Journal of Korea Multimedia Society Vol. 24, No. 1, January 2021(pp. 58-66)
https://doi.org/10.9717/kmms.2020.24.1.058

접속로그와 패스워드에 따른 가변 및 동적솔트에 관한 연구

정진호†, 차영욱††, 김춘희†††

A Study on the Variable and Dynamic Salt According to Access Log and Password

Jinho Jeong†, Youngwook Cha††, Choonhee Kim†††

ABSTRACT

The user's password must be encrypted one-way through the hash function and stored in the database. Widely used hash functions such as MD5 and SHA-1 have also been found to have vulnerabilities, and hash functions that are considered safe can also have vulnerabilities over time. Salt enhances password security by adding it before or after the password before putting it to the hash function. In the case of the existing Salt, even if it is randomly assigned to each user, once it is assigned, it is a fixed value in a specific column of the database. If the database is exposed to an attacker, it poses a great threat to password cracking. In this paper, we suggest variable-dynamic Salt that dynamically changes according to the user's password during the login process. The variable-dynamic Salt can further enhance password security during login process by making it difficult to know what the Salt is, even if the database or source code is exposed.

Key words: Password Security, Hashing, Access Log, Salt

1. 서 론

개인정보보호법의 개인정보 안전성 확보조치 기준 제 7조 2항에 따라 웹사이트의 패스워드는 복호화되지 않도록 일방향 암호화하여 저장하여야 한다[1]. KISA의 개인정보 암호화 조치 안내서에서는 일방향 암호화를 해쉬함수를 이용한 암호화로 정의하고 있으며, KISA와 NIST 등 국내·외 보안관련 기관에서 사용을 권고하는 해쉬함수를 소개하고 있다[2]. 컴퓨터의 성능이 좋아짐에 따라, 현재 많이 사용되고 있는 MD5와 SHA-1 해쉬함수의 충돌이 발견되었으며, 무차별공격과 사전공격(dictionary attack)에 의해 해쉬값이 복호화되었다[3,4]. 이처럼 컴퓨터의 성능이 계속 좋아지면 안전하다고 지정된 해쉬함수들도 더 이상 안전하지 않을 수 있게 된다.

패스워드 해쉬값의 안전성을 강화하기 위한 방법으로 제시된 솔트(Salt)는 회원가입 시에 사용자마다 랜덤하게 부여되며, 생성된 솔트는 패스워드의 앞이나 뒤에 붙여져 패스워드 해쉬값을 생성한다[5]. 사용자마다 랜덤하게 부여되는 솔트라 하더라도, 한번 생성된 후에는 데이터베이스에 고정값으로 저장된다. 데이터베이스가 해킹되는 경우에 공격자는 각 사

※ Corresponding Author : Youngwook Cha, Address: (36728) 375, Gyeongdong-ro Andong-si Gyeongsang-buk-do, Republic of Korea, TEL : +82-54-820-5714, FAX : +82-54-820-6164, E-mail : ywcha@anu.ac.kr
Receipt date : Oct. 13, 2020, Revision date : Nov. 23, 2020
Approval date : Nov. 24, 2020
† DJ FAMILY (E-mail : jgw2846@naver.com)
†† Dept. of Computer Engineering, Andong National University
††† Dept. of Electronic&Information Communication Engineering, Daegu Cyber University
(E-mail : chkim@dcu.ac.kr)
※ This work was supported by a Research Grant of Andong National University LINC+.

용자에게 부여된 솔트를 쉽게 파악하여 패스워드 크래킹에 이용할 수 있는 취약점이 있다.

기존의 고정 솔트값 기반 패스워드 해쉬값에 대한 취약성을 보완하기 위하여 가변의 접속로그들로 구성되는 솔트조합이 패스워드에 따라 동적으로 변하는 방안을 제안한다. 본 논문에서 제안하는 가변 및 동적 솔트 방안은 데이터베이스나 소스코드가 노출되더라도 공격자가 솔트조합과 솔트값이 무엇인지 파악하기 어려우므로 패스워드 해쉬값을 더 안전하게 보호할 수 있게 된다.

본 논문의 2장에서는 관련 연구동향을 기술하며, 3장에서는 접속로그와 패스워드에 따른 가변-동적 솔트 방안을 제안한다. 4장에서는 제안한 솔트의 구현과 시험에 대하여 기술하며, 5장에서는 결론과 추후 연구계획에 대하여 기술한다.

