UDC 004.492

Common Web Application Attack Types and Security Using ASP.NET

Bojan Jovičić[1], Dejan Simić[1]

[1]FON – Faculty of Organizational Sciences, University of Belgrade
P. O. Box 52, Jove Ilića 154, 11000 Belgrade, Serbia and Montenegro
bojan.jovicic@gmail.com, dsimic@fon.bg.ac.yu

Abstract. Web applications security is one of the most daunting tasks today, because of security shift from lower levels of ISO OSI model to application level, and because of current situation in IT environment. ASP.NET offers powerful mechanisms to render these attacks futile, but it requires some knowledge of implementing Web application security. This paper focuses on attacks against Web applications, either to gain direct benefit by collecting private information or to disable target sites. It describes the two most common Web application attacks: SQL Injection and Cross Site Scripting, and is based on author's perennial experience in Web application security. It explains how to use ASP.NET to provide Web applications security. There are some principles of strong Web application security which make up the part of defense mechanisms presented: executing with least privileged account, securing sensitive data (connection string) and proper exception handling (where the new approach is presented using ASP.NET mechanisms for centralized exception logging and presentation). These principles help raise the bar that attacker has to cross and consequently contribute to better security.

1. Introduction

The security of information systems is a wide area. Its development followed that of information systems, whose development in turn followed advances in hardware. As computers and software have developed real fast: "To put it quite bluntly: as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming had become an equally gigantic problem." [1], so have developed the possibilities for security breaches.

Security and protection are very important areas of computers science and IT industry. One way to describe this area is: "Security can be defined as set of methods and techniques which control data accessed by executing applications. Even wide definition includes a set of methods, techniques and legal standards which control data access by applications and humans, and protect the physical integrity of whole computer system, no matter if it is distributed or not, or if it is centralized or decentralized." [2].

According to the American Computer Security Institute (CSI)'s 2005 Computer Crime and Security Survey, which enclosed big corporations, 56% of the subjects reported the detection of unauthorized use of their computer systems in last year. Also, according to the same survey, more than 95% of responding organizations experienced more than 10 Web site incidents. [3].

Today IT security has many sub areas focusing on different aspects of security, from lower levels of ISO OSI model, all the way to the application layer. Since security in lower levels has made significant improvements in past time, hackers try to get their way into system using the topmost layer. The application layer is specially exposed when used on the Internet in the form of the Web applications.

This paper will try to shed some light on making the Web applications more secure, since this area is mostly the responsibility of developer or is an effect of joint effort of developers and administrators.

2. Web application security

As digital enterprises embrace benefits of e-business, Web technology usage will continue to grow. Companies today use the Web as CRM (Customer Relationship Management) channel, as the means of improving supply chains, means of entrance in new markets, and for delivering products and services to customers and employees. However, successful implementation using Web technologies cannot be attained without consistent approach to the Web application security. In considering the Web application security, corporations often don't consider the following [4]:

- Today the hackers are one step ahead of enterprise (hackers use the newest „out of the box" applications, while companies have problems with setting base security measures)
- It is not a question of **IF** your site will be attacked, but **WHEN**.
- Passwords are not enough (don't use the passwords based on publicly available data about you, but use strong passwords)
- SSL and data encryption are not enough (SSL guarantees secrecy of traffic, but does not protect from Web application misuse)
- Firewalls are not enough (port 80 or port 443 are enough for an attacker)
- Standard scanning programs are not enough (these programs scan for standard mistakes, and can not evaluate security by checking contents of the Web application)
- A chain is as strong as it's weakest link
- It's in the code (most commonly the security neglect is in code, because applications are implemented by the unprofessional developers, or the developers that don't pay much attention to security, since primary concerns are often speed and implementation of the required functionalities)

- Manipulating a Web application is simple (plain Web browser and some determination are enough)

If it is in the code, then the coder needs to know how to code properly, or in security context, how to code defensively. A definition of defensive programming is given in [5]: Defensive programming doesn't mean being defensive about your programming – "It does so work!" The idea is based on defensive driving. In defensive driving, you adopt the mind-set that you're never sure what the other drivers are going to do. That way, you make sure that if they do something dangerous you won't be hurt. You take responsibility for protecting yourself even when it might be the other driver's fault. In the defensive programming, the main idea is to handle all possible errors including the errors in other cooperative routines or programs. More generally, it's the recognition that programs will have problems and modifications, and that a smart programmer will develop code accordingly.

