Journal of KIIT. Vol. 18, No. 9, pp. 89-97, Sep. 30, 2020. pISSN 1598-8619, eISSN 2093-7571
http://dx.doi.org/10.14801/jkiit.2020.18.9.89

접속로그 기반의 동적솔트를 이용한 패스워드 보안강화

정진호*, 차영욱**

Enhancement of Password Security by using Access Log-based Dynamic Salt

Jin-Ho Jeong*, Young-Wook Cha**

이 논문은 안동대학교 기본 연구지원 사업에 의하여 연구되었음

요 약

개인정보보호법에 따라 사용자의 패스워드는 해쉬함수로 일방향 암호화되어 저장되어야 한다. 하지만 컴퓨터의 성능이 좋아짐에 따라 MD5나 SHA-1 같은 해쉬함수들은 충돌이 발견되어 사용하지 않을 것을 권고하고 있으며, 다른 해쉬함수들도 시간이 지남에 따라 안전하지 않게 될 수 있다. 패스워드를 안전하게 저장하기 위한 방안으로 사용하는 솔트는 사용자마다 랜덤하게 부여되며, 해쉬함수에 들어가기전 패스워드의 앞이나 뒤에 붙여 패스워드 보안을 강화한다. 하지만 솔트가 노출되는 경우에 공격자가 패스워드를 크래킹하는데 큰 도움이 된다. 본 논문에서 제안한 접속로그 기반의 동적솔트 방식은 데이터베이스나 로그인 관련 소스코드가 노출되더라도 솔트와 그 조합을 알 수 없게 만들어 패스워드를 보다 안전하게 보호할 수 있다.

Abstract

According to the Personal Information Protection Act, the user's password must be encrypted in one way through a hash function and stored in the database. However, as the computer's performance improved, MD5 and SHA-1 hash functions' collision occurred. Therefore, it is recommended not to use these hash functions, and other safe hash functions will be also unsafe over time. Salt, which is used as a method to safely store passwords, is randomly assigned to each user, and is attached to the front or back of the password before entering the hash function to enhance password security. However, if Salt is exposed to an attacker, it is very helpful to crack the password. The access log-based dynamic Salt method proposed in this paper can protect the password more securely by making the Salt and its combination more difficult to know even if the database or login related source code is exposed.

Keywords
password, login, hash function, access log, dynamic Salt

* 디제이패밀리 대표
- ORCID: http://orcid.org/0000-0002-2602-3062
** 안동대학교 컴퓨터공학과(교신저자)
- ORCID: http://orcid.org/0000-0002-9448-4899

· Received: Jun. 25, 2020, Revised: Aug. 14, 2020, Accepted: Aug. 17, 2020
· Corresponding Author: Young-Wook Cha
Andong National University, Korea
Tel.: +82-54-820-5714, Email: ywcha@anu.ac.kr

Ⅰ. 서 론

개인정보보호법의 개인정보 안전성 확보조치 기준 제 7조 2항에 따라 웹사이트의 패스워드는 복호화 되지 않도록 일방향 암호화하여 저장하여야 한다[1]. KISA(Korea Internet & Security Agency)의 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 발견되었고, 해쉬값이 무차별 공격과 사전공격에 의해 복호화 되었다[3][4]. 컴퓨터의 성능이 계속 좋아지게 된다면 안전하다고 지정된 해쉬함수들도 안전하지 않을 수 있다는 것이다.

해쉬함수를 이용하는 패스워드 관리의 안전성을 높이기 위하여 솔트(Salt)가 제시되었다[5]. 회원가입 시에 사용자에게 랜덤하게 부여되는 솔트는 패스워드의 앞이나 뒤에 붙여지며, 패스워드와 솔트가 해쉬함수에 입력되어 패스워드 해쉬값을 생성한다. 사용자마다 랜덤하게 할당되는 솔트라 하더라도, 한번 생성된 후에는 결국 고정값으로 데이터베이스에 저장된다. 데이터베이스가 해킹되는 경우 공격자는 각 사용자마다 할당된 솔트를 쉽게 파악하여 패스워드 크래킹에 이용할 수 있다는 취약점이 있다.

