Argon2: The Secure Password Hashing Function

Aleena Theresa George
PG Scholar
Department of Master of Computer Applications
Amal Jyothi College of Engineering

Dr. Juby Mathew
Associate Professor
Computer Science & Engineering
Amal Jyothi College of Engineering
Kottayam, Kerala, India

Abstract— A password is a string of characters used to protect access to specified resources from unauthorised users. Anyone with access to the resources should be able to keep the password secret in order to prevent resources from being stolen or misused by those who do not have permission. The system that serves as a resource storage and is also in charge of granting access to those resources should be secure enough to handle access verification. The computer should have a means to validate the password in order to authenticate a person using a password. A security issue arises when a plain password is saved somewhere to be used for verification. Anyone who can identify the location where the password is kept and read it can have access to all of the system's resources. The authentication system should be able to store and validate passwords in a secure manner. With the advancement of computer hardware, the chances of cracking the security of hashing algorithms, such as Bcrypt, are increasing. Certain gear has been developed particularly to execute certain cracking algorithms at their best. To avoid the chance of cracking utilising high-end computer hardware, a more secure hashing technique is required. Argon2 is yet another hashing function that can outperform current hardware in terms of compute power. Argon2 addresses several important flaws in existing algorithms by being designed for the fastest memory filling rate and the most efficient usage of numerous processing units while still offering protection against tradeoff attacks.

Keyword: Argon2, Hashing, Bcrypt, Scrypt, password.

I. INTRODUCTION

Despite its flaws, passwords are still the most common method of authentication for numerous web applications. In most cases, passwords are saved in hashed form in a server's database. Due to the low entropy of passwords, adversaries frequently capture these databases and use dictionary attacks on them. To address these difficulties, protocol designers employ a variety of techniques. A password has been hashed with a random salt value since the late 1970s to prevent the detection of duplicate passwords across multiple users and services. Hash function computations, which have become faster and faster as a result of Moore's law, have been called several times to increase the attacker's cost of password trial. Argon2 the endorsed password hashing algorithm through the Password Hashing opposition is a current set of regulations for securely hashing passwords.

Argon2 addresses numerous key downsides of gift algorithms in that it's miles designed for the very quality memory filling charge and powerful use a couple of computing gadgets at the same time as despite the fact that supplying defense in opposition to tradeoff assaults. Argon2 is parameterized with the aid of using 3 first rate elements: A reminiscence cost that defines reminiscence usage of the set of rules, a time cost that defines the execution time of the set of policies and the massive fashion of iterations and a parallelism issue which defines the amount of parallel threads. Argon2 is to be had in terrific flavors Argon2i and Argon2d. Argon2i that is optimized for password hashing and password based absolutely key derivation Argon2d is quicker and makes use of records mounted memory get entry to making it specifically resistant in the direction of GPU cracking assaults and suitable for programs without a threats from difficulty channel timing assaults. Argon2 summarizes the kingdom of the artwork within the layout of reminiscence-difficult features. it is a streamlined and simple design. It targets at the very best memory filling charge and powerful use of a couple of computing devices, at the same time as nonetheless imparting defense towards tradeoff assaults.

DOI: 10.5281/zenodo.5091700
ISBN:978-93-5426-386-6@2021 MCA, Amal Jyothi College of Engineering Kanjirappally, Kottayam

II. LITERATURE REVIEW

Alex Biryukov at el[1] Should memory addressing (indexing functions) be input-independent, input-dependent, or a combination of both? Because an adversary can precompute the missing block by the time it is needed, the first type of technique, where the memory read address is known in advance, is immediately subject to timespace tradeoff attacks.

Alex Biryukov at el[1] Argon2 for the applications that aim for high performance. Both versions of Argon2 allow to fill 1 GB of RAM in a fraction of second, and smaller amounts even faster. It scales easily to the arbitrary number of parallel computing units. Its design is also optimized for clarity to ease analysis and implementation.

COLIN PERCIVAL at el[3] The initial UNIX CRYPT function iterated the DES cypher 25 times to raise the expense of an attack, whereas Kamp's MD5-based hash [18] iterated the MD5 block cypher 1000 times; more recently, Provos and Mazieres' bcrypt and RSA Laboratories' PBKDF1 and PBKDF2 are specifically defined to execute a user-specified number of iterations2, with the number of iterations presumably being stored along with the password salt.

Christian Forler[5] Argon/Argon2d/Argon2i : First, the padding phase receives the internal state produced from pwd. Following the padding phase, the internal state is overwritten at least R times using the methods ShuffleSlices and SubGroups. Argon is not vulnerable to GC/WGC attacks because of this structure and the fact that pwd is only used to initialise the state. As a result, Argon2d and Argon2i offer a similar level of resistance against (W)GC attacks as Argon.