2. 관련연구

2.1 해쉬함수와 패스워드 길이

해쉬함수는 임의의 길이를 갖는 메시지를 입력으로 하여 고정 길이의 해쉬값을 생성하는 함수이다. 웹사이트에서는 패스워드를 해쉬함수에 입력하여 생성한 해쉬값을 데이터베이스에 저장한다. 사용자가 접속 시에 입력한 패스워드에서 생성한 해쉬값과 기존 데이터베이스에 저장되어 있는 해쉬값을 비교하여 같으면 로그인 할 수 있고, 다르면 로그인이 거부되는 방식으로 사용한다. 해쉬함수의 요구사항은 일방향 함수와 강한 충돌회피이다. 컴퓨터의 성능 향상과 더불어 MD5와 SHA-1은 충돌이 발견되어 사용을 권고하지 않는 함수가 되었다.

중국의 Wang 교수와 그 동료들은 2004년에 MD5의 충돌을 발견하였으며[3], 구글 연구팀은 2017년에 SHA-1의 충돌을 발견하였다[4]. SHA-1은 1993년 NIST 에 의해 안전한 표준 해쉬함수로 지정되어 사용되었으나 충돌이 발견됨에 따라 더 이상 안전하지 않은 해쉬함수가 되었다. 해쉬함수의 무차별공격은 복호화 하려는 메시지에 대하여 미리 계산한 해쉬값과 네트워크나 시스템에서 획득한 해쉬값이 일치하면 메시지를 찾아내는 것이다. 다른 해쉬함수에 비해 월등히 빠른 계산속도로 인하여 널리 사용되고 있는 MD5의 해쉬값에 대하여, php.net은 무차별 공격으로 쉽게 복호화될 수 있다고 경고하였다[6].

LAUGHFOOL's LAB에서는 MD5 CrackFAST 툴을 이용하여 무차별 공격으로 MD5 복호화 시험을 수행하였다[7]. 패스워드가 알파벳 대문자와 소문자 그리고 숫자의 조합이라 가정하면 n자리 패스워드의 경우에 가능한 패스워드의 수는 62의 n제곱이 된다. 5자리 패스워드의 경우에 가능한 패스워드의 수는 916,132,832(=625) 이다. 가정용 PC(i5 2.67GHz CPU, Thread count 3)를 이용하여 5자리 부터 12자리의 패스워드를 찾는데 걸린 시간을 측정하였다. 5자리 패스워드의 복호화 시간은 6.287초이며, 11자리의 복호화에는 약 360,140일이 소요되었다. Securityledger 사에서는 25개의 virtual AMD GPU 스펙으로 초당 1800억개의 MD5 해쉬값을 생성할 수 있는 패스워드 크래킹 PC를 소개하였다[8]. Securityledger사의 크래킹 PC는 가정용 PC에서 동작하는 MD5 Crack Fast 툴에 비하여 약 2400배 빠르게 해쉬값을 생성할 수 있는 성능이다.

Fig. 1은 2017년에 Statista가 조사한 전세계 사용자의 유출된 패스워드 평균 글자 수를 나타낸다. 자료에 의하면 유출된 약 3억 2천만개의 패스워드 중 가장 많이 사용된 것은 7~8자리 패스워드이었다[9]. 10자리 패스워드의 MD5 해쉬값은 Securityledger사의 크래킹 PC에 의해 약 2.5일만에 크래킹되며, 가장 많이 사용되는 8자리의 경우는 1분도 안되어 크래킹 될 수 있다. 그러나 11자리의 경우에 150일 이상 소요되어 무차별 공격을 통한 크래킹이 쉽지 않다는 것을 알 수 있다. 길이가 길수록 패스워드가 안전할 수 있지만 편리함 때문에 실제로는 7~10자리의 패스워드를 대부분 사용하고 있다.

2.2 솔트와 고정솔트 방식의 취약성

솔트는 Fig. 2와 같이 패스워드의 앞이나 뒤에 추가되어 패스워드 해쉬값을 생성하는데 사용된다[10]. 패스워드에 솔트가 추가되면 해쉬함수의 입력 메시지가 길어져 더 많은 크래킹 시간이 요구되므로 패스워드를 보다 안전하게 보호할 수 있는 방법이다. Pritesh[5]는 솔트를 사용한 패스워드는 사전공격을 줄일 수 있고, 크래킹이 어려울 뿐 아니라 SQL Injection 공격까지 막을 수 있음을 보이고 있다.