고정 솔트값 기반 패스워드 해쉬값의 취약성을 보완하기 위하여 본 논문에서는 로그인 시에 매번 변하는 접속로그 기반의 동적솔트(ALDS, Access Log-based Dynamic Salt) 방식을 제안한다. ALDS 방식에서 n개의 솔트를 사용하는 경우에 패스워드와 솔트의 서로 다른 조합은 n+1의 계승(Factorial) 개수가 된다. 개인정보보호법에 따라 필수적으로 저장해야 하는 최근 접속기록을 솔트로 사용함으로 별도의 동적 데이터 생성을 위한 부담을 없애며, 데이터베이스 노출 시에도 솔트의 존재를 감출 수 있게 된다. 또한 ALDS 방식에서는 소스코드가 노출되더라도 공격자는 크래킹에 (n+1)! 개의 서로 다른 패스워드와 솔트 조합을 적용해야 하므로 패스워드 해쉬값을 더욱 안전하게 보호할 수 있다.

본 논문의 2장에서는 해쉬함수와 솔트 관련연구를 기술하며, 3장에서는 접속로그 기반의 패스워드별 동적솔트 메커니즘을 제안한다. 4장에서는 제안한 동적솔트의 구현과 시험한 내용을 기술하며, 5장에서는 결론과 추후 연구계획에 대하여 기술한다.

Ⅱ. 관련 연구

2.1 취약한 해쉬함수와 패스워드 길이의 현황

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 MD5의 충돌을 발견하였으며[3], 구글 연구팀은 SHA-1의 충돌을 발견하였다[4]. SHA-1은 1993년 NIST에 의해 안전한 표준 해쉬함수로 지정되어 사용되었으나 컴퓨터 성능의 발전에 따라 더 이상 안전하지 않은 해쉬함수가 되었다. 해쉬함수의 무차별공격은 복호화 하려는 해쉬값을 미리 계산하여 두고, 가능성 있는 메시지들에 대하여 생성된 해쉬값과 복호화하고 싶은 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(=62^5) 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.

Securityledger사에서는 25개의 virtual AMD GPU 스펙으로 초당 1800억개의 MD5 해쉬값을 생성할 수 있는 패스워드 크래킹 PC를 소개하였다[8]. Securityledger사의 크래킹 PC는 가정용 PC에서 동작하는 MD5 CrackFast 툴에 비하여 약 2400배 빠르게 해쉬값을 생성할 수 있는 성능이다.

그림 1은 Statista에서 조사한 2017년 유출된 전세계 사용자 패스워드의 평균 글자 수를 나타낸다. 자료에 의하면 유출된 약 3억 2천만개의 패스워드 중 가장 많이 사용된 것은 7~8자리 패스워드들 이었다[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 솔트

솔트는 그림 2와 같이 패스워드의 앞이나 뒤에 추가되어 패스워드 해쉬값을 생성하는데 사용된다[10]. 솔트를 사용하면 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 Pritesh에서 제시한 솔트는 데이터베이스의 회원테이블에 하나의 컬럼으로 저장된다.