Alex Biryukovat[6] To avoid becoming a bottleneck, the compression method should preferably execute at least as well as memcpy() or a similar function, which can operate at 0.1 cycle per byte or even quicker. This is substantially faster than standard stream cyphers or hash functions, but we may not require those primitives' strong features.

Daniel Ryan Levyson at e1[7] HMAC, PBKDF2, and Scrypt are more options besides Bcrypt, which has already been discussed. Because of its flexibility in determining computation cost, Bcrypt is better for password hashing than HMAC and PBKDF2. Bcrypt was created with the goal of reducing computational costs. Specialized computer chips, such as FPGAs, ASICs, and GPUs, are now easier to access by an attacker than massive amounts of memory. This fact reveals Bcrypt's Makalah IF2120 Matematika Diskrit – Sem. I Tahun 2019/2020 flaw, which focuses on computation costs. Scrypt is a newer version of Bcrypt. Scrypt aims for huge memory, however because of a minor time-memory tradeoff, compact implementations with the same energy cost are possible. When it comes to segregating time and memory expenses, Scrypt isn't very versatile.

III. PROBLEM DEFINITION

If the protocol inventor wants to avoid all of the problems associated with key generation, storage, and updating, he only has a few possibilities. Bcrypt and scrypt based on pbkdf2 blowfish. Only scrypt aspires to large memory, however compact implementations with the same energy cost are conceivable because to a simple time memory tradeoff.

Clear text password storage is the same as writing them down on a piece of digital paper. An attacker who obtained access to the database and stole the passwords table would be able to access each user account. This problem is exacerbated by the fact that many users reuse or use variations of a single password, giving the attacker access to services other than the one that has been compromised. That all sounds like a nightmare in terms of security!

The complexity of the present schemes is another issue. Scrypt calls a stack of subprocedures whose design reasoning isn't fully justified (e.g., scrypt calls SMix, which calls ROMix, which calls BlockMix, which calls Salsa20/8, and so on). It's tough to analyse, and even more difficult to gain confidence in, and it's rigid when it comes to dividing time and memory costs. Simultaneously the history of cryptographic competitions has shown that the most secure designs are simple with every constituent well motivated and as few entrance points as feasible for cryptanalysts

The Password Hashing Competition, which began in 2014, brought attention to the following issues:

- Because an adversary can precompute the missing block by the time it is required, should memory addressing indexing functions be input independent, input dependent, or a combination of both? Because an opponent may precompute the

DOI: 10.5281/zenodo.5091700
ISBN:978-93-5426-386-6@2021 MCA, Amal Jyothi College of Engineering Kanjirappally, Kottayam

missing block by the time it is required, the first type of methodology in which the memory read address is known in advance, is immediately subject to timespace tradeoff attacks. As a result input dependent algorithms are subject to side channel attacks as timing information enables a faster password search.

- Is it better to fill more memory and suffer from timespace tradeoffs, or is it better to make more passes over the memory to be more robust? Due to the lack of general tradeoff tools that would examine the security against tradeoff assaults, as well as a single methodology to quantify adversary costs, this topic was difficult to address.
- What method should be used to compute the input independent addresses several options that appeared to be safe have been breached.
- How big should a single memory block be? Because of the CPU cache's spacial locality principle, reading smaller random-placed blocks takes longer (in cycles per byte). Due to the limited number of long registers, larger blocks are more complex to process.
- How do you choose the internal compression function if the block is large? Should it be cryptographically secure, or should it be more lightweight, with simply basic input mixing? Many applicants merely suggested an iterative structure and argued that cryptographically strong transformations were unnecessary.

IV. IMPLIMENTATION

Argon2 is a hashing method that we offer. To combat GPU assaults, Argon2 uses a preset memory capacity, CPU time, and a degree of parallelism to ensure security against brute force assaults. Argon2, the Password Hashing Competition's recommended password hashing algorithm, is a current algorithm for hashing passwords securely. Argon2 addresses several important flaws in existing algorithms by being designed for the fastest memory filling rate and the most efficient usage of numerous processing units while still offering protection against tradeoff attacks. It summarises the state of the art in memory hard function design. It's a simple and basic design. It aims for the highest memory fill rate and the most efficient utilisation of numerous compute units while yet defending against tradeoff assaults. Argon2 is designed for x86 architecture and takes advantage of latest Intel and AMD processor cache and memory structure. Argon2 is available in two forms: Argon2d and Argon2i. Argon2d is faster and employs data-dependent memory access, therefore it's ideal for cryptocurrencies and applications where side-channel timing attacks aren't a concern. For password hashing and password-based key derivation, Argon2i employs data-independent memory access. Argon2i is slower because it makes more memory passes to prevent against tradeoff attacks.

