A COMPARISON OF PASSWORD PROTECTION METHODS FOR WEB-BASED PLATFORMS IMPLEMENTED WITH PHP AND MYSQL

Nedyalko Katrandzhiev, Daniel Hristozov, Borislav Milenkov

University of Food Technologies, Plovdiv, Computer Systems and Technologies
e-mails: ned.katrandzhiev@gmail.com, d_hristozov@uft-plovdiv.bg,
bmilenkov67@gmail.com
Bulgaria

Abstract: The paper explains the need for using up-to-date security methods in order to protect some of the most sensitive information in every system: user passwords. MD5, SHA1, bcrypt and Argon2 have been compared as password hashing algorithms. The recommendations in this paper will help developers create systems with up-to-date security and avoid poor hashing algorithms.

Key words: web security, PHP, MySQL, hashing algorithms, database.

1. INTRODUCTION

The great opportunities offered by contemporary digital environments and network services are directly related to some important challenges to users' privacy and security. Cyber security and personal data protection are important subjects discussed at different levels, including the European Commission, the US Department of Homeland Security, corporative boards, etc. [1]. Recently, there has been a major user data breach almost every year, and millions of user passwords and data have been compromised [2]. Not even companies as big as Yahoo or Sony are insured against security breaches. They use highly customised systems according to their needs; however, protection methods and standards which conform to the needs of small to medium systems should not be underestimated either. The ethical obligation to minimize harm includes planning adequate protections aimed at ensuring personal information confidentiality on web-based platforms. Planning involves applying physical, technical and administrative measures to guarantee access and data use by authorized persons or information owners only. Public expectations regarding the protection of the personal data and dignity of information providers should be met when such a web-based platform is created. A good security level is important for legal, ethical and operational considerations and should be

taken into account. Security is easier to support on a dedicated server, but this also requires a backup strategy as well as a system for detecting and blocking external hacker attacks that should be installed on the server itself. Secure storage of user passwords is one of the most important responsibilities of every system. In the event of data breaches, it should be highly impossible for passwords to be converted back to the original string. The aim of this paper is to present an overview of the findings in current research, with a focus on the most common ways of protecting user passwords on web-based platforms built with PHP and MySQL.

2. METHODOLOGY

A server was configured, and Apache with PHP and MySQL were installed for the needs of the comparison.

2.1. PHP programming language

PHP is a programming language with syntax very close to C/C++. It has a relatively small learning curve and is very intuitive. In the transfer from PHP 5 to PHP 7, a large part of the code is rewritten and in some operations there is almost twice as fast execution of time for the same operation [3]. PHP can work on all major OS currently used, i.e. Linux, Windows and Mac. There is no need to change the code when moving from one OS to another.

A PHP flaw is the method of finding errors before running the code, something that other major languages have. Errors are shown when the code is executed. The error message displays the file name, row number and a small message describing the error. There are several error reporting levels, and they can be configured on the server or in the code. All error messages must be enabled when a code is written and an application is tested, so all of them can be fixed before finishing the product. In production, all error messages must be disabled and written to a log file viewed by the system administrator.

Errors are divided into three types:

- syntactic, e.g. missing ";" at the end of the line;
- operative, e.g. trying to connect to a database when the server is offline;
- logical, e.g. sending a wrong variable type to function.

More than 83% of all web pages use PHP and over half of them use CMS (Content Management System) [4, 5]. Many of them are open source, which allows detailed understanding of the way they work. The positive effects of this involve very fast removal of a bug in the code and making add-ons through which their creator can gain access to system resources or the database.

2.2. MYSQL databases

From PHP version 5, a new driver connecting to databases called MySQLi is

implemented. It is developed to support the new features in MySQL version 4.1.3. The new features and functionalities include object-oriented interface, support for prepared statements, support for multiple statements, and support for transactions.

SQL injections are one of the most popular ways in which hackers gain access to the website database [6]. In the most common case, the query is interrupted and a new query is added after it, with a request for all the usernames and passwords. There are some simple rules with which the thread from the SQL injection is eliminated.