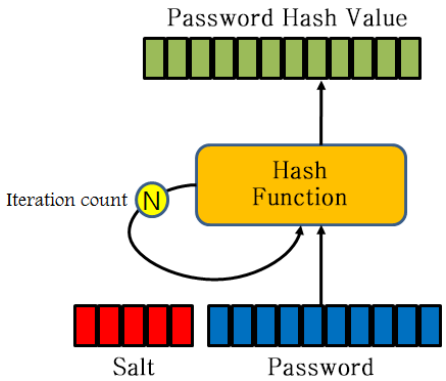


그림 2. 솔트를 이용한 패스워드 해쉬값 생성
Fig. 2. Generation of password hash value using Salt

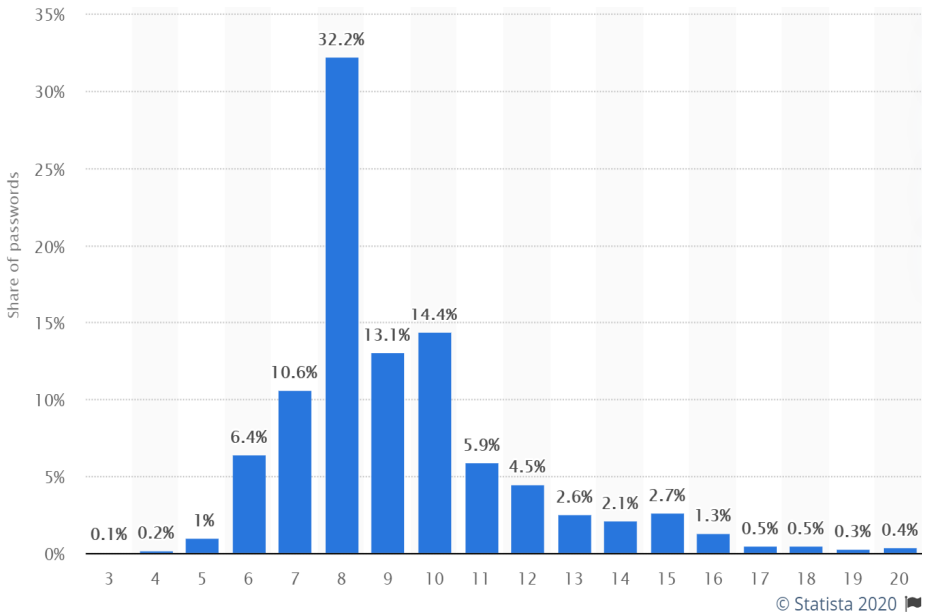


그림 1. 2017년에 유출된 전세계 사용자 패스워드의 평균 글자 수
Fig. 1. Average number of characters in leaked user passwords worldwide in 2017

사용자가 접속 시에 입력한 패스워드에 솔트 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.

III. 접속로그 기반의 패스워드 별 동적솔트 매커니즘

Shannon이 1949년에 고안한 One-Time Pad는 절대적으로 안전한 패스워드임이 입증되어 있다[12]. OneTime Pad의 요소인 “완벽한 랜덤(Perfectly random)”과 같이 데이터베이스에 저장되는 솔트와 패스워드 해쉬값도 계속 랜덤하게 변한다면, 무차별 공격과 사전공격으로 부터 패스워드가 보다 안전할 수 있을 것이다. 로그인 체크 시에 사용되는 솔트값은 고정 또는 가변될 수 있으며, 사용되는 솔트의 유형도 정적이거나 동적으로 변경될 수 있다. 패스워드 해쉬값을 생성하는데 이용하는 솔트의 값과 유형을 사용방법에 따라 표 1과 같이 4가지로 분류할 수 있다.

본 절에서는 고정 솔트값이 사용되는 기존 패스워드 해쉬값의 보안성을 강화하기 위하여 하나 이상의 접속로그를 사용하는 패스워드 별 동적솔트 방안을 제시한다. 접속로그 기반의 솔트는 로그인 체크 시에 솔트값은 변경되나 유형은 정적인 방식이며, 패스워드 별 동적솔트는 솔트의 값과 유형이 매번 변하는 방식이다.

표 1. 값과 유형에 따른 솔트 사용방법의 분류
Table 1. Classification of Salt usage by value and type

Salt	Classification	
Value	Fixed	Variable
Type	Static	Dynamic

3.1 접속로그 기반의 솔트

데이터베이스에 저장된 해쉬값의 변화를 일으키기 위하여 국내·외 개인정보보호 정책은 정기적으로 사용자에게 패스워드 변경을 요구하고 있다[13]. 하지만 2016년과 2017년 영국 NCSC와 미국 NIST에 따르면 정기적인 변경요구가 외우기 쉬운 단순한 패스워드를 사용하게 만들어 오히려 해킹에 취약하다는 결론을 내렸다[14][5]. 이에 NIST는 새로 공표한 2019년의 패스워드 가이드라인에서 정기적인 패스워드 변경 요구항목을 삭제하였다[16]. KISA 또한 2019년 개정된 패스워드 선택 및 이용 안내서에 해당 내용을 제외시켰다[17].

국내·외 개인정보보호 정책에 따라 웹사이트의 개인정보 처리자는 방문자의 접속기록 (IP 주소, 접속날짜, 브라우저, OS 종류 등)을 반드시 데이터베이스에 저장하여야 한다[1][18]. 로그인 시에 변경되는 접속 기록들을 솔트로 사용하면 개인정보보호 정책도 따르면서, One-Time Pad[12]의 완벽한 랜덤까지는 아니더라도 패스워드 해쉬값을 자주 바꿀 수 있게 된다. 즉, 접속로그 기반의 솔트(Access log-based Salt)는 로그인할 때마다 패스워드 해쉬값이 변경되므로 데이터베이스 관리자나 공격자 입장에서는 사용자가 패스워드를 자주 변경하는 것과 동일한 효과를 보이게 된다.

이처럼 접속기록 기반의 솔트를 사용하게 되면 데이터베이스가 유출되더라도 Salt라는 컬럼이 별도로 존재하지 않으므로 공격자에게 솔트의 존재 및 사용여부를 감출 수 있게 된다. 또한, 공격자가 패스워드 크래킹 중에 사용자가 로그인을 하면 패스워드 해쉬값이 바뀌게 되어 크래킹을 새로 시도해

야하므로, 패스워드의 보안 레벨이 매우 높게 안정적으로 유지될 수 있게 된다.