3. Common Web application attack types

Bellow is the list of some of the most common Web attack types:

- SQL Injection (a security vulnerability that occurs in the database layer of an application)
- Cross-Site Scripting (causes a user's Web browser to execute a malicious script)
- Web site defacement (occurs when a hacker breaks into a web server and alters the hosted website or creates one of his own)
- Buffer Overflow (hackers exploit buffer overflows by appending executable instructions to the end of data and causing that code to be run after it has entered memory)
- DOS (Denial Of Service - an assault on a network that floods it with so many additional requests that regular traffic is either slowed or completely interrupted)
- Password Cracking (the process of recovering secret passwords from data that has been stored in or transmitted by a computer system, typically, by repeatedly verifying guesses for the password)

This list can probably be a lot longer, but this work concentrates on the two first listed items. These two Web attacks are listed in the top ten most critical web application security vulnerabilities [6]. According to the statistical analysis of results obtained from the numerous application level penetration tests performed by the Imperva experts for various customers over the years 2000 - 2003., SQL Injection makes 10% and Cross-Site Scripting makes 9% of the attack types [7].

Another common thing for these two types of attack is that both can be prevented by a developer. Some of the principles of good coding practice presented in this paper can be used to help prevent some other attack types listed here.

3.1. Sql Injection

The basic idea behind SQL Injection attack is abusing Web pages which allows users to enter text in form fields which are used for database queries. Hackers can enter a disguised SQL query, which changes the nature of the intended query. Hence the queries can be used to access the connected database and change or delete its data. Although protection from this kind of attack is very simple (especially using Microsoft.NET technologies), there is a big number of Internet systems which are vulnerable to this kind of attack.

SQL query generated by concatenation of the static part of the query and values intended for form fields is the base of this attack. For example, if there is a field for entering the username (tbUser) and a field for entering the password (tbPassword) and we perform authentication in the following manner:

```
string Query = "SELECT COUNT(*) FROM [User] WHERE
Username = '" + tbUser.Text + "' AND Password = '" +
tbPassword.Text + "'";
SqlCommand Command = new SqlCommand(Query, Connection);
if ((int) Command.ExecuteScalar() > 0)
    ...
```

It is enough for a malicious user to enter disguised SQL query in the username field, like:

```
' OR 1=1 --
```

This would turn the database query into:

```
SELECT COUNT(*) FROM [User] WHERE Username = '' Or 1 = 1
--' AND Password = ''
```

Since -- marks beginning of comment line in SQL (The comment operator „--” is supported by many relational database servers, including Microsoft SQL Server, IBM DB2, Oracle, PostgreSQL, and MySql [8]), this query is equivalent to

```
SELECT COUNT(*) FROM [User] WHERE Username = '' Or 1 =
1
```

Since 1=1 always evaluates to true, this query will always return more than 0 rows, if there are records in table, so that malicious user can easily authenticate with invalid credentials.

For this kind of attack, it is not necessary that parameters are passed by POST form method. If the query is formed based on the parameters passed in GET request, and is created using concatenation, the effect is the same.

Besides this kind of misuse, SQL Injection can be used for getting sensitive data from database, updating/deleting data, and other kind of malversations.

If the malicious user wants to gather extra data from database, he can use union SQL operator (UNION). This kind of information extracting is especially devastating if the user is familiar with database structure, because if he is not, he will have to gather extra data about database structure from system tables. Syntax for this kind of SQL Injection would be something like this:

```
' UNION SELECT ...,id,name,... FROM sysobjects WHERE
xtype = 'U' --
```

Here ... should be replaced with literals which are needed to match number and data type of columns. If a hacker discovers that there is a table Users (and often there is), next step could be retrieval of the usernames and passwords:

```
' UNION SELECT ..., Username, Password,... FROM [User]
--
```

Again the matching of number and data type of columns is required.

Updating is possible using SQL Injection like:

```
'; UPDATE Product SET UnitPrice = 0.01 --
```

This would set false prices to all products, although a smart hacker would lower the price to just one product, order it, and then return the original price. In the same fashion, the hackers can delete whole tables (using DROP command).