The **mysqli_real_escape_string()** function filters some special characters that can break the query and eliminate most of the vulnerabilities from the SQL injection. There are several places for collecting, displaying and processing data in PHP and MySQL, so one character encoding for all the information must be used. It should be set in the user browser when it sees or sends information to the system, and to PHP and MySQL where this information is stored and processed. It is very important for every part of the system, i.e. user, server and database, to use the same character encoding and to avoid broken records in the database due to wrong encoding. The function **mysqli_set_charset()** is used for setting the charset used by the PHP to communicate with MySQL.

Another recommendation is that prepared statements be used. There are two main benefits of using them. One is that the volume of sent data is reduced, and the other that faster execution time is gained at the same transaction speed. In the first query, only the structure of the query is sent, without the data. After sending it, MySQL checks if the code is valid, if the tables in the query exist, etc. For example, if we need to add 10,000 rows to our database with a loop sending one query at a time, this will result in 10,000 validations and checks. If we use prepared statements, we will send and validate the query once, and in the loop only the data will be sent. In this way only the data will be checked and no additional validation for the whole query will be made, hence a lower sent data amount and faster execution time will be achieved.

There is a clear distinction between MySQL and data in the whole query when prepared statements are used. Thus, there is no possibility for a query to be terminated and for someone to write another query and have access to information which is not public, such as a list of usernames and passwords. If the information that is sent contains wrong symbols or something else does not conform to the requirements, MySQL will return an error and will not execute the query. As a result, no information will be displayed to the end user and the administrator will be able to catch the error, analyze it and prevent future failures.

2.3. Methods of password hashing in PHP

A survey has been conducted on how long an average user password is (Figure 1). Most users are unwilling to remember difficult passwords with lower and

uppercase letters, digits and special symbols for each website they use. Thus, a need arises for using an evolving protection method [7]. Passwords are stored by hashing algorithms because of the speed they provide in comparison with the cryptographic algorithms.

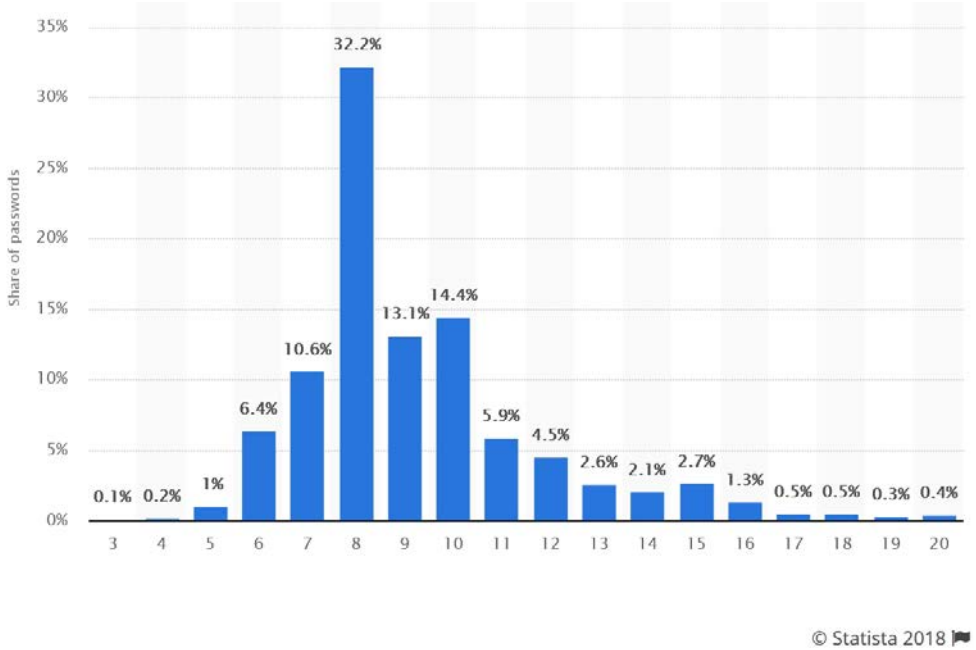

Fig. 1. Distribution of password length in 2017 in percentage [8]

MD5 and SHA1 do not have the option of making the algorithm more complicated. In these algorithms, the same input always gives exactly the same output. Rainbow tables are widely available and easy to use. Part of the hash of all created passwords, e.g. all combinations with small letters and digits from 1 to 10, are written in those tables according to some rules. The output is a very big file, and when using a middle level GPU it becomes clear within minutes whether a given password is in the table and what it is. If a brute-force method is used, it will take more time. The main problem of those algorithms is the growth in computing power and the reduced time for finding the passwords.

