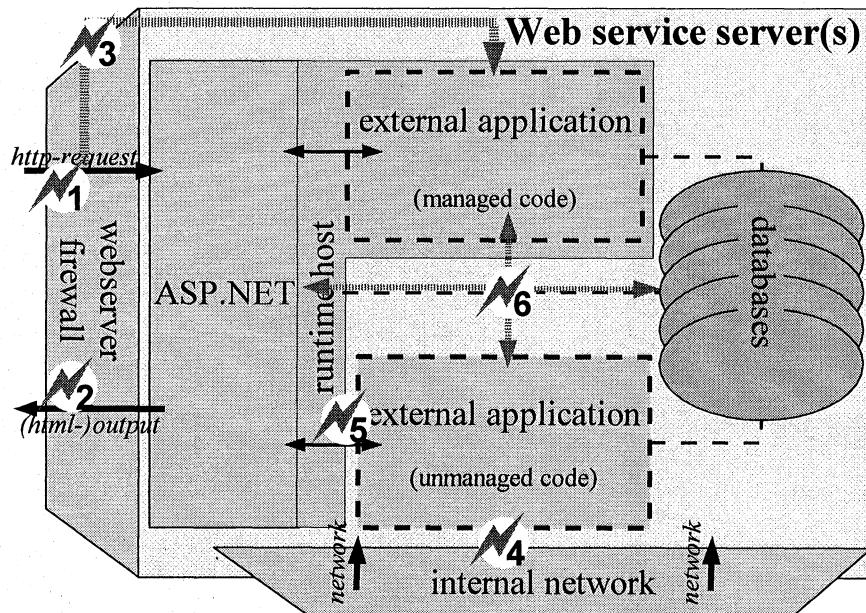


*Figure 1.* Possible attack points on Web services

There are two general directions of attacks:

- *Server is being attacked*: The target of attacks of a client against the server may be **(a)** the assets of the server, e.g., the client retrieves an electronic good without paying for it. Or the target may be **(b)** the assets of another client through a manipulated server, i.e. by retrieving an electronic good which another client has paid for and who is then

prevented from receiving it (impersonation, or stealing of privilege). Case (b) is an attack against the client via an attack against the server, i.e. the attacked client will observe an attack of the server, while the server itself was attacked in order to attack the other client.

• *Client is being attacked*: The target of attacks against the client that will be considered here are always the assets of the client, e.g. his privacy, his money, his knowledge, his privileges, etc.

The following kinds of attack are out of scope of this paper: Servers which attack clients in order to attack other servers through the manipulated client. This can be done either by using the clients' credentials, or by using client resources. The aim would be to enforce attacks (DDoS), or to blame the service provider for an insecure service.

Why is it important to analyse attacks on the client by the server? At first glance, the server is simply regarded as decent, and so there is no point in considering this case. However, there are two reasons why a server (and even more a decent server) is interested in protecting its clients against attacks through the server:

1.  Servers want to protect their reputation against suspicion. For example, shops provide read-access to purchase status points, they provide read- and write-access to personal data of their clients (each client only accesses his data), they reveal their privacy policy, they offer privacy mechanisms like P3P, they sign their parts of obligations (like payment receipts), etc. Servers will also have to make clear to their clients that insider attacks are minimized, e.g. by 4-eyes-principle access rules, or other security-certified mechanisms.

2.  Servers must be aware that unauthorised intruders (outside attackers) compromise the system in order to attack other clients. From the point of view of a server, this is an outside attack against the server, covered by analysis in case 1 above. From the point of view of the client, however, this is an attack of the server on his assets. Servers must make clear how they minimize this danger.

## 4.    ASSETS VS. ATTACK TARGETS

For a general security analysis of the ASP.NET technology, no concrete assets can be identified, because the technology is not restricted to a specific type of application. Instead, we consider general, abstract assets being in the focus of attackers. In supplement to the Microsoft STRIDE-categories (see [3]) of attack targets, we suggest to introduce the common IT security requirements (see [4][5]) as abstract assets of all ASP.NET services:

- Availability
- Confidentiality
- Integrity
- Authenticity
- Accountability (non repudiation)

These requirements can be understood as abstract assets. The assets can be mapped one-to-one on threats, in that threats are understood as negative requirements.