The hackers can run system stored procedures, as xp_cmdshell. xp_cmdshell is one of the most dangerous stored procedures, since it executes given command on the server (the machine, not DBMS). Perhaps equally powerful in this context is stored procedure sp_makewebtask. It puts query result in the given web page. This page can use UNC pathname as an output location. This means that the output file can be placed on any system connected to the Internet that has a publicly writable SMB share on it (The SMB request must generate no challenge for authentication at all) [9].

The degree of possible damage depends on the access rights defined for the account used to access database. It is a good rule to use an account with the least privileges. However, this rule is rarely used, since the most applications use the system administrator database account.

It is very important to understand that this kind of attack is not limited to SQL Server. The more powerful SQL dialect DBMS supports, the more vulnerable the database to this kind of attack is. SQL Injection attacks are not limited to ASP.NET applications. Classical ASP, Java, JSP, and PHP applications are at equal risk [10].

Protection from this kind of attack includes the following principles [10][11]:

"All input is evil"

Never use input for database queries that is not validated. Here ASP.NET helps with regular expression validation control: RegularExpressionValidator. To use this controls, it is enough to set its attributes for which control to validate, and which regular expression to use for validating. It is important to note that validation is performed on both client (using JavaScript) and server side, so no postback overload is generated during invalid postback attempts.

This kind of approach (validation restriction) is possible in two forms: by forbidding problematic characters, or by allowing a limited set of required characters. Although the approach with forbidding problematic characters (ie. apostrophe and dash) is easier for application, this approach is suboptimal for two reasons: first, it is possible to miss the character that is useful to the hackers, and second: often, there are multiple ways to represent problematic characters [11]. The hackers can use the escape characters to create apostrophe, and avoid validation. No matter which approach is applied, it is very useful to limit the length of input, since some kinds of this attack require very big number of characters.

For example, regular expression used for password and login fields with 4-12 characters length could be:

```
[\d_a-zA-Z]{4,12}
```

If you sometimes have to allow entering of apostrophe, use Replace method of String class:

```
string SanitizedUserInput = UserInput.Replace("'",
"''");
```

Avoid dynamic SQL

Never generate SQL queries by concatenating strings, but rather use SQL parameters. .NET provides classes for query parameters with automatic control of parameter values, and provides safe work of application, even without adhering to the first principle. These classes (in .NET there are more of these classes for different data providers: SqlClient, OleDB, Oracle, Odbc) perform control of passed values, so this kind of protection is an excellent defense against SQL Injection attack. Our corrected code for querying could look like this:

```
string Query = "SELECT COUNT(*) FROM [User] WHERE
UserName = @Username AND Password = @Password;

SqlCommand Command = new SqlCommand(Query, Connection);

Command.Parameters.Add(new SqlParameter("@Username",
tbUser.Text));

Command.Parameters.Add(new SqlParameter("@Password",
tbPassword.Text));
```

In this manner we are sure that .NET will perform all necessary checks and conversions to prevent evil SQL query. More control can be added by providing field length:

```
SqlParameter parUsername = new
SqlParameter("@Username", SqlDbType.VarChar, 20);

parUsername.Value = tbUser.Text;

Command.Parameters.Add(parUsername);
```

Next step should be replacing queries with stored procedures. Stored procedures provide additional safety effect, since they are used with parameters, and DB access account can be granted rights to use the stored procedures instead of the rights to access tables directly. In this way the extent of possible damage is greatly reduced.

Execute with the least privilege

Never use the database administrator account (which is usually a member of System Administrator role), but use the special account with defined access rights, or with membership in proper role with defined rights.

By declaring access rights on the stored procedures instead of tables, you are giving hackers plenty of problems, since they can’t access tables directly, but using the stored procedures.

Proper exception handling

Bad exception handling is one of the areas hackers will try to use, especially in SQL injection, because it provides them with feedback which greatly eases attack procedure. If Web application has proper exception handling mechanism, this kind of attack turns into blind SQL injection attack.

This mechanism should provide minimum of information that could help hackers, and should keep detailed info in different logs. There should be a balance between messages helping an ignorant user and giving too much information to hackers.

Setting Web applications for desired behavior in ASP.NET is very easy. It is enough to set *debug* attribute in Web.config file to false, *customErrors* attribute to *On* or *RemoteOnly*. Value *On* always shows defined error page, while *RemoteOnly* allows local machine users to get detailed information about errors, and remote ones get defined error page. You should never use *Off* value in production.

