INFORMATION
SYSTEMS IN
MANAGEMENT

Information Systems in Management (2014) Vol. 3 (3) 170−181

SECURITY MECHANISMS FOR DATA ACCESS IN ASPECTS OF TOOLS AVAILABLE IN .NET FRAMEWORK

ANETA MICHALSKA, ANETA PONISZEWSKA-MARANDA

Institute of Information Technology, Lodz University of Technology, Poland

Software solutions are nowadays commonly used in business. More and more transactions are conducted on-line as well as more and more critical information is being kept on local or remote servers in huge databases. The purpose of presented paper is to analyze and propose the solutions used for the security of sensitive personal data and access to such data provided by the platform chosen for research with respect to the real-life needs of the developers and end-users. The main focus are put on the solutions provided by the .NET platform which next to Java is one of the most commonly used programming environments for both web page and desktop applications.

Keywords. security mechanisms, data security, data access, .NET platform

1. Introduction

Together with the rapidly evolving environment of technological advances which aim to simplify and accelerate the business and production processes, an increasingly important issue becomes the development of appropriate security measures which would provide essential protection of intellectual property. Software solutions are nowadays commonly used in business. More and more transactions are conducted on-line as well as more and more critical information is being kept on local or remote servers in huge databases. The increased availability of information as a whole became a threat for confidential information and that is why the necessity to assure the security of sensitive data became undeniable. Develop-

ers of information systems put more and more stress on the aspect of security, as they have to ensure that their product will comply with international and local standards of personal data protection and it will guarantee the customers safe storage and use of data. On the other hand the platform providers try to equip their software designed for developers with built-in security mechanisms and frameworks in order to facilitate the process of software production [1].

The purpose of presented paper is to present the solutions used for the security of sensitive personal data and access to such data provided by the platform chosen for research with respect to the real-life needs of the developers and end-users. The main focus will be put on the solutions provided by the .NET platform which next to Java is one of the most commonly used programming environments for both web page and desktop application projects. .NET framework offers the possibility to use several programming languages and approaches for creation of Internet applications, web pages and desktop programs [2]. The amount of sensitive data such as names, addresses, passwords, credit card numbers, which flow through the net, is enormous and constantly exposed to falling into the wrong hands.

The paper is structured as follows: the first part presents the security mechanisms of .NET environment contributed to establishing fixed and stable position of .NET as a platform for Web application development. The second part deals with the used security mechanisms and their effectiveness in data protection, presenting the recommendations regarding the choice of .NET environment security solutions.

2. Security mechanisms of .NET framework

.NET framework is equipped with mechanisms giving the possibility of applying numerous techniques and a significant number of security name-spaces in order to enable the developer to build a secure program both in case of desktop and Web applications. The largest pressure is put on the Web application security solutions as these are those more liable to threats and violence of data and data access security rules. .NET framework distinguishes between two types of security connected with application design [3, 4]:

- user security (role-based security) and
- code security (code access security).

Both these types of application security are vital. The order of their importance is determined by the purpose which the application serves as well as the user requirements. User security aims to provide a managed access to application resources and operations available to the end-users basing on their privileges. On the other side there is code security which is similarly responsible for resource access and availability of operations but this time the application controls the code

which requests the permissions to these actions. This prevents untrusted pieces of code coming from suspicious sources to be granted access to application interior [7].

User security and code security are not excluding – they may be applied elementarily providing the application with doubled security of different kind. In short one may notice that user security corresponds to the identification of the end-user and answers the question who is using the application and which operations he can perform (Fig. 1), whereas code security tries to determine where did the code trying to gain access come from, who wrote this code and what operations can this code perform (Fig. 2). In case of code security it does not matter who uses the application and what type of account does he have. Code security is based on authorizing the application access to system resources, file system, registry, network, services and databases. The identity of the user is authenticated in the case of user security and permissions are authorized and granted basing on this authentication.