그러나 Pritesh[5]에서 제시한 솔트는 Fig. 3과 같이 데이터베이스의 회원테이블에 하나의 컬럼으로 저장된다. 사용자가 접속 시에 입력한 패스워드에 솔

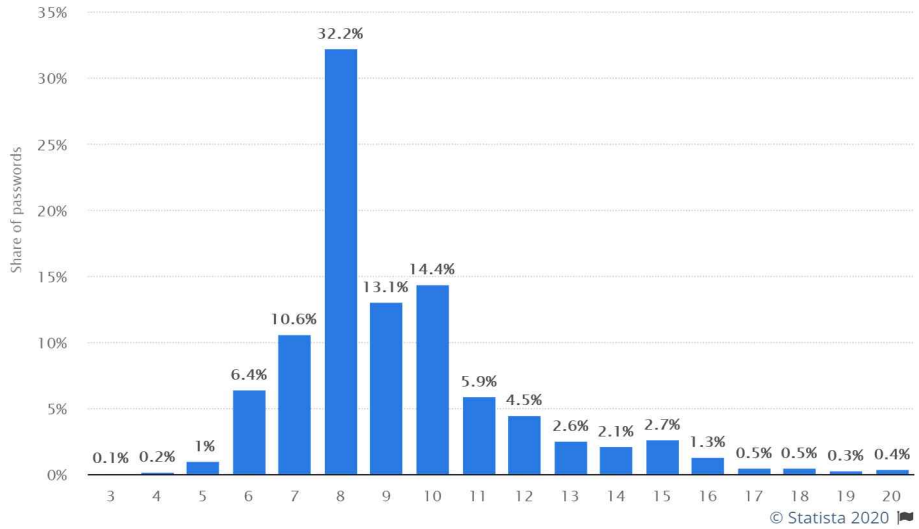

Fig. 1. Average number of characters in leaked user passwords worldwide in 2017.

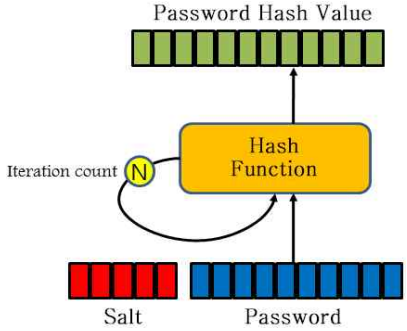

Fig. 2. Generation of password hash value using Salt.

트 컬럼의 값을 추가하고 해쉬함수에 입력하여 생성한 해쉬값과 데이터베이스에 저장되어 있던 패스워드 해쉬값과의 일치여부로 로그인 성공과 실패를 판정한다. 솔트는 사용자마다 랜덤하게 생성되더라도, 생성된 후에는 특정 값으로 고정되어 데이터베이스의 솔트 컬럼에 저장된다. 따라서 데이터베이스가 해킹되는 경우에 공격자는 각 사용자 별로 부여된 솔트를 쉽게 파악하여 패스워드 크래킹에 이용하게 된다. 솔트는 대부분 패스워드의 앞이나 뒤에 붙여 사용되므로, 공격자 입장에서는 솔트를 단순히 패스워드 앞 또는 뒤에 붙이는 경우에 비하여 무차별 공격 회수가 2배 정도 늘어나는 것에 불과하다.

삼성전자의 특허에서 제시한 솔트는 입력된 패스워드에 솔트를 부가하여 조합 패스워드를 생성하고 인증정보를 생성한다. 또한 난수발생기로부터 생성된 솔트를 데이터베이스에 저장하고 인증 시 입력된 패스워드와 함께 조합하여 인증한다[11]. 삼성전자의 특허로 제시된 방법도 솔트가 고정값으로 저장되므로, 솔트가 노출되면 크래킹에 취약한 문제점이 있다.

솔트를 안전하게 하기 위한 노력으로 2016년 Sirapat[12]에 의해 제시된 솔트방법과 2018년 Dr. Abdelrahman[13]에 의해 제시된 솔트방법은 Pritesh와 같이 사용자마다 랜덤하게 솔트를 생성한다. 그러나 Sirapat은 그 솔트를 패스워드의 글자 중간 중간에 삽입하며, Dr.Abdelrahman가 제시한 솔트는 솔트 글자를 재배열하여 패스워드에 붙여 패스워드 해쉬값을 생성해낸다. 이 방법들 또한 패스워드의 솔트가 데이터베이스에 고정값으로 저장되기 때문에 데이터베이스가 공격자에게 노출될 시 그에 따라 솔트도 노출된다. Pritesh에 비해 솔트의 삽입위치가 중간이거나 솔트 글자 배열이 다름에 있어 보다 안전할 수는 있으나, 솔트와 같이 패스워드 크래킹에 직접적인 영향을 주는 중요 데이터가 노출되는 것은

id	account	password	salt
1	admin	e12afeb6153c0e4defeaf140d5098486649616d140dff51685...	6e2ae5ee54e66af2f4d7da0e58d08dfb687504c87071affw11...