Bcrypt is implemented in PHP version 5.5.0. Its use has two important components: the output is always different for the same inputs, and the algorithm complexity can be defined. The complexity (cost) can be from 4 to 31 [9, 10]. Default cost currently is 10. It is possible to set different cost parameters for different user types, for example:

- users can have cost 10;
- moderators can use 12;

- administrators can use 14.

The most important feature of bcrypt is that when computational powers evolve, there is no need to change the algorithm every 2 or 3 years; defining a bigger cost parameter would suffice. Furthermore, users would not be forced to change their passwords when that happens. PHP offers very simple functions for checking and upgrading user passwords if the default cost parameter is changed. There is no need to force the user to perform any actions or send e-mails, and user passwords do not have to be stored in the plain text somewhere.

Bcrypt passwords cannot be attacked with rainbow tables. They can be attacked by brute force, but the time required to hash is doubled by every cost increased. Even with a dedicated machine with 8 GPU (GeForce GTX TITAN X), the speed for finding passwords is about 4000 hashes per second (H/s) for cost 10 [11, 12], compared to 150 GH/s for MD5, and 68 GH/s for SHA1.

A new algorithm, Argon2, has been introduced in version 7.2 of PHP [13]. It was the winner of the Password Hashing Competition in July 2015. The main aspect of its creation lay in the rapidly increasing computational powers of GPU and ASIC devices. Their main weakness is that these devices have very limited memory. Increasing memory requirements will make the devices obsolete because most of the processing power will be lost and a very small percentage of it will be used. Most ASICs are built according to very specific needs (just the necessary RAM to be used by the algorithm, nothing more) so that they can be sold at affordable prices. The possibility to change the used memory in this algorithm is a main feature aimed at preventing the insanely fast speed of generated combinations per second of those devices. This algorithm has more than one parameter to configure (compared to bcrypt with cost only), so it is not that easy to recommend settings. Trial and error is the best way of making the most of the server and complying with the requirements for password security. It can be used in shared hosting with a limited amount of available memory with default settings. The Argon2 authors recommend the following steps when determining parameters [13]:

1. Find the maximum amount of threads the system can use for hashing.
2. Find the maximum amount of memory for each call for hashing.
3. Determine reasonable execution time and choose the number of passes accordingly.

The reasonable execution time must be chosen according to the system load and the server configuration, and it can range between 0.05 and 0.1 sec. in a system with a lot of active users, or a few seconds in a closed user low load system.

2.4. Hardware Used

In our test, we used a custom dedicated server for the purposes of the experimental tasks and time comparisons. The server specifications were:

CPU: 1x Intel® Xeon® Processor E5-2620 v4 @2.1 GHz, 8 core / 16 threads, 20 MB Cache;

RAM: 64 GB (4x16GB) DDR4@2133MHz, ECC, Registered;

MB: Asrock EP2C612D16C-4L, Dual Socket LGA 2011 R3, 16x DIMM DDR4, 10x SATA3, 3x PCIe3.0 x16, M.2 max type 22110 support;

SSD: NVMe M.2 Samsung 960 evo 500gb, read/write speed: 3200 /1800 MB/s.

3. RESULTS AND DISCUSSION

PHP has some functions that can prevent a lot of dangerous things happening to our code and data. **mysqli_real_escape_string()** function must be used for the protection of SQL queries. If the received data are not filtered, they may contain a wrong data type or even a malicious code towards the SQL server.

For character encoding, we recommend UFT-8. It uses a variable width and supports many languages and some mathematical symbols.

In our test, the necessary time in which the server created 1 000 000 MD5 passwords was 0.33 sec., and for 1 000 000 SHA1 it was 0.44 sec. At that speed, finding a password below 8 characters would take several hours. If finding a password takes 1 sec. using MD5 in bcrypt, the same password with cost 10 would take about 14 months to be found using the same parameters. SHA1 is about 2 times as slow as MD5, so for each second spent on finding that password, in bcrypt that would be about 7 months for each.