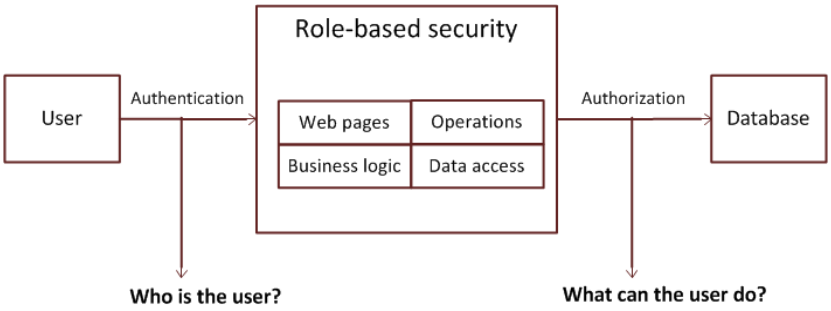

Figure 1. Schema of role-based security

2.1. Role-based security

As mentioned earlier the role-based security refers to the aspect of who can access application resources and which operations he can perform [5, 6]. This type of security is specifically used to authenticate and authorize the users basing on the roles assigned to the user accounts. The roles are determined basing on the business application of the program and they are specified particularly for the domain in which the program is used [7].

The *Principal* and *Identity* objects play the key role in this type of security for .NET platform. The *Principal* object is the reflection of the identity of the user and its membership to the roles. The interactions and principles of behavior of the *Principal* object are based on the *RolePrincipal* and *GenericPrincipal* objects. Major functionality of *Principal* object is that it stores the information about the user roles which determine the user permissions therefore it is attached to every request issued by the user to the Web application. This object can be retrieved using *HttpContext.Current.User* property.

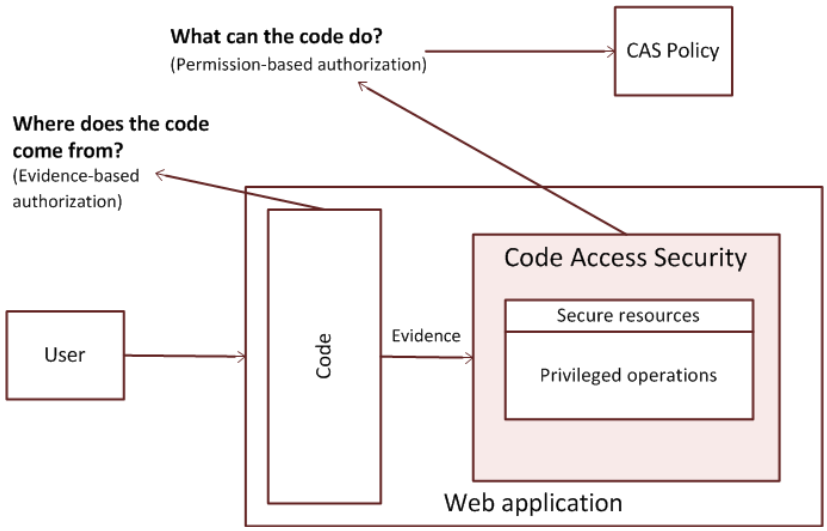

Figure 2. Schema of code-based security

The *Principal* objects store as a property the *Identity* objects. *Identity* objects are responsible for storing the user name, authentication type flag and authentication success or failure flag. Thanks to this information, the *Principal* objects are able to tell between authenticated, non-authenticated and anonymous users [7].

Another type of objects taking part in secure role-based authorization are *PrincipalPermission* objects. They specify the identity and role that the user has to possess in order to perform certain operation.

2.2. Code-based security

Beside the most common mechanisms of managing a user access basing on credentials authentication and resource access restrictions, another perspective needs to be taken into consideration as far as a security is concerned. This perspective embraces the code security as the protection of original source code of the application from the malicious software that this source code is vital for the correct operation of application and security of data it contains. The role-based security mechanisms do not correspond to threats which the application code faces. That is why another type of security based on the code access permissions needs to be applied.

.NET framework provides special mechanism called *Code Access Security (CAS)* which disables the code from unknown sources to penetrate and interfere with the application in an undesired manner [7, 8]. CAS is also helpful when it comes to dealing with the vulnerabilities and errors located in the source code itself. In order to create the applications complying with CAS standards the develop-

ers need to acknowledge and obey certain rules regarding the code composition. These rules refer to writing verifiable and type-safe code, using proper security syntax and secure class libraries.

CAS security means that runtime environment allows the code to perform only those operations it has permissions. The configuration of permissions granted to different parts of code enables to establish a security policy characteristic for every separate application. Security policy refers to a set of rules which can be configured and customized by the application developers. These rules enable the *Common Language Environment* to distinguish between parts of code of varying trust levels and assign appropriate permissions to these parts of code. The parts of code are called *code groups* and the entire code of the application may be divided into code groups according to different categories like for instance original URL addresses, publishers or digital signatures.