Fig. 3. Pritesh Salt.

완전히 안전하다고 볼 수 없다. 만약 소스코드가 노출된다면 솔트의 배열 규칙뿐만 아니라 패스워드와 솔트의 조합 위치까지 알 수 있게 되어 크래킹에 위협이 될 수 있다.

3. 접속로그와 패스워드에 따른 가변-동적 솔트

본 절에서는 고정 솔트값이 사용되는 기존 패스워드 해쉬값의 보안성을 강화하기 위하여 하나 이상의 접속로그를 사용하는 패스워드 별 가변-동적 솔트 방안을 제시한다. 접속로그 기반의 솔트는 로그인 체크 시에 솔트값은 변경되나 유형은 정적인 방식이며, 가변-동적 솔트는 패스워드 별 접속로그 기반의 솔트값과 유형이 매번 변하는 방식이다.

3.1 접속로그 기반의 가변-정적 솔트

국내·외 개인정보보호 정책에 따라 웹사이트의 개인정보 처리자는 방문자 접속기록 (IP주소, 접속날짜, 브라우저와 OS종류 등)을 반드시 데이터베이스에 저장하여야 한다[1,14]. IP주소 및 최근 로그인 시간과 같은 접속 기록들은 로그인 시에 매번 변경되므로 이것들을 솔트로 정하면 개인정보보호 정책도 따르면서, Shannon에 의해 고안된 One-Time Pad[15]의 완벽한 랜덤까지는 아니더라도 패스워드 해쉬값을 자주 바꿀 수 있게 된다. 즉, 접속로그 기반의 솔트는 로그인할 때마다 해쉬값이 변경되므로 데이터베이스 관리자나 공격자 입장에서는 사용자가 패스워드를 자주 변경하는 것과 같은 효과를 보이게 된다.

Fig. 4는 가변의 접속로그를 활용한 정적솔트를 PHP 7.3, MYSQL 5.0 환경에서 구현하고, PHPMYADMIN으로 시각화한 것이다. 이처럼 접속로그를 솔트로 사용하게 되면 데이터베이스가 노출되더라도 솔트가 무엇인지 알 수 없을 뿐만 아니라 솔트를 사용하고 있는지 조차 알 수 없게 된다. 또한, 공격자가 패스워드의 크래킹 중에 사용자가 로그인을 하면 해쉬값이 변경되므로 크래킹을 새로 시도하여야 한다. 하지만 접속로그 기반의 정적솔트 방식에서는 로그인과 관련된 소스코드가 해킹된다면 사용된 솔트가 무엇인지 노출되는 취약점이 있다.

3.2 패스워드 별 접속로그 기반의 가변-동적 솔트

소스코드가 노출되더라도 패스워드를 안전하게 보호하기 위한 방안으로 패스워드 별로 가변의 접속로그가 동적으로 선택되는 솔트를 제안한다. 가변-동적 솔트 방식에서는 솔트의 조합과 패스워드의 숫자화를 이용한다. 접속로그들의 조합이 솔트의 조합이며 N개의 접속로그를 솔트로 사용하는 경우에 최대 가능한 솔트조합의 수는 N의 계승(factorial)이 된다.

Fig. 5는 패스워드를 숫자화한 값에서 계산한 변수 i, j, k를 이용하여 솔트조합이 패스워드에 추가되는 위치를 나타낸다. 사용자가 입력한 패스워드는 사용자만 알고 있으므로 패스워드를 숫자화한 값 또한 아무도 알 수 없게 된다. Fig. 5에서 j는 솔트조합의 개수(N!)를 모듈러 연산하여 나온 값으로, j번째의 솔트조합을 선택하는 변수이다. 변수 i는 패스워드의 숫자화된 값을 패스워드 길이로 모듈러 연산하여 나온 값이며, 패스워드 내의 i번째 위치에 j번째의 접속로그 솔트조합을 삽입한다. 마지막으로 k는 패스워

```
$password = md5($latest.$password);  // various combination
$password = md5("2020-07-06 21:25:51anu13579");  // 27 characters
```