3.2 패스워드 별 동적솔트

접속로그 기반의 솔트는 소스코드가 공격자에 노출되는 경우에 솔트로 사용하는 접속기록도 노출된다. 이 경우에 공격자는 솔트로 사용된 접속기록을 이용하여 패스워드 해쉬값을 크래킹하게 된다. 데이터베이스와 소스코드가 노출되더라도 사용된 솔트들이 무엇인지 파악하기 어렵게 만들어 패스워드 해쉬값을 더 안전하게 보호할 수 있는 방안이 요구된다. 패스워드 별 동적솔트(Dynamic Salt according to password)는 사용자가 입력한 패스워드를 숫자화하여 N개의 접속기록들 중에서 r개를 솔트로 선택하여 패스워드 해쉬값을 생성하는 방식이다. 이러한 패스워드 별 동적솔트는 데이터베이스와 소스코드가 노출되더라도 사용자마다 서로 다른 r개의 솔트를 사용하므로 패스워드 해쉬값을 보다 안전하게 보호할 수 있다.

그림 3은 접속기록 N개를 전부 솔트로 사용하는 경우에 패스워드와 접속기록의 전체 조합을 나타낸다. 패스워드를 포함하여 N개의 접속기록을 모두 사용하는 경우에 가능한 조합의 수는 (N+1)의 계승 개수가 된다. 그림에서 각 조합의 왼쪽에 있는 x는 패스워드를 숫자화 하여 (N+1)!로 모듈러 연산을 수행한 결과이다.

그림 4는 N개의 접속기록을 모두 솔트로 사용하는 경우에 대한 패스워드 별 동적솔트 방식의 로그인 인증절차를 나타낸다. 웹사이트와 같은 서버는 사용자로부터 수신한 패스워드를 숫자화한 후, (N+1)!로 모듈러 연산을 수행한다. 연산의 결과로 x번째 솔트조합을 선택하여 패스워드 해쉬값을 생성하며, 이 해쉬값을 DB에 저장되어 있는 인증용 패스워드 해쉬값과 비교한다. 두 해쉬값이 일치하면 x번째 조합의 새로운 접속로그 기록으로 다음 로그인 인증에 사용할 패스워드 해쉬값을 생성하여 데이터베이스에 저장하며, 로그인 성공을 반환한다.

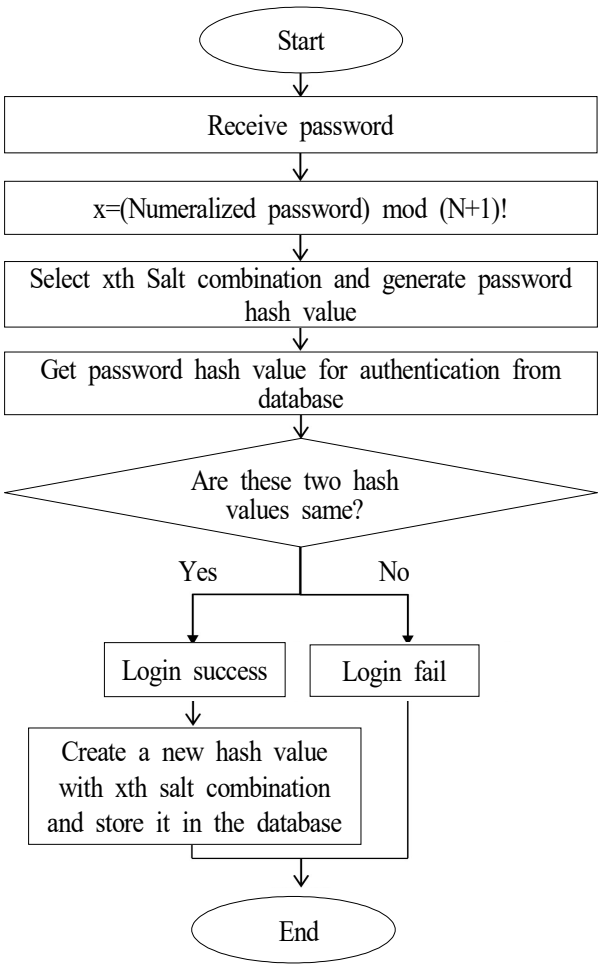


그림 4. 패스워드 별 동적솔트 방식의 로그인 인증절차
Fig. 4. Login authentication procedure of dynamic Salt by password

