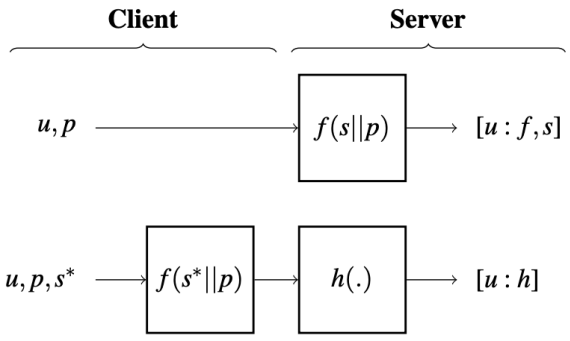
Aryan Yadav, Samvit Jatia
Computer Security and Privacy
CS-2362-1
Professor Mahavir Jhawar
Final Project Report
Client-Side Password Hashing

Introduction

Since the beginning of the digital privacy concepts, password security has been a concern of the highest priority. Getting access to passwords has always been a primary target for attackers due to the fact that users tend to reuse the same password on different systems, a breach in one system can have consequences over a much larger scope. In particular, with the advent of the Internet of Things (IoT), where potentially every embedded low- power device acts as a server, the issues with passwords have become a dilemma. As IoT devices are usually more vulnerable, getting access to their password database is relatively easy. Also, their ability to handle password authentication is significantly hin- dered compared to traditional client/server systems due to limited computation, energy and amounts of memory available for code and data.

Modern password hashing algorithms are specifically designed to eliminate the possible attacker advantage gained from accelerators. Removing such advantages comes at the cost of excessive computation and memory usage. However, there is still a lack of use of such algorithms with strong security parameters on regular systems and more so on resource-constrained systems.

To address the security versus complexity challenge in highly-constrained devices, we propose moving the resource-hungry password hashing step to the client. A Client-side Password Hashing scheme, based on Argon2, that allows user authentication in highly-constrained devices.



The figure above illustrates the idea of client side hashing in comparison to general server side hashing algorithms. By moving the resource intensive operation to the client, the server only needs to compute a low cost one-way hash function (e.g., SHA-256) to prevent leaked database entries can directly be used to access the server.

Literature Review

In today's world, password hashing devices have a three step process to make their password hashes more secure and slow down adversaries: (1) Adding salt, i.e, a unique string, with the password and then, finally, using a one-way hash function, like a cryptographic hash function) to make each password entry unique in the database so that when an offline adversary chooses to implement a dictionary attack they will have to execute a hash once per guess and entry (Wilkes). (2) Increased processing power so that execution of each hash is slow (Morris and Thompson, 1979). (3) High memory and bandwidth utilization such that it prevents the use of efficient GPUs, FPGAs, or ASICs to compute the hash (Hatzivasilis 2017).

Argon2 (Biryukov et al. 2017), is a cryptographic library that won a competition in 2013, aiming to modernize one-way functions for hashing passwords concerning new attack methodologies. Argon2 has two main variants, Argon2d and Argon2i, and various supplementary variants, including Argon2id. We have implemented client-side hashing of passwords using the Argon2id function. Argon2id works as Argon2i for the first half of the first iteration over the memory and as Argon2d for the rest, thus providing both protection against side-channel attacks and against cost savings due to TMTOs (Biryukov et al. 2017).

Implementation

In this project we have accomplished a simple client-server register/login authentication Architecture, in python, using openssl implementation, "socket" python module for the client and server set-up, "sqlite3" python module for users database, "pyargon2" python module for the argon2 password hasher object and "resource" python module to simulate constraint devices. The final code is available in a Github repository.

Design

The figure below shows explicitly how we have implemented client-side password hashing, helping you better understand the structure and working of it.

Algorithm 1 Clipaha's client-side password hashing process

```
function CLIPAHAHASH(Domain, Username, Password)
    NUsername ← CANONICALIZE(Username)
    Salt ← DELIMIT(Domain, NUsername)
    return PASSWORDHASH(Salt, Password)
end function
```

Algorithm 2 Clipaha's server-side hashing process

```
function CLIPAHASERVERHASH(ClipahaHash)
    return HASH(ClipahaHash)
end function
```

Hashing Algorithm

We propose an algorithm to perform password hash- ing on the client. With server relief, the client will provide the brunt of the work to hash the password. Hence, the client must be provided with all necessary parameters, including the salt. Traditionally, said salt would be stored in the password database on the server and looked up when the client provides the username. However, this would open the system to enumeration attacks, as online attackers can easily detect the lack of salt for unregistered users or see how they change over time. To circumvent enumeration attacks, as well as avoiding salt collisions, we do not involve the server in the hashing process and lets the client calculate the salt locally based on the user's unique identifier. This choice deviates from the standard approach of using a random salt. However we argue that a random salt will not gain a significant structural advantage by having a predictable, low-entropy salt compared to the case where it has access to a random, high-entropy one.

Our algorithm is divided into two parts, The first part is executed in three steps and the latter in one. For the first part, the function translates the username into its canonical representation by lowering the case of the username to gain case insensitive usernames.This ensures the system will return the same hash for usernames that should be considered equivalent. Second, we add a salt to the password by reversing the user's identifier and multiplying the size of it by a factor of 8. We do this to ensure we have a deterministic, collision free salt at all times. Finally, the algorithm computes and returns the hash of the password and the salt using argon2 as the hash function.

Continuing with the second part of the algorithm, The input for this would be the output of the above algorithm.This function only performs a call to a one-way function HASH, which does not need to be computationally expensive. This prevents *pass-the- hash attacks.* Otherwise, an offline attacker (with read-only access to the database) could replay the entry in the database to access the service.

Constraint Device

The challenge was to be able to perform the computing-power-intensive task of client-side hashing on a constraint device and so, to simulate a constraint device on our computers we have primarily used the "resource" module in python, along with the "os" and "signal" modules, by limiting CPU power usage by using the "getrlimit()" and "setrlimit()" functions from the "resource" module. We were able to successfully run the argon2 password hashing object, on the password input by the user, on the client-side to mimic a constraint device in under ~2ms.

Conclusion

In conclusion, we've taken inspiration from the Clipaha paper. I have added it as a citation source so it comes in the references section also below.

References

Biryukov, A., et al. "Argon2: the memory-hard function for password hashing and other applications."

Technical Report. *Password Hashing Competition*, 2017.

Hatzivasilis, G. "Password Hashing Status." *Cryptography*, vol. 1, no. 2, 2017.

Morris, R., and K. Thompson. “Password security: A case history.” *Communications of the ACM*, vol. 22, no. 11, 1979.

Riera, Francisco Blas Izquierdo, et al. *Clipaha: A Scheme to Perform Password Stretching on the Client*. no. 1746, Cryptology ePrint Archive, 2022.

Wilkes, M. V. “Time-sharing computer systems.” *Number 5 in Computer Monographs*, New York, American Elsevier, 1968.