Journal of KIIT. Vol. 20, No. 3, pp. 67-77, Mar. 31, 2022. pISSN 1598-8619, eISSN 2093-7571
http://dx.doi.org/10.14801/jkiit.2022.20.3.67

접속로그와 패스워드 별 가변-동적솔트를 사용한 패스워드의 보안 강화

정진호*, 차영욱**

Enhancement of Password Security using Variable-Dynamic Salt by Access Log and Password

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

이 논문은 안동대학교 기본 연구지원 사업에 의하여 연구되었음

요 약

개인정보의 안전성 확보조치 기준에 의거하여 웹사이트의 패스워드는 복호화되지 않도록 해시함수를 사용한 일방향 암호화로 저장되어야 한다. 컴퓨터의 성능이 향상됨에 따라 MD5나 SHA-1 에도 충돌이 발견되었으며, 안전하다고 지정된 해시함수도 시간이 지남에 따라 안전하지 않을 수 있다. 해시함수의 보안강화를 위한 이전 방식에서는 솔트가 각 사용자에게 무작위로 할당되며 한번 생성된 후에는 고정된 값으로 저장되므로, 데이터베이스가 노출되면 솔트가 쉽게 식별되는 취약점이 있었다. 본 논문에서는 로그인 시에 가변되는 접속기록들과 패스워드로 구성되는 솔트조합을 동적으로 선택하는 가변-동적솔트 방식을 제안한다. 가변-동적솔트는 패스워드의 입력값에 따라 동적으로 가변의 솔트조합을 선택하여 패스워드 해시값을 생성하므로, 데이터베이스나 소스코드가 노출되어도 선택한 솔트조합의 식별이 어려워 기존 솔트 방식보다 패스워드를 안전하게 보호할 수 있다.

Abstract

According to Personal Information Safeguard and Security Standard, the password of the website should be stored by one-way encryption using a hash function to prevent decryption. As computer performance has improved, vulnerabilities have been found in MD5 and SHA-1 as well, and hash functions which were designated as safe may not be safe over time. In the previous method for strengthening the security of the hash function, Salt is randomly assigned to each user and stored as a fixed value. There is a vulnerability in which the Salt is easily identified when the database is exposed to attackers. In this paper, we propose variable-dynamic Salt method that dynamically selects a Salt combination composed of variable access log and password during the login process. The variable-dynamic Salt dynamically selects a variable Salt combination according to the password input value and creates a password hash value, so even if the database or source code is exposed, it is difficult to identify the selected Salt combination, so the password is more secure than the previous Salt method.

Keywords
access log, password, salt, hash function

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

· Received: Dec. 20, 2021, Revised: Jan. 11, 2022, Accepted: Jan. 13, 2022
· Corresponding Author: Young-Wook Cha
Andong National University, Korea
Tel.: +82-54-820-5714, Email: ywcha@anu.ac.kr

Ⅰ. 서 론

개인정보의 안전성 확보조치 기준 제 7조 2항에 의거하여 웹사이트의 패스워드는 일방향 암호화를 통해 저장해야 한다[1]. KISA(Korea Internet & Security Agency)의 개인정보의 암호화 조치 안내서에 따라 일방향 암호화는 해시함수를 이용한 암호화로 정의되며, KISA와 미국 국립표준기술연구소(NIST) 등 국내·외 정보보안 전문기관에서는 안전한 해시함수를 권고하고 있다[2]. 많이 사용되었던 MD5(Message-Digest algorithm 5)와 안전하다고 여겨졌던 해시함수인 SHA-1(Secure Hash Algorithm 1)의 충돌이 발견되었으며[3][4], 컴퓨터의 성능이 좋아짐에 따라 안전하다고 지정된 해시함수들도 더 이상 안전하지 않을 수 있다.

패스워드의 보안 강화를 위해 제안된 솔트는 패스워드의 앞이나 뒤에 붙여져 해시값을 생성하므로 패스워드 해시값의 보안을 강화한다[5]. Pritesh[6]가 제안한 솔트 사용방식은 각 사용자마다 랜덤하게 솔트를 부여한 후 데이터베이스의 Salt 컬럼에 저장한다. 이 경우 솔트가 각 사용자마다 랜덤하게 할당되더라도, 한번 생성된 후에는 고정된 값으로 데이터베이스 컬럼에 저장된다. 데이터베이스가 노출되는 경우 공격자는 각 사용자에게 부여된 솔트가 무엇인지 알 수 있으며, 파악한 솔트를 패스워드 크래킹에 이용할 수 있게 된다.

