VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Maysan Abdullah

SPM System Cybersecurity

Information Technology
2016

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Degree Programme in Information Technology

ABSTRACT

Author	Maysan Abdullah
Title	SPM System Cybersecurity
Year	2016
Language	English
Pages	18
Name of Supervisor	Smail Menani

This document discusses the SPM (Student Project Management) system security measures and best practices.

The system will be integrated with VAMK's own database and network, but it should also allow for external participants. Therefore, implementing an authentication mechanism for the system is required, in addition to managing the users' information, passwords, and other sensitive data.

Three aspects are discussed which are: authentication, password hashing, and data encryption.

A secure authentication mechanism is provided, as well as discussing secure password hashing techniques. The best hashing algorithm for the purposes of the system is decided to be the BCrypt algorithm.

Multiple encryption algorithms are discussed and compared. The comparison resulted in the decision to use the Blowfish algorithm as the encryption algorithm for the SPM system.

The chosen procedures and algorithms follow the recommendations of security experts, and are determined to be highly adequate for guaranteeing the safety and security of information in the SPM system.

Keywords SPM, Security, Hashing, Encryption

CONTENTS

LIST OF FIGURES AND TABLES

1 INTRODUCTION

The SPM (Student Project Management) system being developed for the Industrial Innovation Academy is used for teachers to enter projects, assign tasks, and monitor the participants progress and for students to specify what they have accomplished, and enter any required data.

The System will be integrated with VAMK own database, and use its own authentication system, but it should also allow for participants from outside VAMK to access the system. Therefore, a membership database is needed, and an authentication system should be implemented.

Furthermore, it is important to protect user passwords, and any other sensitive information from being exposed to attackers.

In this document we will discuss three aspects of security and the best practices and techniques to follow in order to achieve the best possible security for the system.

The discussed aspects are:

1.1 Authentication

The system authentication mechanism will be discussed. The .NET Web API authentication technique will be used utilizing the OAuth2 protocol.

1.2 Password Hashing

Password protection is important, and the way passwords are secured is by using a cryptographic hashing function on the passwords entered by users, so it cannot be reversed.

Two popular and secure algorithms (PBKDF2 and BCrypt) are discussed and compared, and the best and most secure algorithm will be chosen for the system.

1.3 Encryption

The database might contain important data that should be protected even if the database has been exposed to an attacker. Therefore, an encryption algorithm will be used on these data to ensure their secrecy.

Five common encryption algorithms will be discussed (DES, 3DES, AES, RSA, and Blowfish). For each one, the characteristics of the algorithm will be listed with its advantages/disadvantages, and as a result of the comparison the best algorithm for our purposes will be chosen for the SPM system.

2 AUTHENTICATION

The SPM system user base will be mostly students, and teachers in VAMK, and as such authenticating them can be done using LDAP, and VAMK own database. However, the system should allow for external users as well, therefore their data should be entered in the system own database, and an authentication system should be implemented for this purpose.

It's important to ensure that only authenticated users have access to the system, and that resource access should be determined based on the particular roles of the users.

The authentication process will be implemented using .NET web API utilizing the OAuth2 protocol.

2.1 Credential Flow

The authentication flow happens as follows:

- Users enter their credentials (username, password) through client interface.
- The credentials are sent to the "Authorization Server".
- If the authentication is successful, the server will return an access token to the client.
- When a request for a resource is sent, the client will include this token in the authorization header of the http request.

This flow is demonstrated in figure 1.

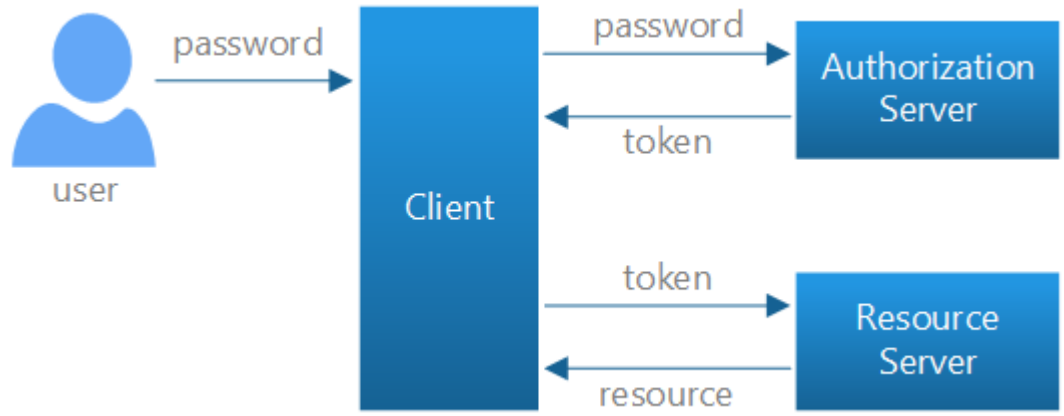

Figure 1. Authentication credential flow.