The necessary time for hashing one password in bcrypt with default cost 10 is 0.09 sec.

In Table 1 we have sampled times for generating one password on our server using different cost parameters.

The syntax is:

password_hash($password, PASSWORD_BCRYPT, ["cost" => $cost]);

where: **$password** is the user password to be hashed,
$cost is the chosen cost parameter.

The way passwords are verified is with:

password_verify($password, $hash);

where: **$password** is the password typed by the user,
$hash is the hashed password from the database.

The **password_verify()** function returns TRUE if the hash and the password match, or FALSE if they do not.

Table 1. Sample execution time in seconds for hashing one password using bcrypt

cost	9	10	11	12	13	14	15	16	17	18	19
Time (sec.)	0.046	0.091	0.162	0.352	0.659	1.334	2.697	5.429	10.335	21.418	42.725

It is evident that when the cost is increased, the computation time is doubled.

Bcrypt has some important limitations. The maximum character length is 72 bytes. If our password is longer than that, other characters will be truncated. This means that 2 passwords 80 bytes long with the same 72 starting bytes and different last 8 bytes, after **password_verify()** function, will result in TRUE.

Applying UTF-8 and depending on what alphabet or symbols are used, the length of what can be typed varies. If the password uses a US-ASCI table, UTF-8 uses one byte per character, and we can type a 72-character password before reaching the distinguishable limit. If Cyrillic, Greek, Arabic or other popular alphabets are used, one character is two bytes, which makes a maximum of 36 characters. Three bytes are used in almost all other languages that are in Basic Multilingual Plane, and some popular symbols like ₤ and €. Here, there is a maximum of 24 characters. With four bytes, the maximum of the UTF-8 standard, the rest of Multilingual Plane and some special mathematical symbols are coded. Here the maximum is 18 characters. This information must be taken into consideration when the hashing method is used. It is highly unlikely for users to use more than 18-character long passwords (less than 1%), but if for some reason much more character space is required, Argon2 can be used. It has $(2^{32} - 1)$ byte as maximum, which is more than enough for all security requirements in length and is the equivalent of LONG TEXT in MySQL.

An example of a maximum login request is shown in Table 2 using bcrypt with different costs on Apache for 1000 requests at concurrency level 10.

Table 2. Apache benchmark:1000 requests with concurrency level 10

Cost	9	10	11	12	13	14	15
Time taken (sec.)	5.862	11.313	21.519	44.436	87.476	178.212	358
Req. per second	170.59	88.39	46.47	22.50	11.43	5.61	2.79
Time per req. (ms)	58.621	113.129	215.193	444.364	874.761	1782.12	3580.487

There are some sampled times for hashing a password with different parameters: threads (T), memory cost (M) in KB, and time cost (P) presented in

Table 3. The necessary time (t) for hashing one password in Argon2 with default parameters (2 threads, 2 passes and 1 MB memory) is 0.05 sec.

The syntax is:

password_hash($password, PASSWORD_ARGON2I, ['memory_cost' => 131072, 'time_cost' => 2, 'threads' => 2]);

where:

$password is the user password to be hashed, and all three parameters are given as array. The way passwords are verified is with:

password_verify($password, $hash);

where: **$password** is the password typed by the user;
$hash is the hashed password from the database.

The **password_verify()** function returns TRUE if the hash and the password match, or FALSE if they do not.

Table 3. Sample execution time in seconds for hashing one password using Argon2

T (threads)	M (memory)	P (passes)	t (time)
2	**1 MB**	**2**	**0.050**
2	**32 MB**	**2**	**0.234**
2	***128 MB***	***2***	***0.776***
4	**128 MB**	**2**	**0.542**
4	**32 MB**	**2**	**0.175**
8	**32 MB**	**2**	**0.170**
8	**32 MB**	**4**	**0.776**
8	**128 MB**	**2**	**0.478**
8	**128 MB**	**8**	**1.570**
8	**1024 MB**	**2**	**3.220**

With Argon2 there is no simple guide, and it all depends on the server capabilities and total number of users.