id	account	password	datetime	latest	login_count	ip	active
1	admin	fd594461833efabfcd041f36f8be3930	2019-05-05 10:24:10	2020-07-06 21:25:51	2	211.124.59.105	y

id	account	password	datetime	latest	login_count	ip	active
1	admin	93f7352fa6a28db545513bfb9b9a4c85	2019-05-05 10:24:10	2020-07-07 09:15:32	3	211.124.59.105	y

id	account	password	datetime	latest	login_count	ip	active
1	admin	341d373e97775abb582c72934ae6d29f	2019-05-05 10:24:10	2020-07-07 15:11:13	4	211.124.59.105	y

id	account	password	datetime	latest	login_count	ip	active
1	admin	853be92c741493a8c7afcef3fe8b4a61	2019-05-05 10:24:10	2020-07-08 12:15:29	5	119.202.188.81	y

Fig. 4. Access log-based static Salt.

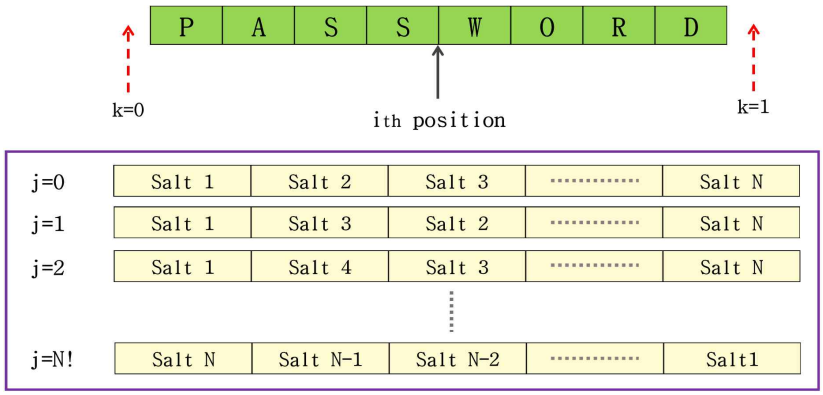

Fig. 5. Salt combination locations in password in case of variable-dynamic Salt.

드의 숫자화된 값을 2로 모듈러 연산하여 나온 값이며, k가 0이면 솔트조합을 패스워드의 앞에, 1이면 뒤에 추가한다.

패스워드에 따른 접속로그 기반의 로그인 인증 절차는 Fig. 6과 같다. 패스워드와 변수 i, j, k의 위치에 추가되는 솔트조합을 이용하여 패스워드 해쉬값을 생성한 후 데이터베이스에 저장된 해당 회원의 해쉬값과 비교하여 일치하면 로그인에 성공한다. 로그인에 성공하면 최신 접속로그를 이용한 동적솔트 매커니즘으로 패스워드 해쉬값을 새로 생성한 후 다음 로그인 시의 체크를 위하여 데이터베이스에 저장한다.

4. 가변-동적 솔트의 구현과 시험

Fig. 7은 가변-동적 솔트 메커니즘을 구현한 PHP 코드로 Fig. 5에서 제시한 변수 i, j, k를 이용하였다. 코드에서 3개의 접속로그를 솔트로 사용하므로 솔트조합의 개수는 3! 인 6개의 솔트조합이 사용될 수 있다. PHP코드에서 함수 unpack의 첫번째 인자인 I는 문자열(string)을 비부호 정수(unsigned integer)로 바꾼다는 의미이다. 코드에서는 j번째의 솔트조합이 사용자가 입력한 패스워드의 i와 k의 위치 2곳에 삽입된 후 해쉬값이 생성된다. 접속로그 기반의 패스워드 별 가변-동적 솔트에서는 소스코드가 노출되더라도 공격자는 최대 (패스워드 길이+1) * N!(솔트조합 개수) * 2(앞 혹은 뒤) 만큼의 크래킹을 시도하여야 한다.

Fig. 8은 웹에서의 로그인 체크 시에 패스워드에 따라 접속로그의 조합이 선택되는 동적솔트와 일정한 조합이 사용되는 정적솔트를 사용하는 경우에 패스워드와 솔트조합의 MD5 해쉬값 생성시간을 나타낸다. 시험을 위한 코드는 최신접속 시간만을 솔트로 사용하여 PHP로 구현하였으며, MYSQL 데이터베