Describing the security for .NET platform in terms of CAS the concept of *Security-Transparent Code* arises. Security transparency means that the code should be divided into two separate isolated parts – the part which runs as *application* and the part which runs as its *infrastructure*. This enables to grant permissions to some pieces of code which is the so called *critical code*, which will be able to execute privileged actions such as calling native code, and other pieces of code which will not have such permissions.

3. Security mechanisms of .NET environment for Web applications

The security mechanisms available in .NET environment are commonly used in Web applications [7, 8, 9]. The example application – Internet portal created to analyze such mechanisms was written in ASP.NET technology.

The application, created in the framework of the presented works, was written using the combination of ASP.NET and C# language. It incorporates most commonly used security mechanisms available in these technologies. The crucial security aspects implemented in the application include user authentication mechanism realized by means of registration and login forms, authorization for resource access based on different roles assigned to users and sensitive data encryption using the chosen standards. The major focus was put to the user security as code security is a built-in feature realized equally in every .NET application.

The application is equipped with basic functionality characteristic for web applications, however the main focus was put on the implementations regarding the security issues. Applied security mechanism include the authentication and authorization mechanisms realized by means of login and registration forms, encryption of data and restricted access to the portal resources and operations.

The created application was designed to operate as a client-server application. The users would be able to send requests from their client computers to the host

located on a remote server where the application core and database would be stored. The system should be able to authenticate the users, authorize an access to the application resources, display the data using graphical interface and process an input given by the users.

From the point of view of the system and its administrators the vital aspect of the application is the insurance of security of stored data especially sensitive personal information such as name, personal number, address, card ID, PIN-code, and photograph to any unauthorized unit. It is also crucial to determine the acceptable response times of the system and security policy.

The major concern regarded the storage of user personal data. This aspect refers to almost any application having access to database and requiring authentication. .NET Framework developers identified the need to automate and unify the process of authentication and authorization and they introduced the so called *Membership* framework which is responsible for managing user accounts and roles. However, Membership framework provides only basic functionality and it has to be extended in order to comply with specific assumptions of the application.

Membership framework uses a pre-defined provider model in order to customize database features to a standardized programming interface. In order to adjust Membership framework features a custom provider was defined. The framework serves two built-in types of providers – *ActiveDirectoryMembershipProvider* and *SqlMembershipProvider*.

As the purpose of presented paper was to investigate the available solutions in field of .NET framework security mechanisms, the created example web application incorporates several mechanisms responsible for guaranteeing security and proper resource access to application users. The security policy is realized basing on the options referring to the most commonly applied security methods:

- authentication of registered users,
- authorization of access to resources and operations basing on privileges,
- sensitive data encryption,
- code access security.

3.1. Authentication

Authentication is the process of validating user credentials and assigning privileges basing on those credentials. Authentication takes place every time the user sends the request for protected resources or operations to the application server. The way the server authenticates users depends on the pre-defined configuration stored in *Web.config* file. The configuration takes place by specifying mode attribute of the <*authentication*> tag:

```
<authentication node = "Forms">
<forms loginUrl = "~/Account/LogIn"
timeout = "3000"
```

```
cookieless = "UseCookies"
protection = "Encryption"
requireSSL = "true"/>
</authentication>
```

The created web application uses the traditional *Forms authentication mode*. Configuration of features available in this mode is realized by defining the attributes of the <*form*> tag. Forms authentication is based on assigning the tickets to users who have been successfully identified. These tickets are sent to the application server each time the user sends a request for resource. Having valid ticket the user is perceived as logged in.

Tickets are most frequently stored in the cookies collection of a Web browser. It is also possible no to use cookies and to store ticket information in the URL. This is defined by setting the *cookies* attribute of <*forms*>.

Tickets are generated and issued to the user by the methods of *FormsAuthentication* class being a part of *System.Web.Security* name-space. The cookie containing the ticket is included in the header of any request sent to the server (Fig. 3). Another class of *System.Web.Security – FormsAuthenticationModule* is responsible for examining the header of each request in search for a cookie containing a valid ticket. In case no such cookie is found the module return a message with HTTP 302 Redirect status meaning that the user cannot access the resource because he is not logged in. In such case the user is redirected to the login page. Otherwise the authentication is confirmed and further check for authorization privileges takes place.