4. CONCLUSION

Using MD5 or SHA1 for current and up-to-date password security is not recommended because of the speed in which they are created and the inability to evolve. Argon2 is highly recommended for use for a new project starting in version 7.2+ with a high need for security such as hashing and encryption. Although Argon2 looks very promising on paper, it must be tested for time in order to prove its worth as a winner and a modern type of algorithm. At present, the majority of operating projects are written in lower PHP versions, and we recommend using bcrypt for

those running on version 5.5 or higher. There is an additional library that systems running on PHP version 5.3.7 and newer can add to use bcrypt but some knowledge would be needed to implement it [14]. Migrating from lower versions to PHP can be difficult and may require most of the code to be rewritten.

With the current hardware, passwords with a cost parameter bigger than 16 can be used only in small closed systems with several users and very high security requirements. For small systems on a shared hosting plan or small VPS with many active users, it is not recommended to use a cost bigger than 10-11.

REFERENCES

[1] Romansky R., Survey on digital world opportunities and challenges for user's privacy, *International Journal on Information Technologies & Security*, No.4, Vol. 9, 2017.

[2] CSO | Security news, features and analysis about prevention, protection and business innovation - The 17 biggest data breaches of the 21st century, 2018, Available: https://www.csoonline.com/article/2130877/data-breach/the-biggest-data-breaches-of-the-21st-century.html

[3] West A., S. Prettyman *Practical PHP 7, MySQL 8, and MariaDB Website Databases: A Simplified Approach to Developing Database-Driven Websites* (2nd ed.), Apress, 2018.

[4] W3Techs - Usage Statistics and Market Share of Server-side Programming Languages for Websites, 2018, Available: https://w3techs.com/technologies/overview/programming_language/all

[5] W3Techs - Usage Statistics and Market Share of Content Management Systems for Websites, 2018, Available: https://w3techs.com/technologies/overview/content_management/all

[6] Maraj, A., E. Rogova, G. Jakupi, X. Grajqevci, Testing Techniques and Analysis of SQL Injection Attacks, *2nd International Conference on Knowledge Engineering and Applications*, 2017.

[7] Raza, M., M. Iqbal, M. Sharif, W. Haider, A Survey of Password Attacks and Comparative Analysis on Methods for Secure Authentication, *World Applied Sciences Journal* 19 (4): 439-444, 2012.

[8] Statista - Average number of characters of leaked user passwords worldwide as of 2017, Available: https://www.statista.com/statistics/744216/worldwide-distribution-of-password-length

[9] Provos N., D. Mazières. A Future-Adaptable Password Scheme. *Proceedings of the FREENIX Track: 1999 USENIX Annual Technical Conference*, Monterey, California, USA, 1999.

[10] Php.net documentation - PHP: password_hash - Manual, Available: https://www.php.net/manual/en/function.password-hash.php

[11] GitHub - 8x GTX Titan X cudaHashcat Benchmark, Available: https://gist.github.com/epixoip/63c2ad11baf7bbd57544

[12] Wiemer, F., R. Zimmermann, High-Speed Implementation of bcrypt Password Search using Special-Purpose Hardware, 02.2015.

[13] Biryukov, A., D. Dinu, D. Khovratovich, *Argon2: the memory-hard function for password hashing and other applications*, University of Luxembourg, Luxembourg, Version 1.3, 03.2017, Available: https://www.cryptolux.org/images/0/0d/Argon2.pdf

[14] GitHub - Compatibility with the password_* functions that ship with PHP 5.5, Available: https://github.com/ircmaxell/password_compat

Information about the authors:

Nedyalko Katrandzhiev: University of Food Technologies, Computer Systems and Technologies, Assoc. Prof., Web Programming, Internet Security, Search Engine Optimization (SEO), Internet of Things (IoT), Internet of Food (IoF).

Daniel Hristozov: University of Food Technologies, Computer Systems and Technologies, PhD Student, Web Programming, Internet Security, PHP.

Borislav Milenkov: University of Food Technologies, Computer Systems and Technologies, Assoc. Prof., Database, Computer Networks, End-user Programing.

Manuscript received on 4 April 2019