## 4.1    IT SECURITY REQUIREMENTS VS. STRIDE ATTACK TARGETS

The same requirements (assets) can be broken (enacted on) by multiple attacks [7]. The abstract assets (i.e. the IT security requirements) can be mapped on the STRIDE categories of attack targets via this matrix:

*Table 1.* common IT security criteria vs. Microsoft's STRIDE concept

|                        | Availability | Confidentiality | Integrity | Authenticity | Accountability |
|------------------------|:---:|:---:|:---:|:---:|:---:|
| Spoofing               | X | X |   | X | X |
| Tampering              |   |   | X | X | X |
| Repudiation            |   |   |   |   | X |
| Information disclosure |   | X |   |   |   |
| Denial of service      | X |   |   |   |   |
| Elevation of privileges| X | X | X | X | X |

## 5.    LIST OF THREATS / ATTACKS

There are three ways to structure attacks:
1. Attack points oriented (as in Fig. 1 above)
2. Assets/Threats oriented (IT security requirements as abstract assets)
3. Attacks oriented (Microsoft's STRIDE)

We have introduced attack points in Fig. 1 above in the section on attack entry points. Assets/Threats were introduced by abstract IT security requirements in the previous section on assets vs. attack targets. Attacks are introduced by the STRIDE model. In Table 1 of that section we have mapped assets (IT security criteria) on attacks of STRIDE type.

As the same threats can be enacted by multiple attacks, we have decided to follow the attack points orientation, because there is least redundancy - the following list of threats is thus ordered by attack points. In order to keep up with the STRIDE structure, we offer a reverse table of attacks vs. our attack numbers at the end of the attack list in this section.

The threats in this list are numbered according to the following scheme: AST represents 'ASP.NET Threat', the first digit refers to the attack point, the last two digits represent the numbering of the threats inside the attack point. Appended to the threat title is the STRIDE-classification displayed by the initial letter(s) of the applicable STRIDE-category(ies) in parenthesis.

There is one type of threat which can be realized at any point of the Web service. This threat may be a side effect of the other threats listed below or may be applied as preparation of any other threat.

**AST001: Provoke errors to reveal system information (I)**

*Description*: The attacker 'misuses' the Web services to provoke the generation of error messages. These messages can be used to gather detailed system information for further attacks.

*Countermeasures*: Only general/generic error messages should be visible to the client and should not disclose any specific information about the internal system and the nature of the error. Detailed error messages are to be written into a logfile.

## 5.1     Attack point 1: External Attacker

The most likely way to attack a Web service is to construct input data contrary to the intention of form(field)s.

### 5.1.1     AST101: any input data is sent to the application (RI)

*Description*: The attacker fills in data into html-forms that is not intended by the application. False information and executable code could be used to manipulate the application.

*Countermeasures*: A server-side validation of input data is necessary. Use .NET validation server controls for this task. Additionally, storage of 'false data' can be avoided by checking the data against a 'valid' database. As a minimum, SQL-Syntax should be denied.

### 5.1.2    AST102: Manipulating form parameters (TRID)

*Description*: False input data is carefully crafted and sent to the server by manipulating the http-request (either by building a URI with parameters [GET-method] or manipulating the http-body [POST-method]).

*Countermeasures*: In addition to the AST101 countermeasures, the form data could be checked to be sent by the POST-method (if action is set to be POST). The session-identifier should be authenticated with additional data (e.g. IP-address) and/or the application should re-ask for authentication credentials in case of critical actions.

### 5.1.3    AST103: Uploading malicious program code (STRIDE)

*Description*: Some code file containing malicious code is uploaded using an upload form. Subsequently the attacker gets to know the save folder on the server and tries to execute his code or the uploaded file is processed by the server and thus executed.

*Countermeasures*: The execution of uploaded files has to be denied (either stand-alone execution on the server or HTML-inline execution). Uploaded files should be validated not to be code, the target folder for upload files should be secured - no (direct) access via http. Uploads could be filtered by denying file types with possibly included code/allowing file types from a positive list.

## 5.2    Attack point 2: Web service attacking client

Though no direct threat to the server, this threat is mentioned because it represents a threat to the servers trustworthiness.

### 5.2.1    AST201: non-transparent data gathering (I)

*Description*: The Web service collects data from clients (required form input) that is not or barely necessary for the applications purpose.