본 논문에서는 기존에 제시된 고정 솔트를 이용한 패스워드 해시값의 취약성을 보완하기 위하여 가변-동적솔트(Variable-dynamic salt) 방식을 제안한다. 제안한 방식에서는 가변의 접속로그들로 구성되는 솔트조합이 사용자가 입력한 패스워드에 따라 동적으로 선택되는 방식이다. 가변-동적솔트 방식에서 n개의 솔트를 사용하는 경우에 패스워드와 솔트들의 서로 다른 조합은 (n+1)! 개수가 된다. 개인정보보호법에 의거하여 반드시 저장해야 하는 최근 접속기록(접속날짜, IP주소, OS종류, 브라우저 등)들을 솔트로 사용함으로써 데이터베이스가 노출되어도 솔트의 존재를 감출 수 있다. 또한 소스코드가 노출되어도 공격자는 크래킹에 서로 다른 패스워드와 솔트조합 (n+1)! 개를 적용해야 하며, 사용자가 새로운 로그인 시에 공격자는 첫 번째 솔트조합부터 다시 크래킹 시도를 해야 하므로 패스워드 해시값을 보다 안전하게 보호할 수 있다.

본 논문의 2장에서는 해시함수와 솔트에 대한 관련연구를 기술하며, 3장에서는 가변-동적솔트 메커니즘을 제안한다. 4장에서는 제안한 가변-동적솔트를 구현하고 시험한 내용을 기술하며, 5장에서는 결론 및 추후 연구계획에 대하여 기술한다.

Ⅱ. 관련 연구

2.1 해시함수와 솔트

해시함수의 보안적 요구사항은 역상저항성과 충돌저항성이다. 컴퓨터의 성능이 향상됨에 따라 MD5와 SHA-1은 충돌이 발견되어 사용을 권고하지 않는 해시함수가 되었다[3][4]. 정보보안 뉴스 웹사이트인 Securityledger에서는 초당 1800억개의 MD5 해시값을 생성할 수 있는 25개의 가상 AMD GPU를 장착한 패스워드 크래킹 용 PC를 소개하였다[7].

2017년 Statista의 통계에 따르면 전세계에 유출된 약 3억 2천만개의 사용자 패스워드 중 가장 많이 사용된 글자 수는 그림 1과 같이 8자리이었다[8]. 패스워드가 알파벳 대소문자와 숫자의 조합인 경우 n자리에 대한 모든 경우의 수는 62의 n제곱이며, 8자리 패스워드의 MD5 해시값은 Securityledger의 크래킹 PC에 의해 10분정도 만에 크래킹 될 수 있다. 그러나 11자리의 경우에 4년 이상의 시간 소요되어 무차별 공격을 통한 패스워드 크래킹이 쉽지 않다. 사용하는 패스워드의 길이가 길수록 보안성은 좋지만 불편함 때문에 7~10자리의 패스워드를 많이 사용하고 있다.

솔트는 그림 2와 같이 패스워드의 앞이나 뒤에 추가되어 패스워드 해시값을 생성하는데 사용된다[9]. 패스워드와 솔트를 결합하면 해시함수의 입력이 길어져 해시값을 크래킹 하는데에 있어 더 많은 시간이 소요되어 패스워드를 보다 안전하게 보호할 수 있는 방법이다. Pritesh[6]는 패스워드에 솔트를 사용하면 사전공격의 위협을 줄이며, 크래킹이 어려워 질 뿐만 아니라 SQL인젝션 공격 또한 막을 수 있다고 기술하였다.