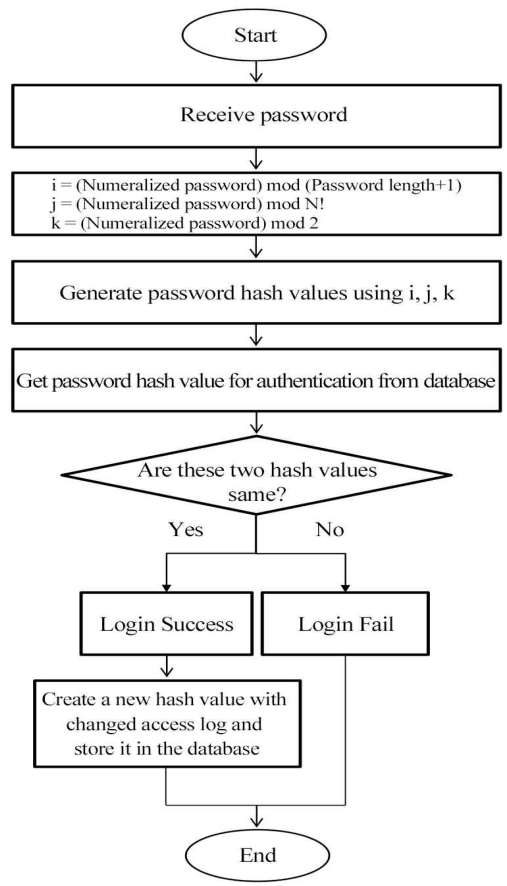

Fig. 6. Login authentication procedure using variable-dynamic Salt.

```php
$password_length = strlen($password);
$combination_count = 6;

$i = unpack("I", $password)[1] % ($password_length + 1);
$j = unpack("I", $password)[1] % $combination_count;
$k = unpack("I", $password)[1] % 2;

if($j == 0)
    $combination = $login_latest.$login_ip.$login_count;
else if($j == 1)
    $combination = $login_latest.$login_count.$login_ip;
else if($j == 2)
    $combination = $login_count.$login_latest.$login_ip;
else if($j == 3)
    $combination = $login_count.$login_ip.$login_latest;
else if($j == 4)
    $combination = $login_ip.$login_latest.$login_count;
else if($j == 5)
    $combination = $login_ip.$login_count.$login_latest;

if($k == 0)
    $hash = md5($combination.substr_replace($password, $combination, $i, 0));
else
    $hash = md5(substr_replace($password, $combination, $i, 0).$combination);
```

Fig. 7. Implementation of variable-dynamic Salt using PHP code.

```
password : anu13579
Salt : 2020-07-06 21:25:51
Text : 2020-07-06 21:25:51anu13579
Hash with MD5 : fd594461833efabfcd041f36f8be3930
Spent time : microtime(0.048259019851685), milliseconds(48)
```

(a) Access log-based static Salt

```
password : anu13579
i : 1
j : 1
k : 1
combination : 1.1.1.12020-07-06 21:25:511188
INPUT : a1.1.1.12020-07-06 21:25:511188nu135791.1.1.12020-07-06 21:25:511188
Hash with MD5 : cf3d199d099f4381956a56d7ffba434e
Spent time : microtime(0.049512147903442), milliseconds(50)
```

(b) Access log-based dynamic Salt

Fig. 8. Performance comparison of variable-static and variable-dynamic Salts.

이스와 3.40GHz CPU 환경에서 50회 측정한 생성시간의 평균을 계산하였다. 정적솔트에 비하여 동적솔트에는 패스워드를 숫자화하고 모듈러 연산을 수행하여 j번째 솔트조합을 선택하며, 패스워드의 i번째와 k번째 위치에 솔트조합을 삽입하는 기능이 추가되었다.

Table 1은 Pritesh에서 제시한 솔트방식과 본 논문에서 제안한 솔트방식을 솔트의 값과 유형, 취약성, 해쉬값의 생성시간, 그리고 요구되는 최대 공격횟수를 비교하였다. Pritesh의 솔트와 접속로그 기반의 정적솔트에 대한 패스워드 해쉬값의 생성시간은 솔트의 개수를 1개라 가정하였을 때 동일하며, 시험에서 측정된 생성시간은 48 ms이었다. 서로 다른 3가지의 접속로그를 솔트로 사용하는 동적솔트 방식에서의 생성시간은 50 ms로 측정되었다.

동적솔트에 여러 기능이 추가되었지만 해쉬값의 생성시간은 Pritesh의 정적솔트에 비하여 약 4%인 2 ms 정도만이 증가되었다. 정적솔트에서는 공격자

Table 1. Comparison of Salt methods.