Besides that, ASP.NET provides two more extremely useful mechanisms in exception handling. One of them is the possibility to define a custom page to display errors. This page will replace the default ASP.NET error page (known as "yellow screen of death" in ASP.NET developer community). This page can contain a friendly message and a form where users can describe actions that brought to the error generation. This page is set using *defaultRedirect* attribute of element in Web.Config file.

Other equally powerful mechanism is application centralized exception handling of all unhandled exceptions. By implementing Application_OnError method you get the possibility to examine each unhandled exception. This method is implemented in Global.asax file, which contains all global application events.

In this paper proposed approach for best exception handling is combining these two mechanisms so you can log the exception and then show the custom error page. In this manner you get both the system error description, plus user description of actions that caused the exception.

Store secrets securely

It is very easy to use SQL injection to retrieve data from tables which hold usernames and passwords. This table often has its data in clear text. Much better approach is to keep data encrypted or hashed. In case of passwords, hash is better, since it can't be decrypted, and there is no need for securing the key used for encryption. Hash can be additionally reinforced by applying salt (cryptographically safe random value) to hash. Hashing in .NET is performed using GetNonZeroBytes method of RNGCryptoServiceProvider class for generating cryptographically safe random value. Then this value is turned into Base64 string using ToBase64String method of Convert class. After joining secret and salt, we apply some of hashing algorithms to this string (.NET contains implementations of following hashing algorithms: MD5, SHA1, SHA256, SHA384 and SHA512).

Although not directly connected with SQL Injection attack, connection string protection (being some of the most sensitive data) is very important, especially if it contains password for DB access. Since there is a need for the decrypted version of connection string, hashing is not an option. Usual place for keeping connection string are configuration files (Web.Config in ASP.NET applications). As with every encryption/decryption, the problem is safety of the key used in this process. One of solutions to this problem is the Win32® Data Protection API (DPAPI) for encrypting and decrypting data [13].

DPAPI is particularly useful in that it can eliminate the key management problem exposed to applications that use cryptography. While the encryption ensures the data is secure, you must take additional steps to ensure the security of the key. DPAPI uses the password of the user account associated with the code that calls the DPAPI functions in order to derive the encryption key. As a result the operating system (and not the application) manages the key. It also supports adding additional entropic value, for reinforcing the key.

3.2. Cross-Site Scripting

CSS (**C**ross-**S**ite **S**cripting) or XSS is possible with dynamic pages showing the input that is not properly validated. This allows the hacker to inject the malicious JavaScript code in a generated page and execute the script on a computer of any visitor of that Web site. Cross-site scripting could potentially impact any site that allows users to enter data. This vulnerability is commonly seen on:

- Search engines that echo the search keyword that was entered,
- Error messages that echo the string that contained the error,
- Forms that are filled out where values are later presented to the user, and
- Web message boards that allow users to post their own messages [14].

A hacker that successfully uses cross-site scripting can access sensitive data, steal or manipulate cookies, create HTTP requests which can be mistaken for those of a valid user, or execute malicious code on an end-user system.

A typical CSS attack entails that the unsuspecting user follows a luring link that embeds escaped script code. The fraudulent code is sent to a vulnerable page that trustfully outputs it. Here's an example of what can happen:

```
<a
href="http://www.vulnerableserver.com/error.aspx?err=
<script>document.location.replace(
'http://www.hacker.com/HackerPage.aspx?Cookie=' +
document.cookie); </script>">Click here to win Master
Studies Scholarship!</a>
```

The user clicks on an apparently safe link and ends on a vulnerable page with a piece of script code that first gets all the cookies on the user's machine and then sends them to a page on the hacker's Web site.

It is important to note that CSS is not a vendor-specific issue and doesn't necessarily exploit holes in the Internet Explorer. It affects every Web server and browser currently on the market. Even more important, note that there's no single patch to fix it. You can surely protect your pages from CSS, you do so by applying specific measures and sane coding practices [15].

You should be aware that `<img src>` and `<a href>` can also point to script code, not just a “classic” URL. For example, the following is a valid anchor:

```
<a href="javascript:alert(1);">Click here to win Master
Studies Scholarship!"</a>
```

Note that there is no script block here. Other notable fact is that user doesn’t actually need to click the link. The malicious code can be set in an OnMouseOver event. The JavaScript alert method is often used to test for CSS vulnerability.