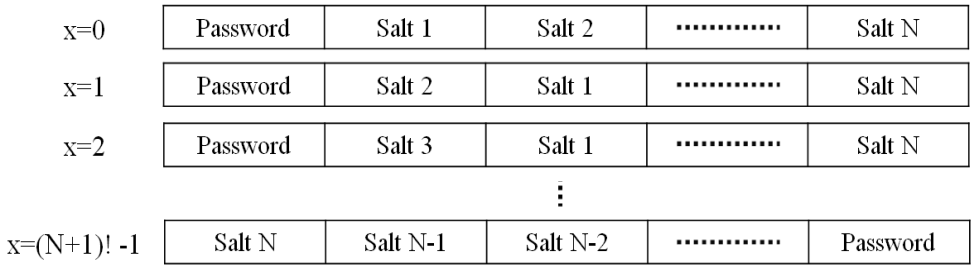


그림 3. 패스워드와 N개 솔트의 조합
Fig. 3. Combination of password and N Salts

패스워드 별 동적솔트 방식에서 공격자가 솔트조합을 얻기 위해서는 사용자가 입력한 패스워드를 숫자화한 후 모듈러 연산을 수행하여야 한다. 접속로그 기반 솔트와 다르게 패스워드 별 동적솔트 방식에서는 소스코드가 노출되더라도 공격자는 사용자의 패스워드를 알 수 없으므로 어떤 솔트조합을 사용하여 패스워드 해쉬값을 생성하였는지 쉽게 알 수 없다. 해쉬값이 바뀌지 않더라도 공격자는 소스코드에 나타나는 모든 경우의 솔트조합을 이용하여 패스워드 해쉬값을 크래킹하여야 한다. N개의 접속기록을 모두 솔트로 사용하는 경우라면 패스워드와 솔트를 포함한 전체 경우의 조합 수는 (N+1)! 이므로 평균 (N+1)! /2번의 크래킹이 요구된다.

접속기록 뿐만 아니라 사용자의 아이디, 이메일 주소, 생일과 같은 고정값을 솔트로 같이 사용하면 조합의 수가 증가하고 해쉬함수의 입력도 길어져 보안이 더욱 강화된 패스워드 해쉬값을 생성할 수 있다. 고정값들이 솔트조합에 포함되더라도 접속기록과 같이 로그인할 때마다 변하는 가변값들이 솔트조합에 섞여 있으므로 동적솔트의 장점은 유지하면서 패스워드 해쉬값의 보안 레벨을 매우 높게 그리고 안정적으로 유지할 수 있게 된다.

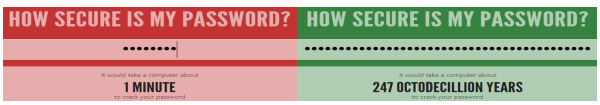
IV. 접속로그 기반의 동적솔트 구현과 시험

최근 접속시간(login_latest), 접속한 IP주소(login_ip), 그리고 총 로그인 횟수(login_counter)를 솔트로 사용하는 경우에 총 24개의 서로 다른 패스워드와 솔트의 조합이 가능하다. 그림 5는 접속로그 기반의 패스워드 별 동적솔트에 대한 PHP구현의 예이다. PHP의 unpack 함수에서 첫번째 인자인 I는 사용자 패스워드(user_password) 문자열을 정수로 변환하여 준다. 변환된 정수값을 솔트와 패스워드 조합의 전체 수인 24로 모듈러 연산을 수행한 후, 연산 결과에 따른 x번째 조합으로 MD5 해쉬함수에 입력하여 패스워드 해쉬값을 생성한다. 사용자의 패스워드가 “anu13579”라는 8자리 문자열이라고 가정하였을때, 솔트를 사용하지 않은 8자리 패스워드의 경우에 Securityledger사의 크래킹 PC기준 약 54초가 걸리게 된다.

