Cryptanalysis of Secure Hash Password Technique(CSHPT) in Linux

HARSHAVARDHAN METLA[1], VINAY REDDY MALLIDI[1], SAI KIRAN CHINTALAPUDI[1], MADHU VISWANATHAM V[*]

[1]School of Computer Science and Engineering, Vellore Institute of Technology, Vellore, India (e-mail: metlaharsha.vardhan2015@vit.ac.in, chintalapudisai.kiran2015@vit.ac.in, vinayreddy.mallidi2015@vit.ac.in, vmadhuviswanatham@vit.ac.in)
* Corresponding author

ABSTRACT
The very basic aspect when creating user account systems is to provide protection of data and other details. Hence, there is a need to protect the passwords so that even when the hacker steals the database, the user's passwords are secured using various hashing algorithms. Cryptographic hashing functions like MD5, SHA-216 etc. can be easily hacked with powerful hardware on the hacker’s side. Moreover, these are not slow functions, there is a need for implementation of slow hashing functions along with salt or pepper added, which can withstand the growing technology utilized by the attacker. Generally, these functions are used in Linux/Unix password database for authentication of users and also for other security purposes. The slow hashing functions implemented in this paper are PBKDF2, BCrypt, Scrypt and for cryptanalysis, we have used known-plaintext attack because all the brute-force, dictionary attacks become useless in case of slow hashing functions. So, we are going to implement and analyze the performance of different algorithms and also make comparisons among them.

Index Terms: Cryptanalysis, Hashes, Known plain-text, Linux, Pepper, Performance, Salt, Slow hash functions.

I. INTRODUCTION

Passwords are hashed to provide further security as a secondary defense system. Passwords are validated by making use of the data present in the server for the purpose of authentication. Validation can be thought of as a simpler process of comparison in which systems have the password details stored in them. But if the external person is facilitated with the option to view the file contents or tables of a database that contains the passwords- the system is prone to attack by the attacker. Usually, there are chances of such breaches of read-only and partial nature occurring frequently. Attack by SQL injection is one such case. Offline dictionary attacks can be performed by the attacker if he has details of the server of read-only type. This is because just the server's contents can be used for validation purposes. The attacker keeps on trying all passwords which are potent enough,until he finds a match, and this cannot be avoided. So, there is a need to avoid the attacks by making it hard for the attackers to find a match.

Hash functions are cryptographic in nature. These hash functions are interesting objects that are mathematical in nature. They can be computed easily by everyone but even then, they are difficult and can hardly be inverted by the attackers. This tool works well for our issue- the server is used to store the password's hash. When it is presented with a password that is putative- the server needs to simply hash in order to provide the same value. In this way, the password is not revealed even though the hash is known. Hence there is a need to try with some better and efficient methods. It can be made possible by using salting and slow hashing functions together. It is a tough task to design a hash function- if the result should be more secure. Even then there is a need to depend on standard construction methods.

II. BACKGROUND

A. Storing Passwords in Linux

The main mechanism that is used to access a machine running on Linux consists of account of user with the password that corresponds to it. Passwords corresponding to all the user accounts in the systems need to be stored in a database/file to enable verification during the login attempts by the user. But as we know the file/database that has the passwords is present as hash values in a format that is encoded. Therefore, it is difficult to know the real password from the hashed password that is encoded. Dictionary attacks can be applied to the encoded password to obtain the real password. Same hash cannot be encoded to two data at the same time. Even if a single character in the data is changed, an entirely different hash is produced. The hashing strength is used in many ways in almost every protocol of communication. It is used to check and confirm the data’s originality and integrity. There are chances that attackers can get the hash that is encrypted and try several combinations using the word-thereby producing the same hash. Due to the computing advances –the attacker can check with many combinations within a short span of time.

The passwords that are stored in LINUX Systems are also prone to these types of risks. The attacker can find out the file that contains passwords and break the passwords even if encoding was done on the passwords previously. Passwords in Unix are present in /etc/password. This file has word readable access, that is, every user of the system will be able to read the password file. This has a purpose, i.e the password file has other critical information in addition to passwords. Several system tools and applications need this information to function correctly. There is a need to keep the passwords in a separate file that is accessible only by the root. This is implemented by “shadow-utils” package in Linux [2]. Shadow utils package separates passwords for /etc/passwd. Then the passwords are saved in /etc/shadow file- which is accessible only by the root. The read permission for /etc/shadow file is given only to the root user.