In Web API, the previous flow is implemented as demonstrated by the following figure.

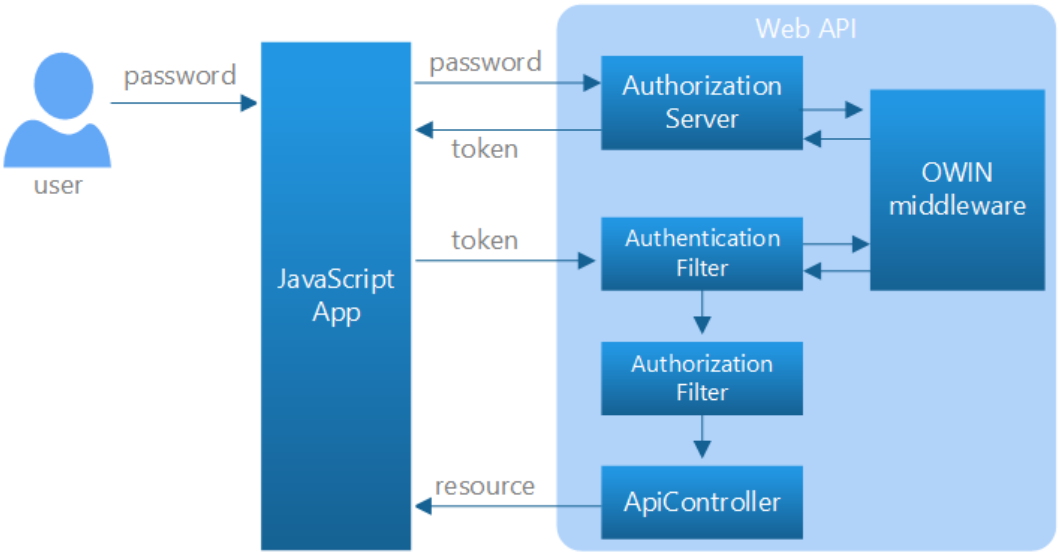

Figure 2. Web API authentication flow.

Users will enter their credentials in the JavaScript client, which will send them to the authorization server. The credentials will be validated, and in the case of success an access token will be sent back. When the user tries to call certain function, the client will send this token in the Authorization header of the request sent to the

Authentication filter, which will check the validity of the token. If the token is valid, the request will be delegated to the Authorization filter that will check if the call is designated with the “Authorize” attribute and make sure the user has permission to access that particular functionality. If the authorization succeeds, the appropriate API controller is called to respond to the request, otherwise an Unauthorized error (401) is sent.

The OWIN middleware component forms a link between the authorization server, and authentication filter on one hand, and the OAuth2 protocol implementation on the other.

3 PASSWORD HASHING

User passwords need to be safely stored in the database in a way that even if an attacker was able to access the database they won't be able to know the passwords, or at least it will take a long time giving users a chance to change them.

To accomplish this, a hashing function is applied to the passwords entered by users which will convert each one into a string of fixed-length, that cannot be converted back to the original password.

When a user tries to login to the system, the hashing function will be applied to the entered password, and the result will be compared with the value stored in the database and if they match, the login process will succeed. This means that the actual password is never stored anywhere.

Algorithms used for hashing passwords need to be specifically designed to be secure and are known as cryptographic hash functions as opposed to faster but unsecure hashing functions used in programming data structures.

Cryptographic hashing algorithms include: SHA256, SHA512, RipeMD, BCrypt, WHIRLPOOL, and others.

3.1 Password Hash Attacks

There many types of attacks that can be tried on a password hash in order to crack it. Here are the most common ones.

3.1.1 Brute Force Attack

The simplest type of attack. Every possible combination is hashed and compared with the available hash, and if they match the original password is discovered.

This attack cannot be prevented, but it's not efficient, and it takes a long time.

3.1.2 Dictionary Attack

This attack is similar to the previous one, except that instead of trying all possible combinations, a file with the most commonly used passwords is used as a source with possibly some operations applied to each word to get different variations.

This attack like the brute force one is not very efficient, but countering it requires that users choose difficult and not common passwords.

3.1.3 Lookup Table Attack

Lookup tables are obtained by pre-compiling a list of hashes from some dictionary of passwords along with the passwords themselves in a data structure where getting the corresponding password from its hash is very fast.