The above description implies that there exist three possible scenarios for a user trying to access a protected resource (Fig. 3). Either the user has a valid ticket so the authentication will be successful or he will be redirected to the login page where *FormsAuthenticationModule* will generate a valid ticket. The third option refers to the situation when the user login will end in failure.

Because of the fact that tickets are stored in cookies there comes the notion of timeouts. The cookies lose their validity after some time and so do the tickets contained in them. To define the time after which a ticket will become invalid one has to specify timeout attribute in the <*forms*> tag. This will increase the security of the application because a user will not stay logged in for indefinite amount of time which will prevent unauthorized units from using his accounts.

Other parameters specifying the security features are *protection* and *requireSSL* attributes. Boolean value of *requireSSL* indicates whether secure SSL connection is necessary during the authentication process. *Protection* enables to select type of security measure used to protect the ticket in the cookie. This attribute indicates how the ticket will be sent – either in plain text or using encryption. Encryption may be done in two ways. Either by sending encrypted ticket to the server or by generating *message authentication code (MAC)*. MAC is a special representation of data contained within the ticket. In case of using this type of protection both ticket – sent as plain text and MAC are included in the header of the request.

The server compares the received MAC with the text that came in. If the data correspond to one another than the server knows that the cookie was not modified.

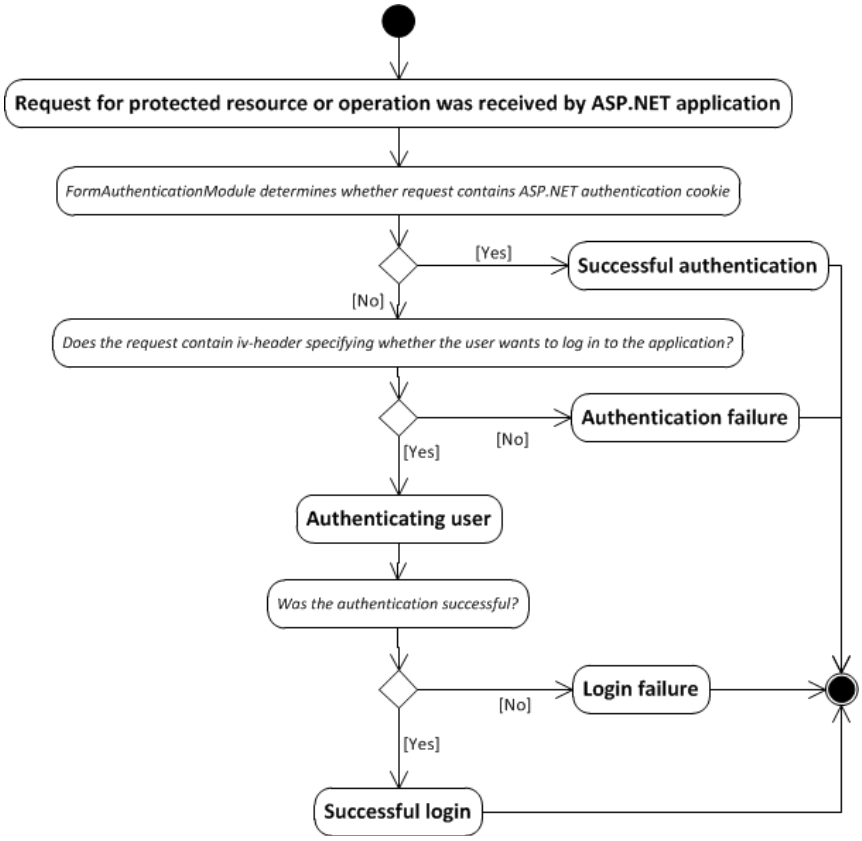

Figure 3. Activity diagram of authentication process

Authentication enables the server to tell authenticated users from guests and on this basis the authorization of access to resources and operations is granted.

3.2. Authorization

Authorization is the process of assigning the privileges to specific users, groups of users or actions. Basing on the user membership to defined the roles it is possible to determine which resources and operations he can access.

System.Web.Security name-space contains a series of classes responsible for the role managements. The core class *Roles* provides an interface for adding, deleting roles, assigning users to roles, retrieving all roles the user is assigned to, etc.