*Countermeasures*: Forms should be constructed in a way that only minimal, necessary data is required. In addition, transparency tools can be used (P3P: publish privacy policies, privacy audit label).

### 5.2.2    AST202: Web service delivers malicious code (STIE)

*Description*: The Web service creates and sends code that forces the client to crash. As this could (but needs not) be code created by an attacker, AST103 is a possible origin of this threat.

*Countermeasures*: Carefully create Web service (HTML-/script-) output and avoid additional/plugin media/technology where possible. Before delivering any code, check if the client supports the needed technology and offer alternative technology.

## 5.3    Attack point 3: ASP.NET-gate circumvented

As ASP.NET serves as 'gateway' to the web applications, threats could be possible by contacting applications without ASP.NET intervening.

### 5.3.1    AST301: Reveal location of subordinate application (I)

*Description*: An attacker causes the server application to generate output that exposes the location of subordinate applications, databases etc.

*Countermeasures*: Force the applications to receive input data through a central, filtering application that is redirecting the data on the server-side.

### 5.3.2    AST302: Execute subordinate application directly (STRIDE)

*Description*: An attacker executes a subordinate application that is intended not to be executed from outside.

*Countermeasures*: This should not be possible due to the prerequisites. Subordinate applications should be configured not to be executed from outside the Web service. Establish trust management on the server by defining a 'need-to-know' access rules matrix for internal applications (read/write).

## 5.4    Attack point 4: Internal aggressor

The Web service files could be accessed by an internal attacker being connected to the company network. As a consequence, the www-interface is avoided by accessing the Web service structure from the internal network.

### 5.4.1    AST401: Accessing applications with internal authentication data (STRIDE)

*Description*: An attacker from inside the network accesses/executes components with 'insider' privileges. Manipulation of data/communication could be possible with those access rights.

*Countermeasures*: Web service components should only be executed with restricted privileges. Additionally, a sophisticated rights management prevents execution by real users (run as special, 'virtual' user only). Using 4-

eyes principle access rules for very sensitive actions and data secures those areas.

### 5.4.2    AST402: Accessing stored data (TRIDE)

*Description*: An attacker from inside the network accesses data stores (file system, data base) to get information. The data stored is accessed directly through OS means, not via ASP.NET.

*Countermeasures*: Access to data stores should only be permitted to Web service components.

### 5.4.3    AST403: Manipulating source code (TIE)

*Description*: An attacker from inside the network accesses the file system and manipulates the source code files.

*Countermeasures*: Limit write access, Sign/create hash values for component files and deny execution if authentication of component fails. Additionally, the changes of source code can be logged (logfile, notification mail etc.).

## 5.5    Attack point 5: Unmanaged Code

As ASP.NET allows the integration of a broad variety of applications, also 'old' code can be used in ASP.NET-based Web services. It is then necessary to exchange data between these (un)managed code components.

### 5.5.1    AST501: Inconsistent data (ID)

*Description*: When components running outside the CLR are used, data-types have to be converted but can't always be mapped 'perfectly' between these components. An attacker could use this to cause components to malfunction (by generating a complex piece of data).

*Countermeasures*: Native code should be ported to .NET code ('partial port' approach according to [6]) and/or the critical native code should be rewritten. Exception handling to catch wrong data types should be implemented.

## 5.6    Attack point 6: Aggressor application inside Web service

If some application inside the Web service structure is used to attack the Web service, this set of threats is conceivable.

### 5.6.1 AST601: Revelation of data from inside (I)

*Description*: Web service data-store components / store controls can easily be contacted by a 'hi-jacked' component to retrieve stored data.

*Countermeasures*: Implement system integrity checks (viruses, Trojan horses), establish a 'need-to-know' access rules matrix for internal applications (read/write) and define a strong access control. Additionally check if request originally comes from 'outside' and is generated by a http-request through the official, allowed routes. Sign components to prevent manipulation of the hi-jacked components' source code.

### 5.6.2 AST602: Manipulation of data from inside (STRIDE)

*Description*: Web service data-store components / store controls can be contacted by a 'hi-jacked' component to manipulate stored data.

*Countermeasures*: see AST601

### 5.6.3 AST603: Contacting Applications from inside (STRIDE)

*Description*: Web service applications can be contacted by a 'hi-jacked' component (Trojan horse or stolen privileges). The component can request the service of other applications without any outside-triggered need.

