IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Vol. 8, Issue 6, June 2019

Isilon Credential Vault: An Authentication Provider

Prof. R.B. Murumkar[1], Sanket Kulkarni[2], M. Shushanth[3], Priyanka Fulzele[4], Esha Shah[5]

Department of Information Technology, Pune Institute of Computer Technology, Pune[1,2,3,4,5]

Abstract: In this era of digitisation, information security is of utmost importance. It is provided usually by PINs or passwords, whose security is very crucial in order to preserve data integrity. This paper explains insights on the multi-level security being implemented for the credential vault. Passwords in the system are stored as salted hashes. Database is then encrypted and then , the key for decryption would be stored externally. Also we aim to provide APIs for the system through which open source applications like Apache can make use of the authenticator to store credentials.

Keywords: Salting, Hashing, Authentication, Security, Attacks

I. INTRODUCTION

Data security is one of the most important aspect of the era of digitisation, thus it is an important issue to be looked after. The purpose of information security policies is to preserve confidentiality, integrity and availability of data. The most common way to authenticate the data is by using either passwords or PINs. Most password managers available as of now are very vulnerable to attacks, which compromises the data privacy. The way password managers that come with browsers store application details is not very secure technique used and the data can be retrieved without much efforts. A secure system makes data available only to the authorised users.

II. EXISTING METHODOLOGIES

Table I: Existing Password Managers

Platform	Password Manager	Same Protocol and action	Different Protocol	Different form action on load	Different form action on submit	Auto complete = off	Broken HTTPS
MAC OSX 10.9.3	Chrome34	Auto	No Fill	Manual	Auto	Auto	No fill
	Firefox 29.0.1	Auto	No Fill	None	Auto	No fill	Auto
	Safari 7.0.3	Auto	No Fill	Auto	Auto	Auto	Auto
Safari Ext	1Password 4.4	Manual	Manual	Manual	Manual	Manual	Manual
Safari Ext	LastPass 3.1	Auto	Manual	Warning	Auto	Auto	Auto
Safari Ext	Keeper 7.5.26	Manual	Manual	Manual	Manual	Manual	Manual
Windows 8.1 Pro	IE 11.0	Auto/Man	No Fill	Auto/Man	Auto/Man	Auto/Man	Manual
	KeePass 2.24	Manual	Manual	Manual	Manual	Manual	Manual
IE addon	IdentitySafe 2014	Auto	Auto	Auto	Auto	Auto	Auto
iOS 7.1.1	Mobile Safari	Auto	No Fill	Auto	Auto	No Fill	Auto
	1Password4.5	Manual	Manual	Manual	Manual	Manual	Manual
	LastPass Tab 2.0	Auto	Manual	Auto	Auto	Auto	Auto
	Chrome 34.0	Auto	No Fill	No Fill	Auto	No Fill	Auto

The password managers we intend to survey include:

1. Desktop Browser PMs: Google Chrome 34, Microsoft Internet Explorer 11, Mozilla Firefox 29, and Apple Safari 7.
2. Third Party PMs: 1Password, LastPass, Keeper, NortonIdentitySafe, PasswordSafe, and KeePass.
3. iOS PMs: Mobile Safaris password manager synchronises with the desktops safari through Apples iCloud Keychain synchronisation service.
4. Android PMs: All these PMs offer an auto-fill functionality, wherein the PM automatically populates the user credentials within the users web browser.

Another password manager we closely studied was KeePass. It is a free and open source password manager which requires the user to remember only one master key to unlock the database. Database encryption is carried out using AES

IJARCCE
ISSN (Online) 2278-1021
ISSN (Print) 2319-5940
International Journal of Advanced Research in Computer and Communication Engineering
Vol. 8, Issue 6, June 2019

and Twofish Algorithm. KeePass is reliable because of its strong security, easy database transfer, support of password groups and ability to generate strong random passwords to name a few.

III. PROPOSED ARCHITECTURE

The vault will work on the basis of certificate authentication. Application is supposed to authenticate to the user with a valid certificate when asked for. The flow of execution will be as follows:

A. For an existing user:
User requests for applications certificate. On receiving so, it sends its credentials (user- name, passwords). The application takes the password and fetches the appropriate salt, hashes it with the password and checks to see the results in the database. If an appropri- ate match is found, user is logged into the system. The hashed password is retrieved from the database. The entered password is compared with the hashed password using Argons crypto pwhash str verify() function.