The *RoleManagerModule* is a class responsible for creation of *RolePrincipal* object during the authentication process and attaching this object to the context of the current user. Thanks to *RolePrincipal* object which is another security class, it is possible to extract information about roles the user belongs to by using *IsInRole()* method:

```
<roleManager enabled="true" defaultProvid-
er="SubsciptionPortalSqlRoleProvider" cacheRolesCook-
ie="true">
<providers>
<clear/>
<add name="SubsciptionPortalSqlRoleProvider"
type="System.Web.Security.SqlRoleProvider"
connectionString-
Name="SubsciptionPortalConnectionString"
applicationName="/" />
</providers>
</roleManager>
```

Another way to determine the user roles in runtime is to specify that role names for a user could be cached in a session cookie. This improves the performance of an application and can be done by setting the *cacheRolesInCookie* attribute of the <*roleManager*> tag. Similarly as in case of membership provider the <*roleManager*> determines the schema for managing roles. If we want to use a role-based authorization in our application the role manager ought to be enabled and added to the role providers list. Role provider refers to the database schema for managing the roles:

```
<location path="Customer/Basket.aspx">
<system.web>
<authorization>
<allow roles="Standard, Premium"/>
<allow users="*"/>
<deny roles="Distributor"/>
</authorization>
</system.web>
</location>
```

Authorization may be defined either in local *Web.config* files defined on the package level or specified globally in the configuration file which was used until now. The authorization may be defined at any level of resource complexity. It can be specified for entire application, for separate packages or single resources like web pages. To determine who is allowed to use the selected resource the properties of <*authentication*> tag need to be set. These properties allow and deny the attributes which are given one of three additional parameters: users, roles and verbs.

Users enable to define specific users which can or cannot access resource. *Roles* parameter allows restricting access for entire groups of users and *verbs* gives the possibility to choose one of the three values: "GET", "POST" and "HEAD" in order to restrict performing certain request. The *verb* parameter has to be always accompanied by users or roles unless we want to restrict operations for all users – which happens almost never. The parameters take user names or role names as values, however there is additional option to indicate all users by writing "*" or only anonymous users denoted by "?":

```
<location path="Customer/Discounts.aspx">
<system.web>
<authorization>
<allow roles="Admin, Premium"/>
<deny verbs="POST" roles="Standard"/>
<deny roles="Distributor"/>
</authorization>
</system.web>
</location>
```

The specification of resource we want to authorize is done by modifying the path attribute in <*location*> tag. The value of an attribute may be an address of a particular page or entire package. If particular user or role is neither denied nor allowed an access to the resource by default such access is granted:

```
<location path="Administration">
<system.web>
<authorization>
<allow roles="Admin"/>
<deny roles="Distributor, Standard, Premium"/>
</authorization>
</system.web>
</location>
```

The process of authorization is a simple one and occurs after sending request to the server. It results in one of two actions – either an access is granted or it is denied (Fig. 4). The authentication ticket stored in the header of a request either already contains user role or the server performs a check basing on obtained user identity.

If the user is anonymous and the page requires authentication because it is a protected resource, the user is redirected to the login page. This results from the *FormAuthenticationModule* default behavior which returns HTTP 402 Redirect status. In case of authenticated users the *AuthorizationModule* checks the user roles and the resource permissions and returns a successful authorization result by redirecting the user to desired resource or returns HTTP 302 Error status indicating that the user does not have adequate privileges.

