Jocaxian Protection for Database Passwords

Joao Carlos Holland de Barcellos

São Paulo University

ABSTRACT

We will show a new database password protection method that can be used in programming languages, specially in those that are interpreted, where the executable code is humanly readable as well, being able to leave the database password exposed, with the risks of information theft and data changes.

Classification: H.2.7

Language: English

London Journals Press

LJP Copyright ID: 975731
Print ISSN: 2514-863X
Online ISSN: 2514-8648

London Journal of Research in Computer Science and Technology

Volume 19 | Issue 2 | Compilation 1.0

Jocaxian Protection for Database Passwords

Joao Carlos Holland de Barcellos

ABSTRACT

We will show a new database password protection method that can be used in programming languages, specially in those that are interpreted, where the executable code is humanly readable as well, being able to leave the database password exposed, with the risks of information theft and data changes.

I. INTRODUCTION

In several interpreted programming languages, such as, for instance: PHP, Lisp, Python, Java etc.. whenever we need to access a database we must provide - among other parameters - a password for the programming-language bank opening command, and thus, inside the program we'll have to pass this password to the opening command.

Therefore, the password becomes dangerously exposed in the program that stays on the application server

We must note that, in several cases, mainly on especially in multilayer architectures, the database server, where the corporate data stay, can be much more important than the application servers, so it would not make much sense to spend 'fortunes' on protecting of this server if the entire bank vulnerability is on the application servers where passwords are exposed.

Here also goes the saying: "The safety of the chain is not higher than its weakest link"[03].

II. CRYPTOGRAPHY

A slightly more secure way would be to store the encrypted database password in a constant in the source program, or in an external file.

However, since the decrypt function is in interpreted code, which is humanly readable, it could be easily copied and executed by the intruder, and so - again

- the database password could be decrypted and revealed.

III. PRECOMPILERS AND OBFUSCATORS

A precompiler transforms the source code into an intermediate language (not humanly readable) that can then be executed by an interpreter program.

However, many precompilers have tools that "decompile" the compiled code. Let's look at the PHP language:

"...No protection guarantees 100% safety. Even the best paid solutions presented above are not 100% secure. The site zendecode.com, for example, claims to instantly decompile Zend Guard code and Icon Cube ...."

An obfuscator is a program that makes source code extremely difficult to understand. It does this, for example, by taking the comments of the programs, changing the name of the variables, the name of the functions, taking out the indentations, etc ...

Of course this makes the process of finding the password in the source program daunting a little more, but with a little patience it is still possible to find it.

London Journal of Research in Computer Science and Technology

IV. COMPILED LANGUAGES

It is important to note that, depending on the danger and importance of the database, not even compiled languages can safely hide database passwords because there are decompiler programs [06] and disassemblers [07] that can convert the executable code back in a source program, that is, it does the reverse work of compilation, and again, the database's password could be dangerously exposed.

V. THE JOCAXIAN METHOD

One solution to this problem is to tie / merge / encrypt the Database (BD) password with the User ID (ID) and the System User password.

In this way, the intruder, even in possession of the source programs, will not be able to find the database password without knowing the password and the user ID.

As the user ID / password is not stored on the application server - and strictly speaking, nowhere - because it is entered by the user when it is identified ('logged in') in the system

- it is extremely difficult to discover the database password without this data.

In order to implementing this method we will basically need two functions:

The first is used to merge (or shuffle or encrypt) the database password with the user ID and password:

Blend_DB_password=merge(User_ID, User_Password, DB_password);

The second is used to 'undo' (or unscramble or decrypt) the password (previously merged) using the user ID and password:

BD_password = unmerge (User_ID, User_Password, Blended_DB_Password)

This way, the DB password would be visible only in the program variables at runtime, and not statically saved in the source program.

The Pre Password Table Of course we will need a Table (let's call it the Pre-Password Table) to store the merged passwords, which would be formed by one line for each user of the system.

A possible implementation of the Pre_Password table would be having 3 fields per line:

(ID_USER_HASH, PASSWORD_USER_HASH, BLENDED_DB_PASSWORD)

This table could be written as a text table on the application server itself or, in the case of relatively many users, in *another* database, intended to store only this table and of course, with another own password and unprotected.

VI. A SIMPLE IMPLEMENTATION

Of course the user ID / password comes from a data field of the application at run time, that is, it comes from fields that the user fills out of his terminal and, therefore, don't need to be recorded on any part of the system.

Our Pre_Password Table could so be formed by 3 fields:

The first field of the table, USER_ID_HASH, would be a field with HASH of the user ID, such as the HASH function that could be , as example, the sum of the bytes of its ID multiplied by the order in which this byte appears in the string.

The second field of the table, the PASSWORD_USER_HASH, would be just a field with the HASH of the user's password. This field is not really necessary but would, along with USER_ID_HASH, be used to form a key that would identify which line of the PreSenha table is located the user, and then find the field that really matters: the MERGED_DB_PASSWORD.