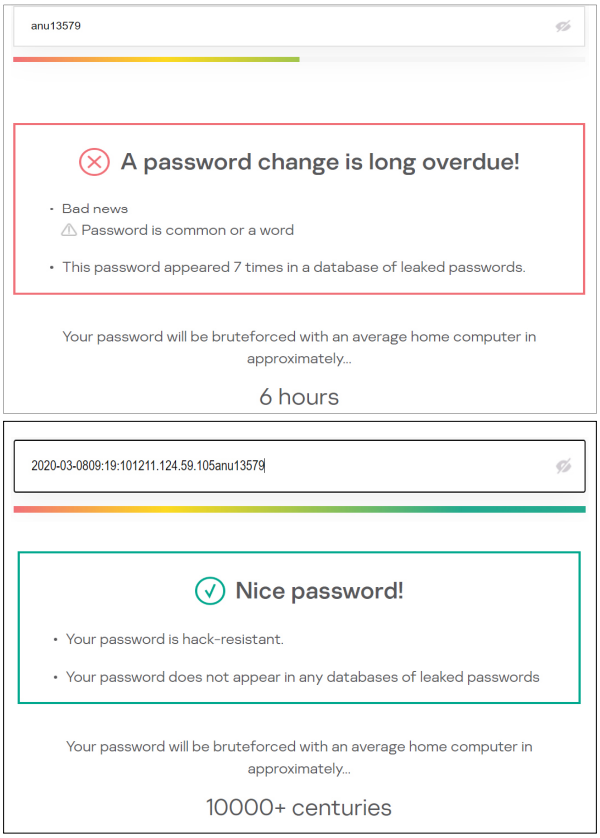
```
$mod = unpack("I", $user_password) [1] % 24;

if($mod == 0)
        $hash = md5($user_password.$login_latest.$login_ip.$login_count);
elae if($mod == 1)
        $hash = md5($user_password.$login_latest.$login_count.$login_ip);
elae if($mod == 2)
        $hash = md5($user_password.$login_count.$login_latest.$login_ip);
elae if($mod == 3)
        $hash = md5($user_password.$login_ip.$login_latest.$login_count);
elae if($mod == 4)
        $hash = md5($user_password.$login_ip.$login_count.$login_latest);
elae if($mod == 5)
        $hash = md5($user_password.$login_count.$login_ip.$login_latest);
                                        .
                                        .
                                        .
elae if($mod == 21)
        $hash = md5($login_ip.$login_latest.$login_count.$user_password);
elae if($mod == 22)
        $hash = md5($login_ip.$login_count.$login_latest.$user_password);
elae if($mod == 23)
        $hash = md5($login_count.$login_ip.$login_latest.$user_password);
```

그림 5. 패스워드 별 동적 솔트의 PHP코드
Fig. 5. PHP code of password based dynamic Salt



(a) howsecureismypassword.net의 크래킹 예상시간
(a) Estimated cracking time of howsecureismypassword.net



(b) Kaspersky 패스워드 안전성 체크툴
(b) Kaspersky password checker
그림 6. 패스워드와 솔트조합의 크래킹 예상시간
Fig. 6. Estimated cracking time for password and Salt combination

그러나 위의 접속기록 3가지와 패스워드를 포함한 길이는 42자리가 되어 12자리 패스워드 크래킹에 걸리는 약 9,304일 보다 훨씬 더 많은 시간이 걸리게 된다.

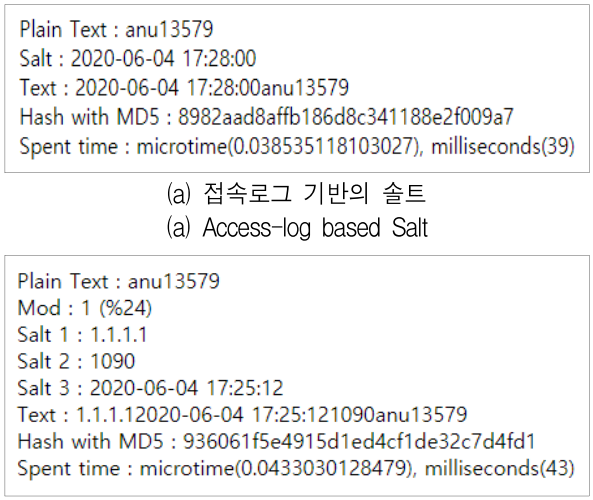
패스워드의 크래킹 예상 시간을 알려주는 사이트인 howsecureismypassword.net[19]에서 예시로 제시한 패스워드 “anu13579”는 그림 6과 같이 1분만에 크래킹 된다고 나왔다. 반면에, 패스워드와 솔트들의 조합 중 하나인 “2020-03-0809:19:101211.124.59.105anu13579”는 크래킹이 불가능한 247 OCTODECILLION YEARS 라는 큰 시간이 측정되었다. Kaspersky의 패스워드 안전성 체크툴[20]에서는 “anu13579”가 가정용 PC기준 6시간만에 크래킹 된다고 측정되었으며, 조합 중 하나인 “2020-03- 0809:19:101211.124.59.105anu13579”는 10000+centuries라는 큰 시간이 측정되었다.