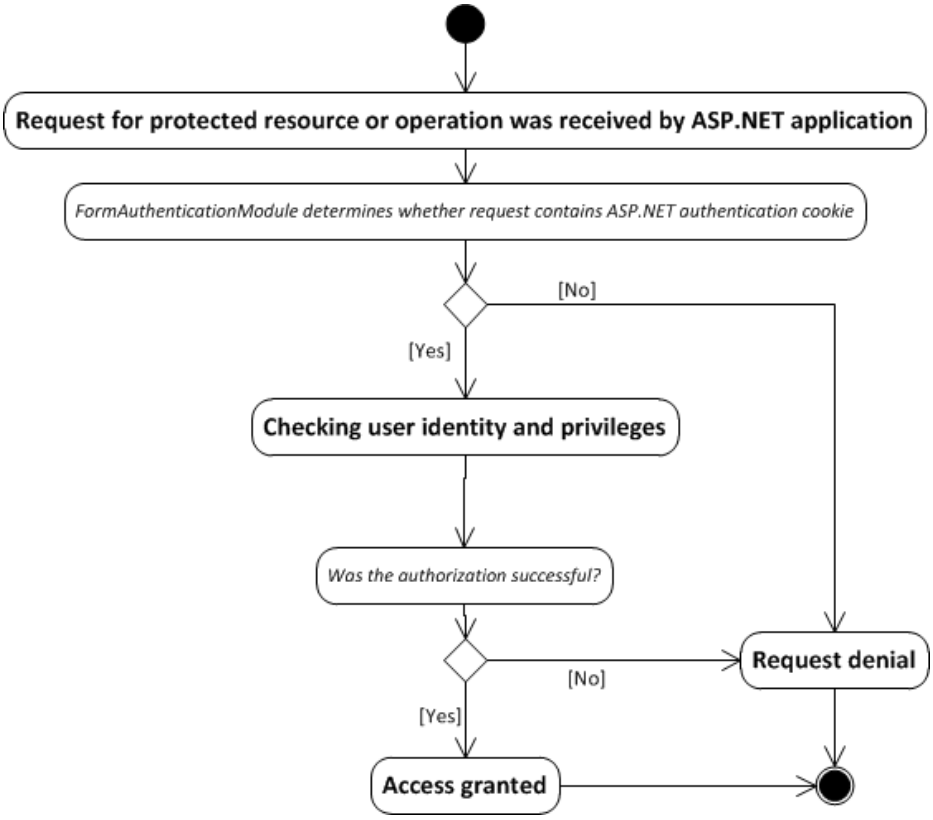

Figure 4. Activity diagram of authorization process

The authorization process is vital when it comes to managing the application resources in web application. It enables to distinguish between different types of users who are allowed to perform and access only these operations they should.

4. Conclusion

The main focus put on the solutions provided by.NET platform which next to Java is one of the most commonly used programming environments for both web page and desktop application projects. The above description demonstrates how to secure the application with basic security mechanisms in a simple and efficient manner. The implemented functionalities provide valid fundamentals for building a web application which would be resistant to the invalid user operations, provide effective distribution of tasks which depend on the user roles and privileges as well to the external attacks aiming to capture the sensitive personal data.

The authentication process which takes place every time is issued to the server allows validating the source of the request and its credibility. The built-in code access security mechanisms remain alert to the violation of the internal structure of the application and will not allow any untrusted piece of code to penetrate inside the application core. The unauthenticated user will not be allowed to get into privileged sections of the application. Moreover, the authorization process prevents the users from performing actions they are not allowed to perform and to access the resources which are beyond the scope of their rights. This ensures that sensitive data will only be visible to those for which they were designed for.

Basing on the conducted research referring to the security mechanisms for .NET platform it may be concluded that the tested environment provides the software developers with reliable tools for software protection offering a board variety of features which can be adjusted for the specific application purposes. It is recommended to study the requirements for the developed system in order to select the most suitable authentication and authorization methods as well as to include data encryption in every case where sensitive user information might be liable to any potential threat. It should also be remembered that user security should go hand in hand with code security as only the combination of these two will make the application reliable, efficient and resistant to accidental and deliberate violations of the security policy.

REFERENCES

[1] *State of Web Application Security*, Executive Summary, Ponemon Institute (2013).

[2] Pingdom AB (2013) *Internet 2012 in numbers*, Available at royal.pingdom.com

[3] A. Getman *The .NET Framework Security Model*, Available at www.codeproject.com/Articles/13947/The-NET-Framework-Security-Model

[4] Freeman A., Jones A. (2003) *Programming .NET Security*, O'Reilly Media, 2003.

[5] Sandhu R. S., Coyne E. J., Feinstein H. L., Youman C. E. (1996) *Role-Based Access Control Models*, IEEE Computer, Vol. 29, No. 2, pp. 38-47.

[6] Ferraiolo D., Sandhu R. S., Gavrila S., Kuhn D. R. , Chandramouli R. (2001) *Proposed NIST Role-Based Access control*, ACM TISSEC.

[7] Microsoft Corporation (2012) *Security in .NET Framework*, Available at msdn.microsoft.com/en-us/library/fkytk30f

[8] Microsoft Corporation (2012) *.NET Security Overview*, Available at msdn.microsoft.com/en-us/library/648652

[9] Freeman A., Jones A. (2004) *Guide to Microsoft .NET Framework Security*, National Security Agency.