The third field the MERGED_DB_PASSWORD would be formed by a reversible function that associates the USER_ID , USER_PASSWORD and the DB_PASSWORD.

As an example, we could have, in case of numerical fields:

MERGED_DB_PASSWORD = merge (USER_ID, USER_PASSWORD, DB_PASSWORD) = HASH (USER_ID + USER_PASSWORD) + DB_ PASSWORD
And its reverse function:

DB_PASSWORD = unmerge (USER_ID, USER PASSWORD, MERGED_DB_PASSWORD) = MERGED_DB_PASSWORD - HASH (USER_ID + USER_PASSWORD)

Simple Example

To exemplify and make it easy, let's assume that all passwords are numeric up to 5 digits.
Thus, using prime numbers [05], we could write:

USER_HASH_PASSWORD=REST_DI ISION((USER_PASSWORD * 104711) , 9991)

USER_ID_HASH= "X" + USER_ID; // (to simplify)

For the merge function let's use the sum function as an example (in practice a more complicated function must be used):

Merge function (USER_ID, USER_PASSWORD, DB_PASSWORD)=DB_PASSWORD*2017 + USER_PASSWORD;
So that:

DB_MERGED_PASSWORD = DB_PASSWORD * 17 + USER_PASSWORD;

And the unmerge function would be:

Unmerge function (USER_ID, USER_ PASSWORD,DB_MERGED_PASSWORD) = (DB_MERGED_PASSWORD
USER_PASSWORD) / 17;

Trivial Example for Elucidation

Let's consider 2 users and their passwords as an example: MARIA, 1234
JOAO , 999

If the password of our BD is, for example: **54321**, and using the functions of the 'simple example' above, we will have in our PRE_PASSWORD table:

ID_HASH,PASSWORD_HASH,
DB_MERGED_PASSWORD (XMARIA, 9762, 924.691) (XJOAO ,519, 933.456)

When the system receives the user data (when logged in): ID = JOAO ; USER_PASSWORD = 999
It calculates the HASH of the user ID and password, in this example: HASH_ID= XJOAO

HASH_PASSWORD = REST_DIVISION((999 * 104711) / 9991) = 519

With this composite key (XJOAO, 519) the program finds the line, in the Pre_Password table, which contains the merged password, in our example = 933.456

And it uses the unmerge function = (933.456 - 999) / 17 = 54321 and retrieves the database password ☺

VII. CONCLUSION

We have seen that database security (DB) is sometimes much more important than applications. And, as the link of a chain, the security and privacy of the DB is no greater than the security and privacy of application servers.

To make DB security even greater and break the weak link of application servers is unnecessary, and often even pointless, to overshadow source programs, which embeds database passwords, rather than this, is better merge (encrypt) the password of the DB with the ID and Password of the user so that it becomes almost impossible to get it by an intruder out of the system.

REFERENCES

1. Interpreted Language https://en.wikipedia.org/wiki/Interpreted_language
2. Applications Architecture in 2, 3, 4 or N layers. https://www.softwaretestingmaterial.com/software-architecture/
3. Ramses II Phrases. https://www.phrases.org.uk/meanings/the-weakest-link.html

London Journal of Research in Computer Science and Technology

5 How to compile a PHP file?_https://softwareengineering.stackexchange.com/questions/369122/can-i-compile-php-to-hide-the-code
5 Number Theory/10000 cousins https://primes.utm.edu/lists/small/10000.txt
6 Decompiler https://en.wikipedia.org/wiki/Decompiler
7 Disassembler https://en.wikipedia.org/wiki/Disassembler

An Efficient Method of Vehicle Registration Number Plate Extraction and Recognition using Image

S M Sohel Rana, M. Humayun Kabir, Md. Shohel Rana & Sanjoy Kumer Sarker

Islamic University

ABSTRACT

Due to increasing population, the number of vehicles is growing day by day. These increases numbers of vehicles create various problems for traffic police such as signal light violations, parking problems, wrong lane violations and toll booth violations. This research will be helpful to control these traffic violations for traffic police. Moreover, it will also be helpful for the other number plate extraction and character recognition applications. We proposed in this research car registration number extraction and character or number recognition. This research will be able to extract and recognize alphanumeric characters in a given image. The final output will be stored in a text file. This file will have extracted alphanumeric characters. This technology will be, cost effective, fast and highly accurate. In this research we will try various algorithms and logics in MATLAB and find best process to extract number plate and recognition of alphanumeric each character. The main goal of this research is to reduce the manpower, cost, time and to make the process quick and highly available.

Keywords: character recognition, extraction, alphanumeric character, traffic.

Classification: I.4.1

Language: English

London Journals Press

LJP Copyright ID: 975732
Print ISSN: 2514-863X
Online ISSN: 2514-8648

London Journal of Research in Computer Science and Technology

Volume 19 | Issue 2 | Compilation 1.0