B. For a new user:
User requests for the certificate and on obtaining the certificate, user provides with its user- name and password. A randomised salt is generated for the user and is stored in the database. The salt along with the password is hashed and sent to the database for storage and future retrieval. In the key generation step, key is generated using argons crypto pwhash() func- tion. The hash for the password is created using Argons function crypto pwhash str() with opslimitset as 3 and memlimit as 100000. These values are changeable and have been set based on our experiments. Also, the username is encrypted using AES and this along with the hashed password is stored in the database. AES is a symmetric encryption algorithm that uses the same key for encryption and decryption of digest. Certificate based authentication uses digital certificate to identify the participants in communication before access is granted to a resource or application.

Two libraries we are using here are Sodium and Argon 2. Sodium is a software library which can be used for password hashing, encryption, decryption and much more. It is suit- able because it is cross-compilable, cross-platform and cross-language. Argon2 has a stream- lined and simple design. It is a state of the art design in the memory-intensive, memory-hard functions. It has 2 flavours, Argon2i and Argon2d. Argon2i uses data-independent memory access. This approach is preferred for password hashing and password based key generation. It has 2 inputs: a message P and a nonce S. These would be the password and the salt respectively. Our scheme provides more features and better trade-off resilience than pre-PHC designs and equals in performance with the PHC finalists. We aim to maximize the cost of password cracking on ASICs. There can be different approaches to measure this cost, but we turn to one of the most popular the time-area product. We assume that the password P is hashed with salt S but without secret keys, and the hashes may leak to the adversaries together with salts:

Tag H(P; S);
Cracker f(Tagi; Si)g:

In the case of the password hashing, we suppose that the defender allocates certain amount of time (e.g., 1 second) per password and a certain number of CPU cores (e.g., 4 cores). Then he hashes the password using the maximum amount M of memory. This memory size translates to certain ASIC area A. The running ASIC time T is determined by the length of the longest computational chain and by the ASIC memory latency. Therefore, we maximize the value AT.

The entire vault will be encrypted using an external key. This prevents it from attacks and acts as an additional security layer; an analogy for which can be thought of a case where a person has the treasure chest right in front of him but no key to unlock it and access its contents. When a user begins to use the application, the digital certificate pertaining to the application will be sent to the user. There will be a provision for mutual authentication. The exchange of information further will be done by using asymmetric keys. Passwords are salted using randomised salt and hashed using memory hard, slow hashing functions like Argon 2i. The hashed password is stored in a database which is then encrypted using APIs available in Sodium Library. Verification of passwords is done by performing similar salting and hashing on the input password and comparing with the database entry. This will be done using verification APIs available in Sodium Library and Argon.

C. Hashing
Producing hash values for accessing data or for security. A hash value (or simply hash), also called a message digest, is a number generated from a string of text. The hash is substantially smaller than the text itself, and is generated by a formula in such a way that it is extremely unlikely that some other text will produce the same hash value. Hashes play a role in security systems where they are used to ensure that transmitted messages have not been tampered with. The

IJARCCE
ISSN (Online) 2278-1021
ISSN (Print) 2319-5940
International Journal of Advanced Research in Computer and Communication Engineering
Vol. 8, Issue 6, June 2019

sender generates a hash of the message, encrypts it, and sends it with the message itself. The recipient then decrypts both the message and the hash, produces another hash from the received message, and compares the two hashes. If they are the same, there is a very high probability that the message was transmitted intact.

D. Salting Algorithm

Password salting is adding a random string of characters to passwords before their hash is calculated to make password hashing more secure and it makes them difficult to reverse. The random string of characters can be a combination of letters, numbers and other characters.

E. Levels of security

We are employing three levels of security in our application. The passwords with their respective salts are hashed and then stored in the database, on top of which symmetric encryption will be performed. The entire database is then encrypted with a single key, which will be stored externally; externally meaning on a different device from where the application/database is housed.

IV. SYSTEM DESIGN

The following diagram describes the workflow of the proposed system. The components are user, randomized salt generator, hashing component, encrypted database and external key. The key is stored externally and not on the system.