B. Parallel Concept

Parallelism is one of the advantages that is possible by the attacker, over the defender. The attacker gets the hashed passwords list. He is interested to break all the passwords that are vulnerable to be broken. Several attacks are tried by him in parallel.

For example, one password which is potential is considered by the attacker. The password is hashed, then this password is compared with other 100 passwords that are hashed. The hashing cost for many passwords that are attacked is shared by the attacker.

An optimization method that is similar to this are the tables which are precomputed-such as rainbow tables. This is also considered parallelism which is still, and has a change of coordinates with space-time. All the attacks have a common characteristic called parallelism. Parallelism can be used on several passwords. They are processed with the same hash function. Several hash functions are used in Salting as against to using a single hash function. Own hashing function should be used in each instance of hashing a password. From amongst a family of hash functions, salting is a method to choose a particular hash function. Salts that are applied properly will completely prevent the attacks that are parallel-inclusive of rainbow tables [3].

C. Slowness

Faster computers are being produced over time. Hence, they are becoming fast over time. But the human brains’ fastness is fixed. Users can remember simple passwords but it is difficult for them to remember very complex passwords even though several potential passwords can be tried by them. To act as a check for this trend-inherent hashing that is slow can be made use of to define the hashing function. These hashing functions that are inherently slow use many internal iterations [4].

MDA and family of SHA are a few standard hashing functions that are cryptographic. It is difficult to build a secure hash function that has very less/no operations that are elementary in nature. Competitions are held by the cryptographers to test the efficiency of hashing functions so that the most efficient algorithm is the one that cannot be broken and that no one knows how it can be broken.

A hash function is inappropriate for hashing of the passwords. This is because:

- The password is unsalted. Therefore, it is prone to attacks by parallelism (such as MD5 and SHA-1 can be attacked by making use of rainbow tables).
- Hashing rate is very fast- it keeps increasing with advances in technology.

D. Slow Hash Functions

PBKDF2 comes from PKCS#5. It is parameterized with associated iteration count (a whole number, at least 1, no higher limit), a salt (a sequence of bytes, no constraint on length), an output length (PBKDF2 will generate Associate in Nursing output of configurable length), Associate - "underlying PRF". In application, PBKDF2 is usually used with HMAC, that is itself a construction designed over underlying hash operation. Therefore, we say "PBKDF2 with SHA-1", we tend to truly mean "PBKDF2 with HMAC with SHA-1"[1].

Bcrypt was designed by reusing and increasing components of a block cipher. The iteration count may be a power of 2, that may be a small indefinite quantity less configurable than PBKDF2. This is often the core parole hashing mechanism within the OpenBSD software system.

Scrypt, over PBKDF2 and a stream cipher known as Salsa20/8, however these measures the tools around the core strength of scrypt, that is RAM. scrypt has been designed to inherently use plenty of RAM (it generates some pseudo random bytes, then repeatedly scan them in an exceedingly pseudo-random sequence). "Lots of RAM" are some things that are difficult to form parallel. A basic laptop is nice at RAM access, and cannot try and scan dozens of unrelated RAM bytes at the same time. An associate offender with a GPU or a FPGA can wish to try and do that and can realize it troublesome.

III. Implementation

We have implemented PBKDF2, Bcrypt, and Scrypt using java in eclipse platform. The reason to choose Java is that - it is platform independent. First, the user enters the plaintext, this password is hashed by applying the three hashes. The user can know which type of hash is implemented by doing cryptanalysis. Once the user enters a value specified, he can get to know the algorithm used. Upon, hashing the plain-text using three algorithms - Scrypt hash took more time to give the output because it utilizes both time and memory exponentially. The software and hardware used are mentioned in the below tables.

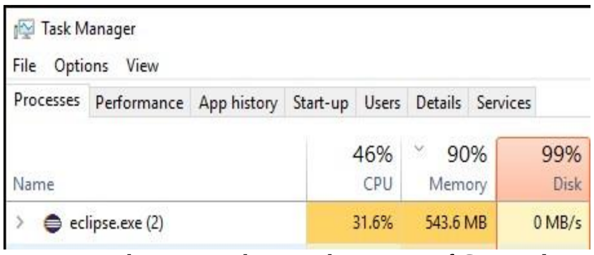
TABLE I. SOFTWARE SPECIFICATION

Type	**Software used**
Operating System	Windows 10, Linux
Language	JAVA 8
IDE	Eclipse, Net Beans

TABLE II. HARDWARE SPECIFICATION

Type	Hardware used
Processor	Intel i5
Ram	4 GB
Clock Speed	2.20 GHz

We have done cryptanalysis based on known plaintext attack. Since these are slow hash functions and are hashed with salt and pepper. We have utilized task manager to calculate the amount of CPU cost and memory used by the different hash algorithm and have plotted a graph. The certain amount of time took for the cryptanalysis was set to 5 minutes.