This attack is very efficient, and if the passwords chosen weren't following security guidelines (dictionary words, commonly used passwords), the hash can be cracked almost instantaneously.

3.1.4 Rainbow Table Attack

Rainbow tables are similar to lookup tables but they are designed to be as space efficient as possible instead of being fast. This allows them to store a very large number of passwords making them effective for password hash cracking.

3.2 Countering Attacks

In order to prevent these attacks or at least make them inefficient, the following procedures will be applied.

3.2.1 Adding Salt

Lookup and rainbow table attacks rely on password hashes being the same for the same password. This can be countered by adding a random factor that varies the hashes even for the same password. This can be accomplished by adding a random string known as "salt" to the password before hashing it. This way even if two users choose the same password, the has for each will be different. The salt

needs to be stored in the database as well, so that when authenticating the stored salt will be added to the entered password then hashed and compared to the value stored in the database.

This will ensure, that lookup and rainbow tables cannot be pre-compiled in advance since the attacker won't have knowledge of the salt beforehand.

In order for the salt to be secure, it has to be generated every time a password is chosen and not reused again. It should also be long enough, otherwise cracking it could be done faster.

It is also important to generate the salt using a Cryptographically Secure Pseudo-Random Number Generator (CSPRNG). These generators, unlike the normal random number generators used in programming languages are designed to be as random and unpredictable as possible to ensure that cracking the salt is more difficult.

In .NET there is a class implementation for this purpose which is the class: System.Security.Cryptography.RNGCryptoServiceProvider.

3.2.2 Using Secure Hash Algorithm

As mentioned before, brute force and dictionary attacks cannot be prevented. The only counter measure is to ensure that the cracking process is as slow and inefficient as possible giving the administration and users enough time to change passwords and protect data.

To accomplish this, a special type of hashing algorithms designed to be slow and computing intensive should be used. This way even if the attacker is using high-end devices, the process will still take a very long time.

These algorithms take an argument that specifies how secure the hash should be, which will determine how slow the process will be. The argument should be chosen to provide the slowest time while still being tolerable by the users.

3.3 Choosing Hashing Algorithm

There are two algorithms that are usually recommended for password hashing: PBKDF2, and BCrypt. In the following, these two algorithms advantages and disadvantages will be compared and the best algorithm for our purposes will be chosen.

3.3.1 PBKDF2

PBKDF2 is an easy function that performs HMAC (Hash based Message Authentication Code) as many times as specified by a parameter. HMAC is performed by applying a hash function in combination with a secret key.

This algorithm is designed to be computation intensive, so it's very taxing on the CPU making any cracking attempt very slow as long as the attacker is relying only on CPUs. However, this algorithm can be implemented efficiently on GPUs, so if the attacker has a powerful GPU, cracking hashes produced by this algorithm can become much faster.

NIST (National Institute of Standards and Technology) actually recommends this algorithm for password hashing, so it's safe enough to use as long as a high enough iteration count is used.

3.3.2 BCrypt

BCrypt is based on the Blowfish block cipher, and it's designed to be difficult to perform on GPUs. This is accomplished by relying on access to table that is constantly changing in memory, which is hard to do in GPUs, and it's easier for CPUs. Therefore, BCrypt is both computation and memory intensive making it harder to crack than PBKDF2 even if the attacker uses high end GPU.

The problem however arises if attackers use FPGA devices instead which can implement this algorithm much better than GPUs making the cracking process faster.

3.3.3 Conclusion

Both of these algorithms are good enough for our purposes, and are generally recommended to use. BCrypt however has some advantage in that it's both taxing on the CPU and RAM. It was developed in 1999, so it has been tested for a long time, and so far it has not been broken. It's also widely used and supported in the security community, and has good implementations available for it.

Therefore, the bcrypt algorithm is chosen as the password hashing algorithm for the SPM system.

4 ENCRYPTION

There are pieces of information that are important to protect and need to be kept secret even if an attacker has access to the database or was able to intercept communication. Therefore, we need to apply an encryption algorithm to these data to guarantee security.

In the following the most common algorithms will be discussed and compared, then the best algorithms for our purposes will be chosen.

4.1 DES

DES (Data Encryption Standard) is a symmetric key encryption method for data which is outdated by now, but it's used as the basis for more advanced algorithms.

DES is a block cipher algorithm, so the cryptographic key and algorithm are applied to a block of data simultaneously rather than one bit at a time. The message is divided into 64-bit blocks and then encrypted. The process involves 16 rounds. The key length determines how difficult brute force attacks are. In DES the key length is 64 bits but only 56 are actually usable and the rest are used for parity.