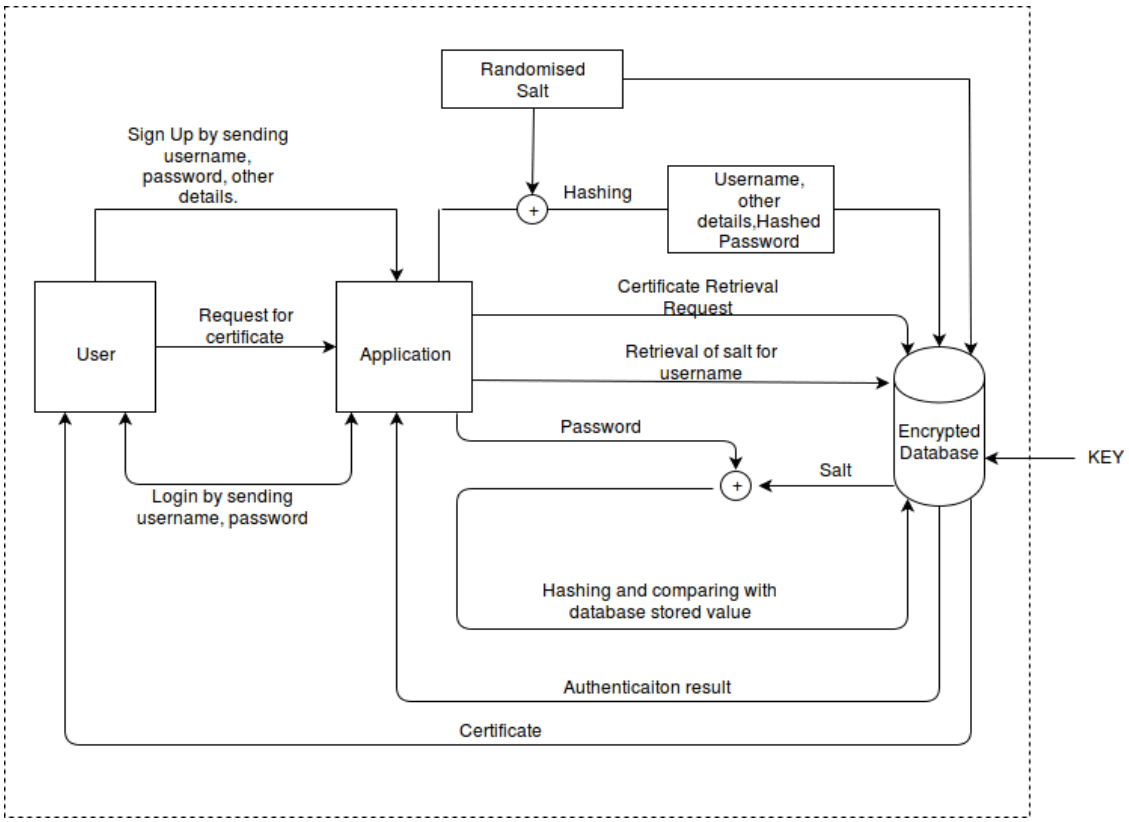

Fig. 1. System Architecture

To provide storage for the authenticator, we have used the inbuilt solution for the system making use of MySQL database at the backend. MySQL is a fast, easy-to-use RDBMS being used for many small and big businesses. MySQL is developed, marketed and supported by MySQL AB, which is a Swedish company. MySQL is becoming so popular because of many good reasons as listed below:

- MySQL is released under an open-source license. So you have nothing to pay to use it.
- MySQL is a very powerful program in its own right. It handles a large subset of the functionality of the most expensive and powerful database packages.
- MySQL uses a standard form of the well-known SQL data language.
- MySQL works on many operating systems and with many languages including PHP, PERL, C, C++, JAVA, etc.
- MySQL works very quickly and works well even with large data sets.

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Vol. 8, Issue 6, June 2019

- MySQL supports large databases, up to 50 million rows or more in a table. The default file size limit for a table is 4GB, but you can increase this (if your operating system can handle it) to a theoretical limit of 8 million terabytes (TB).
- MySQL is very friendly to PHP, the most appreciated language for web development.
- MySQL is customizable. The open-source GPL license allows programmers to modify the MySQL software to fit their own specific environments.