그림 7은 웹에서의 로그인 체크시에 MD5 해쉬함수를 이용하는 경우에 접속로그 기반의 솔트와 패스워드 별 동적솔트 방식에 대한 패스워드 해쉬값의 생성 시간을 비교한 것이다.

접속로그 기반의 솔트에서는 최근 접속시간을 솔트로 사용하였으며, 패스워드 별 동적솔트에서는 그림 5에서 사용한 3가지 접속로그를 솔트로 사용하였다. 측정을 위한 코드는 PHP로 구현하였으며, MYSQL 데이터베이스와 3.40GHz CPU 환경에서 50회를 측정한 평균을 계산하였다.

표 2는 Pritesh의 정적솔트와 본 논문에서 제안한 솔트방식을 솔트의 값과 유형, 취약성, 요구되는 공격 증가량, 그리고 해쉬값의 생성시간을 비교하였다. 정적인 솔트 유형을 사용하는 접속로그 기반의 솔트와 Pritesh의 정적솔트에 대한 패스워드 해쉬값의 생성시간은 솔트의 개수를 1개라 가정하였을 때 동일하며, 시험에서 측정된 생성시간은 39ms이었다. 서로 다른 3가지의 접속로그를 솔트로 사용하는 동적솔트 방식에서의 생성시간은 43ms로 측정되었다. 패스워드를 숫자화하고 모듈러 연산을 추가하여 실제 로그인 체크의 환경처럼 구현하여 측정했음에도 접속로그 기반의 솔트방식과 패스워드 별 동적솔트 방식의 생성시간 차이가 거의 나지 않음을 확인할 수 있었다.

사용되는 솔트값이 가변이며, 솔트유형이 동적으로 선택되므로, 패스워드 별 동적솔트 방식은 데이터베이스와 소스코드가 노출되더라도 솔트와 솔트조합을 알 수 없게 하여 패스워드를 보다 안전하게 보호할 수 있다.



(a) 접속로그 기반의 솔트
(a) Access-log based Salt

(b) 패스워드 별 동적솔트 (N=3)
(b) Password based dynamic Salt (N=3)

그림 7. 패스워드 해쉬값의 생성시간 비교
Fig. 7. Comparison of generation times for password hash values

표 2. 솔트방식의 비교
Table 2. Comparison of Salt methods

	Pritesh Salt	Access-log based Salt	Password based dynamic 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)	39ms(N=1)	39ms(N=1)	43ms(N=3)
Maximum number of attacks	Twice	Twice	(N+1)! times

N개의 솔트와 패스워드를 이용하여 만들 수 있는 전체 조합의 수는 (N+1)! 이며, 패스워드별 동적솔트에서는 최대 (N+1)! 만큼의 패스워드 해쉬값에 대한 공격시도가 요구되므로 정적솔트에 비하여 패스워드를 더 안전하게 보호할 수 있다. 예를 들어, 솔트의 개수가 10개라면 전체 만들 수 있는 조합의 수인 최대 39,916,800배 만큼의 크래킹 시간이 증가할 수 있다.

V. 결 론

고정 솔트값을 사용하는 기존 패스워드 해쉬값의 보안성을 강화하기 위하여 접속로그 기반의 패스워드 별 동적솔트 방안을 제시하였다. Pritesh의 기존 솔트 방식은 솔트값이 데이터베이스의 Salt라는 컬럼에 고정값으로 저장된다. 반면에, 본 논문에서 제안한 동적솔트 방식은 접속로그를 사용하므로 로그인 체크시에 솔트값이 매번 변경되며, 사용자가 입력하는 패스워드에 따라 동적으로 N개 솔트와 패스워드의 조합이 선택되어 패스워드 해쉬값이 생성된다.

동적솔트 방식은 어떠한 해쉬함수와도 결합될 수 있으며, 사용하는 솔트의 개수에 따라 해쉬함수의 입력 길이가 늘어난다. 취약성에도 불구하고 계산속도가 빨라 여전히 많이 사용되고 있는 MD5의 경우에도 솔트조합에 의하여 입력되는 길이가 12이면, 크래킹에 9304일 이상이 소요되는 것으로 알려져 있다[7][8].