	Pritesh Salt	Variable-static Salt	Variable-dynamic Salt
Salt Value	Fixed	Variable	Variable
Type	Static	Static	Dynamic
Vulnerability	Vulnerable in DB exposure	Vulnerable in DB and source code exposure	Safe in DB and source code exposure
Hash generation time (Salt count = N)	48 ms (N=1)	48 ms (N=1)	50 ms (N=3)
Maximum number of attacks	2	2	(Password length+1) * N! * 2

가 소스코드에 노출된 한 개의 정해진 솔트조합으로 크래킹을 시도한다. 그러나 동적솔트에서는 소스코드가 노출되더라 N개의 솔트를 사용하는 경우에 N! 개의 솔트조합들에 대하여 평균 N!/2 번 이상, 그리고 최대 (Password length+1) * N! * 2 만큼의 크래킹을 시도하여야 한다. 솔트의 개수가 5개이고 패스워드 길이가 8인 경우의 동적솔트에서는 최대 2160 (=9*120*2) 번의 크래킹을 시도하여야 한다. 요구되는 크래킹 횟수가 2인 가변-정적 솔트 방식에 비하여 월등히 많은 크래킹 횟수가 요구되는 가변-동적 솔트 방식에서 패스워드 해쉬값이 더욱 안전하게 보호될 수 있음을 확인할 수 있다.

패스워드가 얼마의 시간 만에 크래킹 되는지 알려주는 웹사이트인 howsecureismypassword.net[16]를 이용하여 평문 패스워드와 정적 및 동적 솔트 방식에서 생성된 패스워드와 솔트조합의 크래킹 시간을 비교하였다. 평문 패스워드 “anu13579”의 크래킹 예상 시간은 1분이었으며, 접속시간을 이용하는 정적솔트 매커니즘에서 솔트와 패스워드의 결합인 “2020-07-06 21:25:51anu13579”는 “744 OCTILLION YEARS”라는 큰 시간이 나왔다. 동적솔트 매커니즘이 적용된 패스워드와 솔트조합의 예인 “a.1.1.1.12020-07-06 21:25:511188nu135791.1.1.12020-07-06 21:25:511188”는 “47 TRESTRIGINTILLION YEARS” 라는 셀 수 없을 정도의 크래킹 예상 시간이 나왔다. 데이터베이스나 소스코드가 노출될 경우에 가변-정적 솔트 방식에서 패스워드 “anu13579”의 크래킹 예상시간은 솔트가 적용되지 않은 경우와 유사하게 1분으로 예상이 된다. 그러나 가변-동적 솔트 방식에서의 평균 예상시간은 (1분) * (패스워드길이 + 1) * (N!) * 2 / 2 이므로, 솔트 조합의 개수와 패스워드의 길이에 따라 급격히 크래킹 요구 시간이 증가하게 된다.

5. 결 론

본 논문에서는 공격자에게 데이터베이스와 소스코드가 노출되더라도 패스워드 해쉬값을 안전하게 보호할 수 있도록 접속로그 기반의 패스워드 별 가변-동적 솔트를 제안하였다. 접속기록을 솔트로 이용하면 공격자로부터 데이터베이스가 노출되더라도 솔트가 무엇인지 알 수 없게 된다. 또한 로그인 할때마다 솔트값이 변하므로 패스워드 해쉬값도 같이 변하게 되어 패스워드를 안전하게 보호할 수 있다. 동적솔트는 사용자가 입력한 패스워드를 숫자화한 후 모듈러 연산으로 솔트조합을 선택하고, 선택된 솔트조합과 패스워드를 결합하여 해쉬값을 생성한다. 동적솔트 방식은 어떠한 해쉬함수와도 결합될 수 있으며, 사용하는 솔트의 개수에 따라 해쉬함수의 입력길이가 늘어난다.

데이터베이스나 로그인 관련 소스코드가 유출되어도 패스워드 별로 결정되는 동적인 솔트 조합의 유추가 어려우므로, 공격자는 모든 경우의 패스워드와 솔트 조합을 이용하여 크래킹을 시도하여야 한다. 솔트의 개수가 5개이고 패스워드 길이가 8이며, 데이터베이스와 소스코드 노출 시에, 정적솔트에서는 한 번의 크래킹 시도가 요구되지만 동적솔트에서는 평균 1080(=9*120*2/2)번의 크래킹을 시도하여야 한다. 가변-동적솔트는 가변-정적솔트에 비하여 패스워드 보안이 훨씬 강화되지만 패스워드 해쉬값 생성 시간은 약 4% (2ms) 정도만의 추가적인 시간이 요구되었다.