V. FEATURES AND OUTCOMES OF SYSTEM

- A "Password database" and authentication provider to store passwords and certificates for applications.
- Applications that perform password-based authentication must store those passwords as a salted hash. Salts will be at least 8 bytes long. There will be a separate salt per password entry.
- Components that authenticate to external systems using passwords will store those passwords in an encrypted format.
- The database (encryption keys and certificates) will be encrypted by a separate key. The key used for this encryption would be stored separately.
- Simple APIs to store and retrieve passwords from the database.
- The database will be FIPS compliant.
- At least 1 open source application storing its password and one application storing certificate on the authentication provider system.

VI. IMPLEMENTATION AND RESULTS

A. Overview

While stating our project objectives above, we have stated that at least one open source application shall store it's credentials, and perform authentication through our system. For showcasing this use case, we have provided a patch for the Apache server, wherein our system shall store Apache passwords as salted hashes and in turn, Apache shall perform authentication through the system. Below is a short description of how and why Apache authentication is used. If we have information of the server that is sensitive or intended for only a small group of people, there are access control techniques through which, access to only specific set of people can be given. This is achieved through a set of Apache Directives.

eg. 1) The AuthType directive selects that method that is used to authenticate the user.
2) The AuthUserFile directive sets the path to the password file that we will use.

Because storing passwords in plain text files or flat files has it's own many problems, we propose to store credentials safely in a database. Therefore, we first formed basic APIs for the system supporting CRUD operations.

- int db insert(App*conn, const char *username, const char *PASSWORD);

We consider a scenario where the database right now stores the username and pass- word. The password is stored as a salted hash, with a unique salt for each username. Hence this API will need the connection made to the database. Returns 1 if insertion is successful else returns 0.

- int db verify(App*conn, const char *password, const char* username);

Similar to the insertion process, the verification process shall return 1 on success(passwords match) else 0 on failure. Further, we can have this function to either return the ex- tracted password for the username or have it return the result of the verifying process(0 or 1).

- int db update(App*conn, const char* username, const char* password);

Updates the password based on the username entered. Fetches the previous hashed entry and replaces it by computing the salted hash of the new password. Returns 1 on success and 0 if the update operation fails.

To add Apache authentication to our system, we need to direct Apache to bypass it's inbuilt authentication routine, and make use of our defined method. For this, we have proposed one approach:

- Through writing a C handler: A handler (function) will be called when request to restricted directory is received, and the handler will accept the credentials and au- thenticate it against our database.

To achieve this, we added a module into Apache, called as mod authnz external, through which we directed Apache to refer to our verification API to perform authentication.

The module was included and added into Apache by using apxs, and linking the libraries using 'mysql config –cflags –libs' -lsodium, for the database connections and the sodium library respectively. Further, we added directives to the configuration files to restrict access to a particular directory on the server. Now when request for access for that particular directory is received, Apache prompts the client to authenticate himself. Upon receiving the credentials, authorization is performed with the help of the external module, which includes our verification API. If the client is authorized to access the resource, then an OK status is returned to the server, and client is allowed to access the

IJARCCE
ISSN (Online) 2278-1021
ISSN (Print) 2319-5940
International Journal of Advanced Research in Computer and Communication Engineering
Vol. 8, Issue 6, June 2019

resource. If however authentication fails, i.e, the client is not authorized, then a HTTP UNAUTHORIZED status is returned, where the server raises an alert saying that credentials are not authorized to access the resource in question.
The afore discussed module is built and compiled using apxs, which is a tool for building and installing extension modules. Upon further study, we found out that apxs is extensively used for the Apache HTTP server. This revelation made our module useful only for Apache. Hence, at this step, we decided to further add abstraction to our authentication provider, and the approach finalized after much discussion was to build kernel objects (ko's) through which any application can make use of the APIs.