Some Web pages echo text passed by URL parameters, i.e.

```
http://www.vulnerableserver.com/error.aspx?err=Nonexist
ing%20User
```

If this page echoes parameter value on the page, it is possible to inject the malicious code instead of Nonexisting%20User.

A hacker has to study the HTML code of the target page, and try to override page, to create more damage. The idea is to send data to an custom address where this data will be saved:

```
</form><form action="login.aspx" method="post"
onsubmit="XSSimage = new
Image;XSSimage.src='http://www.hacker.com/' +
documents.forms(1).tbUsername.value + ':' +
document.forms(1).tbPassword.value;">
```

This code should be injected in typical login page. The first part of attack code (`"</form>"`) closes the original form, and the next part defines start of a new form which encloses existing elements. The new form uses the same attributes for the method and action of form, but has an additional attribute *OnSubmit* added in form definition. This attribute sets instructions to be executed when the user clicks submit button, and before sending the real form request.

The first command creates a new image object. The second one specifies URL address of the image. Location will always start with an address of some site, that the hacker has access to. The second part of URL will be values of tbUsername and tbPassword fields, separated by colon. So, when the new image object is created, which will happen when the user submits this typical login form, a request will be secretly sent to address www.hacker.com. This request will contain credentials of the user trying to login. Right after this malicious request, real form submission will occur, so that user has no idea that anything unusual happened. The hacker just needs to retrieve data from his Web site.

Usually the hackers send enormous number of email messages, in which they invite users to click the malicious link. From this huge number, certainly a few users will click and become victims.

In order for CSS to be exploited, the user browser must allow some kind of scripting language. Vulnerability to this kind of attack can be mitigated by the proper filtering of user input. All alphanumerical data entered by user should be URL encoded, and then displayed to the user. In this way "<" (less then) would be converted to "`&lt;`". Here ASP.NET helps with HtmlEncode method of HttpServerUtility class. So the typical Cross-Site Scripting vulnerability test code would be encoded as:

```
&lt;script&gt;alert('XSS')&lt;/script&gt;
```

When this is presented in a page, it is totally harmless.

Besides this, ASP.NET performs automatic validation of the page to prevent scripting attacks. This validation is present in ASP.NET since version 1.1. Validation is performed on objects of the QueryString, Form and Cookies collections. The collection QueryString contains parameters passed through URL, and the Form collection contains form objects, while the Cookies collection contains cookies. This validation is active if the *ValidateRequest* attribute of page is set to true, or if this attribute for the whole application is set to true. By default, this is set to true for each page. This validation will reject any potentially dangerous code which is passed as a request in above mentioned collections, and will generate an exception.

From the end user aspect, the easiest solution to Cross-Site Scripting problem would be to turning off the support for all scripting languages. However this leads to loss of functionality of such a level, that some sites are rendered useless. Even this measure can’t help if a hacker injects code in URL, or enters it in existing form.

It is recommended for end users to take care how they are visiting new Web sites. They should never click unknown links in email messages from unknown senders. It is best to follow the links from main site, which has been used before.

There are some attempts to create personal Web firewall application that will mitigate cross-site scripting attacks. One of these is Noxes [16] which acts as a Web proxy and uses both manual and automatically generated rules to mitigate possible cross-site scripting attempts. Noxes effectively protects against information leakage from the user's environment while requiring minimal user interaction and customization effort.

3.3 Tools for identifying Web application security holes

If a developer is not paying attention to security during development, someone else (ie. IT professional or Systems Administrator) can use custom tools in order to check if application has security breaches and inform the developer about them. Some of the tools for Web application security analysis are:

- SWAP (Secure Web Applications Project) [17]
- WebSSARI [18]
- WebInspect [19]

4. Conclusion

It is up to the developer to use provided mechanisms, and create safe code. For the developers that aren't into security there are some tools that can help in Web application security analysis, but these tools only identify the problem, they don't eliminate it.

In order to eliminate application security problems the developers have to pay attention to security and have to code securely. In this paper we have shown that ASP.NET, and now ASP.NET 2.0, integrates a number of defense mechanisms that can be easily applied:

- Classes for SQL parameters that prevent SQL injection,
- Automatic checking for CSS attack, and
- Custom error pages and centralized exception handling.