The password hash() function now accepts either PASSWORD ARGON2I or PASSWORD ARGON2I as the algorithm, as well as the memory cost, time cost, and parallelism degree as parameters. When working with Argon2. The examples below demonstrate the new features. If an Argon2 hash is given, the password verify() method returns true or false. This method does not require any API changes.

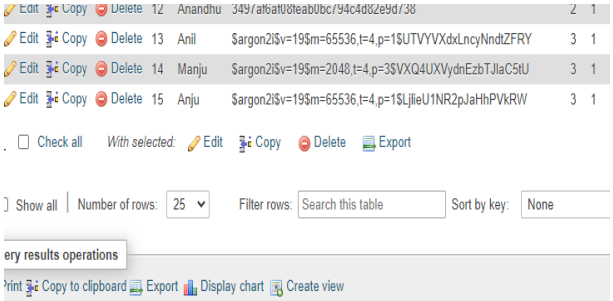
```php
session_start();
$con=mysqli_connect("localhost","root","","projectdb");
$name=$_POST['name'];
$addr=$_POST['address'];
$phone=$_POST['phone'];
$dist=$_POST['dist'];
$city=$_POST['location'];
$email=$_POST['email'];
$uname=$_POST['username'];
$pass=$_POST['password'];


$pass=password_hash($pass, PASSWORD_ARGON2I);
$sc=$_POST['sc'];
$service=$_POST['service'];

    $sql="select location_id from tbl_location where location_id='$city'";
    $result=mysqli_query($con,$sql);
    $loc_row=mysqli_num_rows($result);
    if($loc_row>0)
    {
    while($loc_data=mysqli_fetch_array($result))
    {
        $loc_id=$loc_data['location_id'];
```

A 16-byte salt can optionally be specified, albeit it is deprecated within password_hash(). If you don't specify a salt, a 16-byte salt will be produced for you.

V. RESULT



DOI: 10.5281/zenodo.5091700
ISBN:978-93-5426-386-6@2021 MCA, Amal Jyothi College of Engineering Kanjirappally, Kottayam

The argon2 hash in the preceding output is expressed in a standardised format, and it contains the Argon2 algorithm's configuration parameters. the random salt + the derived key The salt and derived key should be different at each code execution by design.
Let's have a look at what this means:

- $argon2i — the variant of Argon2 being used.
- $v=19 — the version of Argon2 being used.
- $m=65536,t=4,p=3 — the memory (m), iterations (t) and parallelism (p) parameters being used.
- $UTVYVXdxLncyNndtZFRY—the base64-encoded salt, using standard base64-encoding and no padding.

VI. CONCLUSION

Despite the fact that there are other methods for authentication, the password is still the most used. A system should include a reliable authentication gate because it is so important in securing access to resources. Password hashing is a secure way of storing passwords that conceals the bare password form from an attacker. Preimage attack, second preimage attack, and collision attack resistance should all be present in a good hash function. The threat of brute force, dictionary, and rainbow table attacks necessitates a hash function that is both slow and scalable enough to keep up with increasing computational power.

Despite the fact that Bcrypt can adapt to increased processing power, it solely considers computation time. Scrypt, as a younger hash algorithm, is unable to distinguish between time and memory costs. By taking into account today's compute power and unique technology, Argon2 improves on prior hash functions to prevent broad cryptographic threats.

VII. REFERENCES

[1] M. E. Hellman, “A cryptanalytic time-memory trade-off,” *Information Theory, IEEE Transactions on*, vol. 26, no. 4, pp. 401–406, 1980.

[2] M. Robshaw and O. Billet, *New stream cipher designs: the eSTREAM finalists*. Springer, 2008, vol. 4986.

[3] Colin Percival, “Stronger key derivation via sequential memory-hard functions,” 2009

[4]Alex Biryukov and Dmitry Khovratovich. ARGON and Argon2: Password Hashing Scheme.

[5]Christian Forler⋆ , Eik List, Stefan Lucks, and Jakob Wenzel .Overview of the Candidates for the Password Hashing Competition And Their Resistance Against Garbage-Collector Attacks

[6]Alex Biryukov, Daniel Dinu, Dmitry Khovratovich. Argon2: new generation of memory-hard functions for password hashing and other applications.

[7]Daniel Ryan Levyson. Argon2: The Better Password Hashing Function Than Bcrypt

DOI: 10.5281/zenodo.5091700
ISBN:978-93-5426-386-6@2021 MCA, Amal Jyothi College of Engineering Kanjirappally, Kottayam