4.2 Triple DES

Designed as a replacement for DES, which was dominant for a while, but is also outdated now. It uses 3 keys 56-bits each for a total of 168 bits.

Despite being outdated it is still used in some hardware encryption solutions in financial services and other industries.

4.3 AES

AES (Advanced Encryption Standard) algorithm is used by the U.S. Government and other organizations as the standard encryption algorithm.

It is efficient in 128-bit form, but it also uses keys of 192 and 256 bits for even safer encryption.

AES is considered impervious to most attacks, except brute force, which attempts all possible combinations in the 128, 192, or 256-bit cipher in addition to key recovery and side channel attacks. But it is still considered a very safe encryption algorithm.

4.4 RSA

RSA is a public-key encryption algorithm and the standard for encrypting data over the internet.

RSA is an asymmetric algorithm due to its use of a pair of keys. There is a public key, which is used to encrypt the message, and a private key to decrypt it. The result of RSA encryption is a huge batch of data that takes attackers a long time and processing power to break.

4.5 Blowfish

Blowfish was also designed to replace DES. It is a symmetric cipher that splits messages into blocks of 64 bits and encrypts them individually. It uses a variable key length ranging from 32 to 448 bits, and uses 16 rounds.

Blowfish is known for both its tremendous speed and overall effectiveness and it has no know vulnerabilities.

4.6 Comparison

The following table lists the previous algorithms with various attributes comparing them to each other.

Table 1. Encryption algorithms comparison.

PARAMETERS	DES	3DES	AES	RSA	BLOWFISH
DEVELOPMENT	In early 1970 by IBM and Published in 1977.	IBM in 1978.	Vincent Rijmen, Joan Daeman in 2001	Ron Rivest, Shamir & Leonard Adleman in 1978	Bruce Schneier in 1993
KEY LENGTH (Bits)	64 (56 usable)	168,112	128,192, 256	Key length depends on no. of bits in the module	Variable key length i.e. 32 – 448
ROUNDS	16	48	10,12,14	1	16
BLOCK SIZE (Bits)	64	64	18	Variable block size	64
ATTACKS FOUND	Exclusive Key search, Linear cryptanalysis, Differential analysis	Related Key attack	Key recovery attack, Side channel attack	Brute force attack, timing attack	No attack is found to be successful against blowfish.
LEVEL OF SECURITY	Adequate security	Adequate security	Excellent security	Good level of security	Highly secure
ENCRYPTION SPEED	Very slow	Very slow	Faster	Average	Very fast

4.7 Conclusion

From the previous table, we can see that the algorithm that provides the highest level of security, with good speed and performance is the Blowfish encryption algorithm, and as such this algorithm will be used in the SPM system.

5 FINAL CONCLUSION

In this document, we have discussed the topics of authentication, password hashing and data encryption.

.NET Web API authentication mechanism has been used utilizing the OAuth2 protocol, where we get an access token after the login process, and use it for authorization purposes.

We have talked about the importance of password hashing and using strong cryptographic hashing functions in order to make cracking the passwords as difficult and time consuming as possible.

We discussed two hashing algorithms (PBKDF2 and BCrypt) and we arrived at the conclusion that BCrypt has some advantages over PBKDF2, and decided to use it the hashing algorithm in the SPM system.

In the future more recent algorithms such as SCrypt can be studied and analysed to determine if it is a better alternative to BCrypt.

Finally, we studied and compared five common encryption algorithms (DES, 3DES, AES, RSA, and Blowfish).

After comparing the attributes of each algorithm, it has been determined that Blowfish is the most secure encryption algorithm with good performance, and decided to adopt it as the encryption algorithm for the SPM system.

We think that the chosen mechanisms and algorithms provide a very strong security for the system, and satisfy the security goals required for the SPM system.

REFERENCES

Wasson, M. Secure a Web API with Individual Accounts and Local Login in ASP.NET Web API 2.2. http://www.asp.net/web-api/overview/security/individual-accounts-in-web-api

Defuse Security. Salted Password Hashing - Doing it Right. https://crackstation.net/hashing-security.htm

Preziuso, M. Password Hashing: PBKDF2, Scrypt, Bcrypt. https://medium.com/@mpreziuso/password-hashing-pbkdf2-scrypt-bcrypt-1ef4bb9c19b3#.3o8pk5be6

Bradford, C. 5 Common Encryption Algorithms and the Unbreakables of the Future. http://www.storagecraft.com/blog/5-common-encryption-algorithms

Bhanot, R. Hans, R. A Review and Comparative Analysis of Various Encryption Algorithms. http://www.sersc.org/journals/IJSIA/vol9_no4_2015/27.pdf