패스워드를 보다 안전하게 보호하는 방법은 로그인과 상관없이 패스워드 해쉬값을 지속적으로 변화

(a) (b) (c)

Fig. 9. Comparison of password cracking times. (a) Plain text password, (b) Variable-static Salt, and (c) Variable-dynamic Salt.

시키는 것이다. Shannon이 고안한 One-time Pad의 완벽한 랜덤에 근접할 정도의 성능을 가지면서, 사용자와 개발자가 사용하기 편리한 패스워드 보호방법을 추후 연구할 계획이다. 그리하여 컴퓨터 성능의 발전이 개인정보 유출에 악영향을 끼치지 않는 사이버 세계에 기여하고자 한다.

REFERENCE

[1] The Ministry of Public Administration and Security (Personal Information Protection Policy), "Basic Measures for Securing Safety of Personal Information," 2019.

[2] Korea Internet & Security Agency (KISA), "Personal Data Encryption Action Guide," 2019.

[3] X. Wang, D. Feng, X. Lai, and H. Yu, "Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD," *Cryptology ePrint Archive, Report 2004/199*, pp. 1-4, 2004.

[4] M. Stevens, E. Bursztein, P. Karpman, A. Albertini, and Y. Markov, "The First Collision for Full SHA-1," *CRYPTO 2017*, pp. 1-23, 2017.

[5] P.N. Patel, J.K. Patel and P.V. Virparia, "A Cryptography Application using Salt Hash Technique," *International Journal of Application or Innovation in Engineering & Management (IJAIEM)*, Vol. 2, Issue 6, 2013.

[6] PHP.net. https://www.php.net/manual/en/function.md5.php (accessed August 17, 2020).

[7] LAUGHFOOL's LAB. https://laughfool.tistory.com/16 (accessed August 17, 2020).

[8] The security ledger. https://securityledger.com/2012/12/new-25-gpu-monster-devours-passwords-in-seconds/ (accessed August 17, 2020).

[9] Statista. https://www.statista.com/statistics/744216/worldwide-distribution-of-password-length/ (accessed August 17, 2020).

[10] Safe Saving Password. https://d2.naver.com/helloworld/318732 (accessed August 17, 2020).

[11] Samsung Electronics Co. Ltd., *User Device Performing Password Based Authentication and Password Registration and Authentication Method Thereof*, 10-2014-0024427, Korea, 2007.

[12] S. Boonkrong and C. Somboonpattanakit, "Dynamic Salt Generation and Placement for Secure Password Storing," *IAENG International Journal of Computer Science*, Vol. 43, pp. 27 - 36, 2016.

[13] A. Karrar, T. Almutiri, S. Algrafi, N. Alalwi, and A. Alharbi, "Enhancing Salted Password Hashing Technique Using Swapping Elements in an Array Algorithm," *International Journal of Computer Science and Technology*, Vol. 9, Issue. 1, pp. 21-25, 2018.

[14] Privacy Statement/Security Notice/Accessibility Statement. https://www.nist.gov/privacy-policy (accessed August 17, 2020).

[15] N. Nagaraj, V. Vaidya, and P.G. Vaidya, "Revisiting the One-Time Pad," *International Journal of Network Security*, Vol. 6, No. 1, pp. 94-102, 2008.

[16] How Secure Is My Password?. https://howsecureismypassword.net (accessed August 17, 2020).

정 진 호

2019년 안동대학교 컴퓨터공학과 (공학사)
2019년~현재 안동대학교 컴퓨터공학과 대학원
2015년~현재 디제이패밀리 대표
관심분야 : 웹보안

차 영 욱

1987년 경북대학교 전자공학과 (공학사)
1992년 충남대학교 전자통계학과 (공학석사)
1998년 경북대학교 컴퓨터공학과 (공학박사)

1987년~1999년 한국전자통신연구원 선임연구원
2003년~2004년 매사추세츠주립대학 방문학자
2018년~현재 경찰청 디지털포렌식 자문위원
1999년~현재 안동대학교 컴퓨터공학과 교수
관심분야 : 망/시스템 제어 및 관리, ICT 및 스마트팜 보안, 개방형통신망

김 춘 희

1988년 전남대학교 전산통계학과 (학사)
1992년 충남대학교 전자계산학과 (이학석사)
2000년 경북대학교 컴퓨터공학과 (공학박사)

1988년~1995년 한국전자통신연구원 연구원
2002년~현재 대구사이버대학교 전자정보통신공학과 교수
관심분야 : 센서 네트워크, 고속통신망, 망 관리 및 제어