B. Database Schema

The database design is very simple and consists of one database named Isilon. Every time a new client registers to the system using the register API, a new table for the specific client is allocated in the database, for which only the specific client is granted all the CRUD privileges. We store the details regarding the clients in another database. The relations have 3 useful columns, comprising of the username, password and the last time the details were updated with respect to the entry. The password column contains the stored password as salted hashes, whereas the other columns are encrypted with a key stored external to the system

C. User Actions

1. Add: Creating an entry in our design will be dependent on whether the user has registered with the system or not. This is achieved using the register API. After registration, when a user creates a new entry the application will add the selected password along with the credentials to the designated relation with respect to the user. The original password is inserted into the database as a salted hash
2. Verify: This operation is performed with the help of the verify API mentioned. The user has to provide a connection pointer to the database so that the credentials can be verified from the dedicated relation. If there is a match, the system returns 1 to the user.
3. Update/Edit: When a user wants to update or edit his password the application will perform the same operation as creating a new entry. The new password specified by the user will be added to the database as a salted hash, the salt generated being a new one. The correct set to edit is located by the system based on the username.
4. Delete: Deleting an entry will delete everything such as the id, username,password and the updated timestamp, including the metadata required for recomputation of the hashes for verification. The correct set to delete is located by the system based on username.

D. Snapshots of the System:

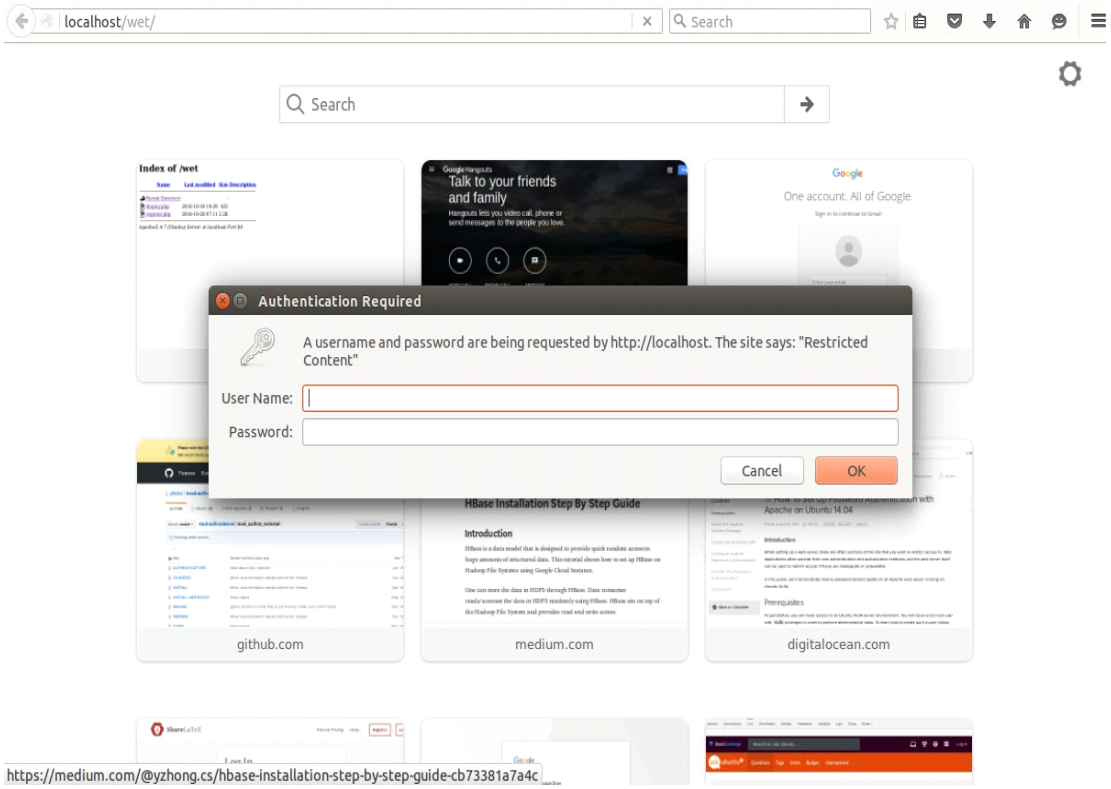

Fig. 2. Prompt for authentication from Apache

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Vol. 8, Issue 6, June 2019

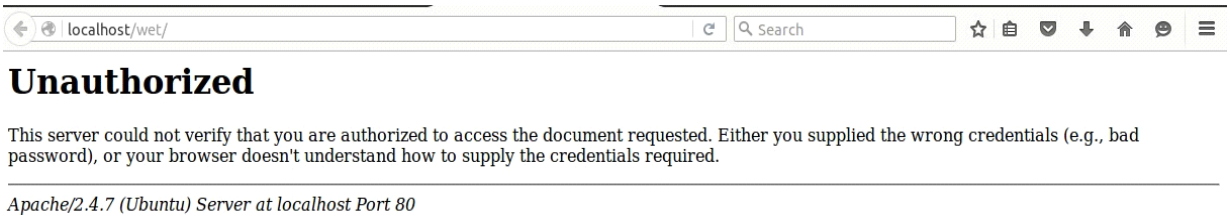

Fig. 3. Authentication failed

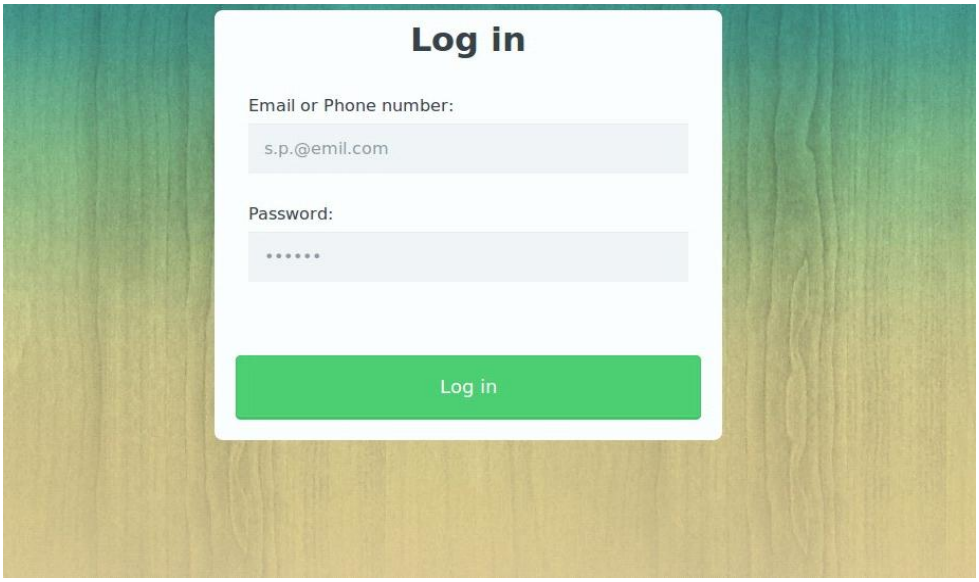

Fig. 4. Authentication successful. Resource access granted.

VII. CONCLUSION

The process of developing a system that will prevent attacks on stored passwords has indeed proven to be a daunting task, yet one which is necessary. We surveyed the types of attacks that happen on the already existing systems and have proposed and designed a system that shall overcome these shortcomings. While doing so, it is also necessary to take into account the robustness of the system, by making the tasks of attackers as slack, exigent and tough as possible.

REFERENCES

[1]. Praveen Gauravaram, Security analysis of salt‖passwordhashes, 2015 International Conference on Advanced Computer Science, Applications and Technologies, IEEE, vol 45, no. 8, pp.152-159.
[2]. Sakinah Ali Pitchay, Wail Abdo Ali Alhiagem, Farida Ridzuan, MadihahMohd Saudi, "A Proposed System Concept on Enhancing the Encryption and Decryption Method for Cloud Computing", 2015 17th UKSIM-ASSM International Conference on Modelling and Simulation, vol. 35, October 2016.
[3]. A. Biryukov, D. Dinu, and D. Khovratovich, (2013, March 3) [Online]. Available: https://umbrella.cisco.com/blog/2013/03/06/announcing-sodium-a-new-cryptographic-library.
[4]. David Silver, Suman Jana, Eric Chen, Collin Jackson, and Dan Boneh Password Managers: Attacks and Defenses.
[5]. Marshall Kirk McKusick, George V Neville-Neil, Robert N.M. Watson, "The Design and Implementation of the FreeBSD Operating System.
[6]. O'Reilly and Associates, Inc. Lincoln Stein and Doug MacEachern "Writing Apache Modules with Perl and C"