*Countermeasures*: see AST601. The 'need-to-know'-rules matrix has to be expanded to cover the execution of components.

### 5.6.4 AST604: Revelation of configuration information: (I)

*Description*: If client software (e.g. Browser, .NET Interface) is run on the server, it is possible for an attacker to gather configuration information (OS version, .NET runtime version etc.).

*Countermeasures*: Deny client software to be run on server.

### 5.6.5 AST605: Buffer overflow (STRIDE)

*Description*: Buffer overflows are used to attack the server. Especially Web service components are permanently cached in the CLR using one memory area. This can be used to cause Buffer Overflows and get access to the Web service.

*Countermeasures*: Cause caching to refresh periodically and implement Buffer Overflow checks. Avoid using unmanaged code.

## 6. REVERSE TABLE OF ATTACKS

*Table 2.* ASP.NET-Threats / STRIDE matrix

| | S | T | R | I | D | E |
|---|---|---|---|---|---|---|
| AST001 | | | | X | | |
| AST101 | | X | X | X | X | |
| AST102 | | X | X | X | X | |
| AST103 | X | X | X | X | X | X |
| AST201 | | | | X | | |
| AST202 | X | X | | X | | |
| AST301 | | | | X | | |
| AST302 | X | X | X | X | X | X |
| AST401 | X | X | X | X | X | X |
| AST402 | | X | X | X | X | X |
| AST403 | | X | | X | | X |
| AST501 | | | | X | X | |
| AST601 | | | | X | | |
| AST602 | X | X | X | X | X | X |
| AST603 | X | X | X | X | X | X |
| AST604 | | | | X | | |
| AST605 | X | X | X | X | X | X |

## 7. DESIGN GUIDELINES

1. Validate input data on the server side
   - Validating on the client side is nice for the user, but insecure for your application because it can be bypassed
   - Server-sided validation can't be by-passed, check input data to be of the correct range, expected length and uploaded files to be of the correct data type or among correct/allowed file types. For this validation, use the .Net validation controls.
2. Do not insert data again that was input and validated before, without new validation (only insert and check deltas to previous state)
3. Establish a 'need-to-know' access rules matrix
   - Which application may connect to another application/to a data storage
   - Which application may request what kind of data
   - Which application may modify data
4. Sign components and check correct signature (checksum, hash) within the Web service structure
5. Make sure data can only be accessed via a Web service component
   - Especially do not allow direct request of uploaded files
   - Deny execution of uploaded files
6. Re-ask for credentials if a critical action (modification of user data) is requested

7. Use managed code wherever possible
   - Convert 'old' applications
8. Minimize forms
   - Collect only data that is needed by the application – the user will begin to trust your application
9. Do not reveal internal (configuration / system) data to user
   - Error codes delivered to the user have to be generic ('an error occurred', if it is that bad)
   - Log error codes in detail to a logfile on the server in a secure folder
10. Have a 'pure' server (see prerequisites section)
    - No client applications installed
    - Only needed services installed / active

## 8.    ACKNOWLEDGEMENTS

## 9.    REFERENCES

[1] J.Butler, T. Caudill - ASP.NET Database Programming - Weekend Crash Course (John Wiley & Sons Inc, 2002)

[2] 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

[3] MSDN Library - Improving web application security: Threats and Countermeasures
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/ThreatCounter.asp, 2003

[4] Information Technology Security Evaluation Criteria (ITSEC):/ Provisional Harmonised Criteria. Luxembourg: Office for Official Publications of the European Communities, 1991. Bundesanzeiger Verlagsges., Köln 1992

[5] The Common Criteria for Information Technology Security Evaluation (CC) version 2.1, Sep 2000. Part 1 - Intro & General Model; Part 2 - Functional Requirements; Part 3 - Assurance Requirements. Standardised as ISO/IEC 15408 1999 (E), available from http://csrc.nist.gov/cc/

[6] Don Box - Migrating Native Code to the .NET CLR
http://msdn.microsoft.com/library/default.asp?url=/msdnmag/issues/01/05/com/toc.asp, 2001

[7] L. Desmet, B. Jacobs, F. Piessens, and W. Joosen. Threat modelling for web services based web applications. Eighth IFIP TC-6 TC-11 Conference on Communications and Multimedia Security (CMS 2004), September 2004, UK. pp 161-174