Task Manager
File Options View
Processes Performance App history Start-up Users Details Services

Name	46% CPU	90% Memory	99% Disk
eclipse.exe (2)	31.6%	543.6 MB	0 MB/s

Figure 1. Task manager showing the amount of CPU and RAM used.

The performance of algorithms is visualized by plotting graphs. The CPU cost was measured in terms of percentage and the RAM utilized was measured in terms of MB. As we can see the CPU vs memory was maxed out in the case of Scrypt and the PC goes to starvation state due to the utilization of all resources. Even the PBKDF2 also went to starvation after a long time. So, we have set the time limit to 5 minutes.

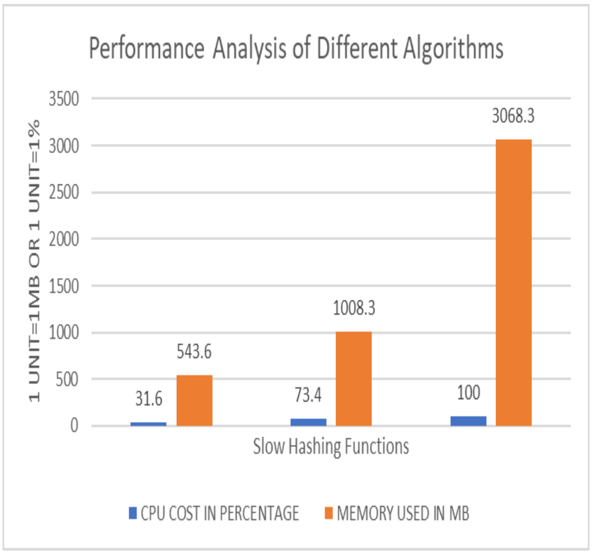


Figure 2. The Algorithms PBKDF2, Bcrypt and Scrypt from left to right. *(Performance Analysis of Different Algorithms).*

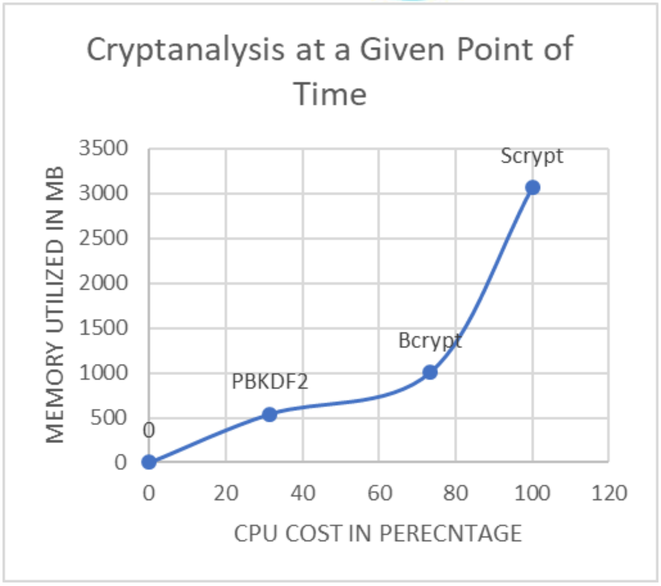


Figure 3. Graph showing the CPU Cost Vs Memory utilized at certain time.

IV. Comparison

A. *Advantages of PBKDF2*

- It has been nominative for an extended time.
- It is already existent in a varied framework (e.g. it's given .NET).
- Extremely configurable(although some implementations don't allow you to opt for the hash operate. For Example, the one in .NET is for SHA-1 only).

B. Drawbacks of PBKDF2

- CPU-intensive solely, to high optimization with GPU (the defender may be a basic server that will generic things, i.e. a PC, however, the assailant will pay his budget on a lot of specialized hardware, which can be provided).
- This will continue to manage the parameters yourself (salt generation and storage, iteration count encryption). There is a typical encryption for PBKDF2 parameters however it uses ASN.1, therefore ,in general, can be avoided. (ASN.1 will be tough to handle in reality) [5].