또한 데이터베이스나 로그인 관련 소스코드가 유출되어도 가변적인 솔트값과 동적인 솔트조합의 유추가 어려우므로 공격자는 모든 경우의 패스워드와 솔트 조합을 이용하여 크래킹을 시도하여야 한다. N개의 솔트와 패스워드를 이용하여 만들 수 있는 전체 조합의 수는 (N+1)! 이며, 평균 (N+1)! / 2 만큼의 패스워드 해쉬값에 대한 무차별 공격시도가 요구되므로 패스워드를 더 안전하게 보호할 수 있다.

패스워드를 보다 안전하게 보호하는 방법은 로그인과 상관없이 솔트를 지속적으로 변화시키는 것이다. 이를 위하여 Shannon이 고안한 One-time Pad의 완벽한 랜덤에 근접할 수 있는 솔트와 패스워드의 조합 방법을 추후 연구할 계획이다. 그리하여 컴퓨터 성능의 발전이 개인정보 유출에 악영향을 끼치지 않는 사이버 세계에 기여하고자 한다.

References

[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] Xiaoyun Wang, Dengguo Feng, Xuejia Lai, and Hongbo Yu, "Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD", Cryptology ePrint Archive, Report 2004/199, pp. 1-4, Aug. 2004.

[4] Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini, and Yarik Markov, "The first collision for full SHA-1", CRYPTO 2017, pp. 1-23, Jul. 2017.

[5] Pritesh N. Patel, Jigisha K. Patel, and Paresh V. Virparia, "A Cryptography Application using Salt Hash Technique", International Journal of Application or Innovation in Engineering & Management (IJAIEM), Vol. 2, No. 6, pp. 1-4, Jun. 2013.

[6] PHP.net, https://www.php.net/manual/en/function.md5.php, [accessed: Jun. 04. 2020]

[7] LAUGHFOOL's LAB, https://laughfool.tistory.com/16, [accessed: Jun. 04, 2020]

[8] The security ledger, https://securityledger.com/2012/12/new-25-gpu-monster-devours-passwords-in-seconds/, [accessed: Jun. 04, 2020]

[9] Statista, https://www.statista.com/statistics/744216/worldwide-distribution-of-password-length/, [accessed: Jun. 04, 2020]

[10] CWE, https://cwe.mitre.org/data/definitions/759.html, [accessed: Jun. 04, 2020]

[11] Samsung Electronics Co. Ltd., "User Device Performing Password Based Authentication and

Password Registration and Authentication Method Thereof", Korean Patent No. 10-2014-0024427.
[12] Nithin Nagaraj, Vivek Vaidya, and Prabhakar G. Vaidya, "Re-visiting the One-Time Pad", International Journal of Network Security, Vol. 6, No. 1, pp. 94-02, Jan. 2008.
[13] Personal Information Protection Total Portal, https://www.privacy.go.kr/a3sc/per/chk/examInfoViewCQ4.do, [accessed: Jun. 04, 2020]
[14] NCSC, https://www.ncsc.gov.uk/blog-post/problems-forcing-regular-password-expiry, [accessed: Jun. 04, 2020]
[15] Boannews, https://www.boannews.com/media/view.asp?idx=56986, [accessed: Jun. 04, 2020]
[16] Alvaka Networks, https://www.alvaka.net/new-password-guidelines-us-federal-government-via-nist/, [accessed: Jun. 04, 2020]
[17] Korea Internet & Security Agency (KISA), "Password Selection and User Guide", 2019.
[18] NIST, https://www.nist.gov/privacy-policy, [accessed: Jun. 04. 2020]
[19] How Secure Is My Password?, https://howsecureismypassword.net, [accessed: Jun. 04, 2020]
[20] Kaspersky Password Check, https://password.kaspersky.com, [accessed: Jun. 04, 2020]

저자소개

정 진 호 (Jin-Ho Jeong)

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

차 영 욱 (Young-Wook Cha)

1987년 : 경북대학교 전자공학과 (공학사)
1992년 : 충남대학교 전자통계학과 (공학석사)
1998년 : 경북대학교 컴퓨터공학과 (공학박사)
1987년 ~ 1999년 : 한국전자통신연구원 선임연구원
2003년 ~ 2004년 : 매사추세츠 주립대학 방문학자
2018년 ~ 현재 : 경찰청 디지털포렌식 자문위원
1999년 ~ 현재 : 안동대학교 컴퓨터공학과 교수
관심분야 : 망/시스템 제어 및 관리, ICT 및 스마트팜 보안, 개방형통신망