These mechanisms combined with other defense techniques and security strategies create powerful toolset to secure application functionalities in the Internet environment with minimal effort from the developer.

5. References

1. Dijkstra, E.: The Humble Programmer. ACM Turing Lecture (1972)
2. Simic, D.: Materials from lectures for post graduate studies of "Security techniques in computer networks". Faculty of Organizational Sciences, Belgrade (2005)
3. Lawrence, G. et al.: CSI/FBI Computer Crime and Security Survey. Computer Security Institute (2005). [Online]. Available: http://www.usdoj.gov/criminal/cybercrime/FBI2005.pdf (current 2005)

4. Sima, C.: Security at the Next Level. SPI Dynamics (2004). [Online]. Available: http://www.spidynamics.com/whitepapers/webappwhitepaper.pdf (current 2004)
5. McConnell, S.: Code Complete, 2nd edition. Microsoft Press, 187-213. (2004)
6. The Open Web Application Security Project: The Ten Most Critical Web Application Security Vulnerabilities. The Open Web Application Security Project (2004). [Online]. Available: http://www.owasp.org/documentation/topten.html (current 2004)
7. Surf, M., Amichai, S.:How safe is it out there?. IMPERVA (2004). [Online]. Available: http://www.imperva.com/application_defense_center/papers/how_safe_is_it.html (current 2004)
8. Howard, M., LeBlanc, D.: Writing Secure Code, 2nd edition. Microsoft Press (2003)
9. Spett, K.: SQL Injection. SPI Dynamics (2002). [Online]. Available: http://www.spidynamics.com/papers/SQLInjectionWhitePaper.pdf (current 2002)
10. Litwin, P.: Data Security: Stop SQL Injection Attacks Before They Stop You. MSDN Magazine (September, 2004)
11. Advosys Consulting Inc.: Writing Secure Web Applications. Advosys Consulting Inc (March, 2004)
12. World Wide Web Consortium: Character entity references in HTML 4. World Wide Web Consortium (2006). [Online]. Available: http://www.w3.org/TR/REC-html40/sgml/entities.html (current 2006)
13. Meier , J.D. et al.: Building Secure ASP.NET Applications: Authentication, Authorization, and Secure Communication. Microsoft Patterns and Practices. Microsoft Corporation, 354-362. (2002)
14. Spett, K.: Cross-Site Scripting. SPI Dynamics (2002). [Online]. Available: http://www.spidynamics.com/whitepapers/SPIcross-sitescripting.pdf (current 2002)
15. Esposito, D.: Take Advantage of ASP.NET Built-in Features to Fend Off Web Attacks. Wintellect (2005). [Online]. Available: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnaspp/html/securitybarriers.asp (current 2005)
16. Kirda, E. et al.: Noxes: A Client-Side Solution for Mitigating Cross-Site Scripting Attacks. ACM (2006)
17. Scott, D., Sharp, R.: Developing Secure Web Applications. IEEE Computer (November/December 2002)
18. Huang, Y.W. et al.: Securing Web Application Code by Static Analysis and Runtime Protection. ACM (2004)
19. SPI Dynamics: WebInspect™ - Your Web Security Partner (2006). [Online]. Available: http://www.spidynamics.com/products/webinspect/ (current 2006)

Bojan Jovicic is a MSc student at FON – Faculty of Organizational Sciences, University of Belgrade. He is working at DELTA SPORT as a Sr. Software Developer, mostly in .NET and MS Dynamics AX.

Bojan Jovicic holds Microsoft Certified Professional, Microsoft Certified Application Developer and Microsoft Certified Solution Developer title in .NET. He also holds the title of Microsoft Business Solutions Certified Professional in Axapta 3.0 Programming. Bojan has created over 30 web applications, some of them reaching over 6000 visits per day.

His interests are Web application development and security, MS Dynamics AX development and business process management.

Dejan Simic, PhD, is a professor at the Faculty of Organizational Sciences, University of Belgrade. He received the B.S. in electrical engineering and the M.S. and the Ph.D. degrees in Computer Science from the University of Belgrade. His main research interests include: security of computer systems, organization and architecture of computer systems and applied information technologies.