C. Advantages of bcrypt

- Several implementations in numerous languages added additional resilient to GPU; this is often as a result of details of its internal style. The bcrypt authors created: they reused Blowfish as a result of Blowfish was supported an enclosed RAM table that is continually accessed and changed throughout the process. This makes life a lot of tougher for whoever needs to hurry up bcrypt with a GPU (GPU aren't smart at creating plenty of memory accesses in parallel).
- Normal output secret writing which has the salt, the iteration count and also the output joined easily to

store character string of printable characters.

D. Drawbacks of bcrypt:

- Output size is fixed: 192 bits.
- Whereas bcrypt is nice at thwarting GPU, it will still be completely optimized with FPGA: trendy FPGA chips have plenty of tiny embedded RAM blocks that square measured conveniently for running several bcrypt implementations in parallel among one chip.
- Input size is fifty-one characters. so as to handle longer passwords, one needs to mix bcrypt with a hash perform (you hash, then use the hash price because of the "password" for bcrypt). Combining scientific discipline primitives is understood to be dangerous thus such games cannot be suggested on a general basis.

E. Advantages of scrypt

- A PC, i.e. specifically what the defender can use once hashing passwords, is that the best platform (or shut enough) for computing scrypt. The offender now not gets a lift by outlay his bucks on GPU or FPGA.
- An extra thanks to tune the function: memory size.

F. Drawbacks of scrypt

- Not as a ready-to-use implementation for varied languages.
- Unclear whether or not the processor / RAM combine is perfect. for every of the pseudo-random RAM accesses, scrypt still computes a hash operate. A cache miss is going to be regarding two hundred clock cycles, one SHA-256 invocation is near one thousand. There is also area for improvement here.
- One more parameter to configure: memory size.

TABLE III.

Comparison Table	**Slow Hashing Functions**		
	PBKDF2	***Bcrypt***	***Scrypt***
Memory Hardening Factor	No	Yes	Yes

V. Conclusion And Future Implementation

In our model the user can set his own salt or a default pepper value is generated randomly. We have done cryptanalysis and compared different slow hashing algorithms. We came to a conclusion that Scrypt is the more secured when compared with others. We know that argon2 has won in the PHC but, we didn't mention it because it came very recently i.e., in the year 2015. We also know that cryptographic hashing can be implemented only when no one is able to hack it for a tenure of certain years like bcrypt and scrypt. We will develop our system more user-friendly. When the user logs into the Linux based system, he can set the algorithm by which he wants to encrypt his password. PBKDF2 can be used when the user account has unimportant data like a PC utilized by individual, Bcrypt can be used when there are small organizations etc. i.e., somewhat important data. The military based encryption they can use scrypt. Parallelism plays a major role in attacking the hashed password because each iteration can be divided into threads and are processed through different processors which reduces the time factor, and gives an advantage to the attacker. So, it utilizes exponential time and memory of the attackers Hardware.

References

Journal Articles

[1] SysTEX '16 Proceedings of the 1st Workshop on System Software for Trusted Execution, Article No. 1, Trento, Italy — December 12 - 16, 2016, ACM New York, NY, USA ©2016

Conference Proceedings

[2] J. Zhang and S. Boonkrong, "Dynamic Salt Generating Scheme Using Seeds Warehouse Table Coordinates," *2015 2nd International Conference on Information Science and Security (ICISS)*, Seoul, 2015, pp. 1-6.

Online Sources

[3] Preziuso, M. (2015, June 22). Password Hashing: PBKDF2, Scrypt, Bcrypt – Michele Preziuso – Medium. Retrieved December 25, 2017, from https://medium.com/@mpreziuso/password-hashing-pbkdf2scrypt-bcrypt-1ef4bb9c19b3

[4] How are passwords stored in Linux (Understanding hashing with shadow utils). (n.d.). Retrieved December 25, 2017, from https://www.slashroot.in/how-are-passwords-stored-linuxunderstanding-hashing-shadow-utils

[5] How to securely hash passwords? (n.d.). Retrieved December 25, 2017, from https://security.stackexchange.com/questions/211/how-to-securelyhash-passwords?answertab=active#tab-top