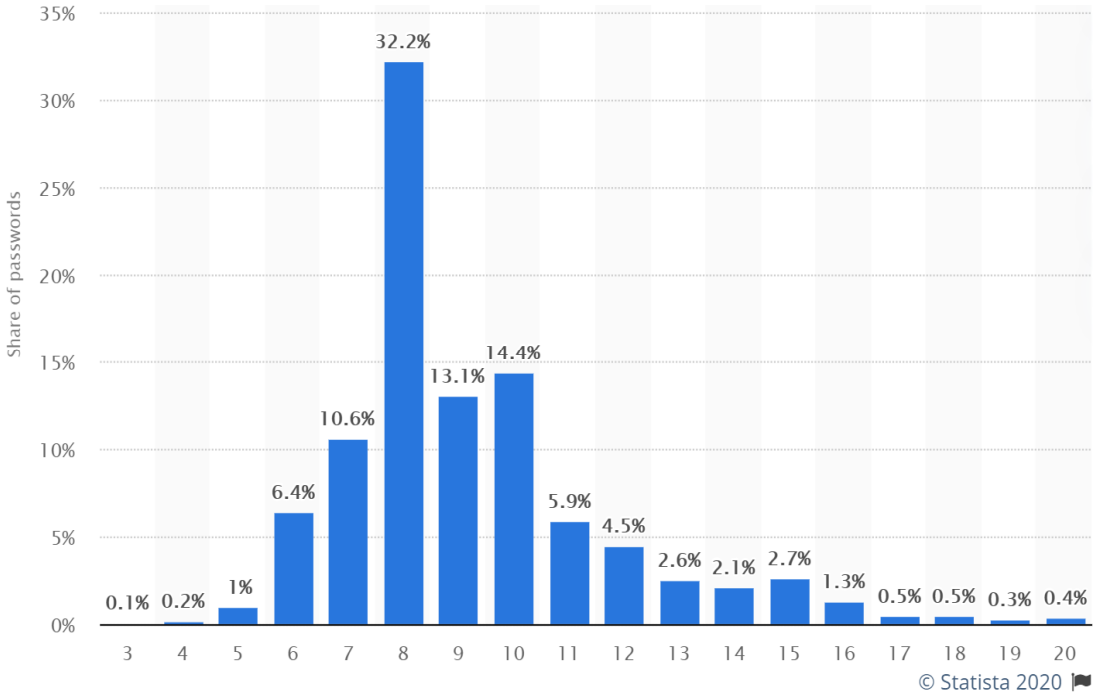


그림 1. 2017년에 전세계 유출된 사용자 패스워드의 평균 글자 수
Fig. 1. Average number of characters in leaked user passwords worldwide in 2017

ID	USERNAME	PASSWORD	SALT
1	anu_admin	1dadc3d49358430f187349a2201d148d64a498462262fc8f46...	03ac674216f3e15c761ee1a5e255f067953623c8b388b4459e...

그림 3. Pritesh 솔트에서의 패스워드 해시값과 salt 컬럼
Fig. 3. Salt and password hash value columns in Pritesh salt

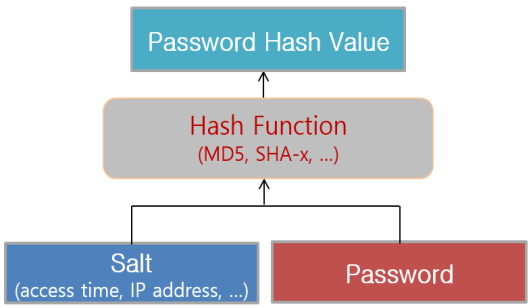


그림 2. 솔트와 해시함수를 이용한 패스워드 해시값 생성
Fig. 2. Generation of password hash value using salt and hash function

그러나 Pritesh에서 제시한 솔트는 그림 3과 같이 데이터베이스의 회원테이블에 salt 컬럼으로 저장된다. 사용자가 로그인 시에 입력한 패스워드에 salt 컬럼의 값을 추가하여 해시함수에 입력하며, 생성한 해시값과 데이터베이스에 저장되어 있던 패스워드 해시값과의 일치여부를 판단하여 로그인 성공 또는 실패를 처리한다. 사용자별로 랜덤한 솔트가 생성 및 부여되어도, 데이터베이스 솔트 컬럼에 특정 값으로 고정되어 저장된다. 이에 따라 데이터베이스가 공격자로부터 해킹되는 경우에 공격자는 각 사용자에게 부여된 솔트가 무엇인지 쉽게 파악하여 패스워드를 크래킹 하는 데에 사용한다. 대부분의 솔트는 패스워드의 앞 또는 뒤에 붙여 사용되므로, 공격자에게는 솔트를 단순히 패스워드 앞이나 뒤에 붙이는 경우에 비해 무차별 공격 회수가 2배가량 늘어나는 것에 불과하다.

삼성전자의 특허에서 제시한 솔트 사용방법은 입력된 패스워드에 솔트를 붙여 조합 패스워드를 생성하고 인증정보를 생성한다. 솔트는 난수발생기로부터 생성되어 데이터베이스에 저장하고 인증 시 입력된 패스워드와 함께 조합하여 인증한다[10]. 이 방법 또한 솔트가 고정값으로 저장되어 솔트가 노출되면 크래킹에 취약하다는 문제점이 있다.

Sirapat[11]와 Dr.Abdelrahman[12]에 의해 제시된 솔트방법은 Pritesh의 솔트 사용방법과 같이 사용자별로 랜덤하게 솔트를 생성한다. 그러나 Sirapat는 생성된 솔트를 패스워드의 문자 사이 사이에 삽입하며, Dr.Abdelrahman가 제시한 솔트는 솔트 문자들을 재배열한 후 패스워드에 붙여 해시값을 생성한다. 이 방법들 또한 패스워드 해시값 생성에 사용된 솔트가 데이터베이스에 고정된 값으로 저장된다. 따라서 공격자에게 데이터베이스가 노출된다면 솔트도 따라서 노출된다. Pritesh의 솔트에 비하여 솔트를 삽입하는 위치를 중간으로 하거나 솔트 글자 배열을 다르게 함으로써 보다 안전할 수는 있으나, 솔트와 같이 패스워드 크래킹에 직접적인 영향을 주는 중요 데이터가 노출되는 것은 안전하다고 볼 수 없다. 만약 공격자에게 소스코드가 노출된다면 솔트의 배열 규칙뿐만 아니라 패스워드에 붙여질 솔트의 조합 위치까지 알 수 있게 되어 크래킹에 위협이 될 수 있다.

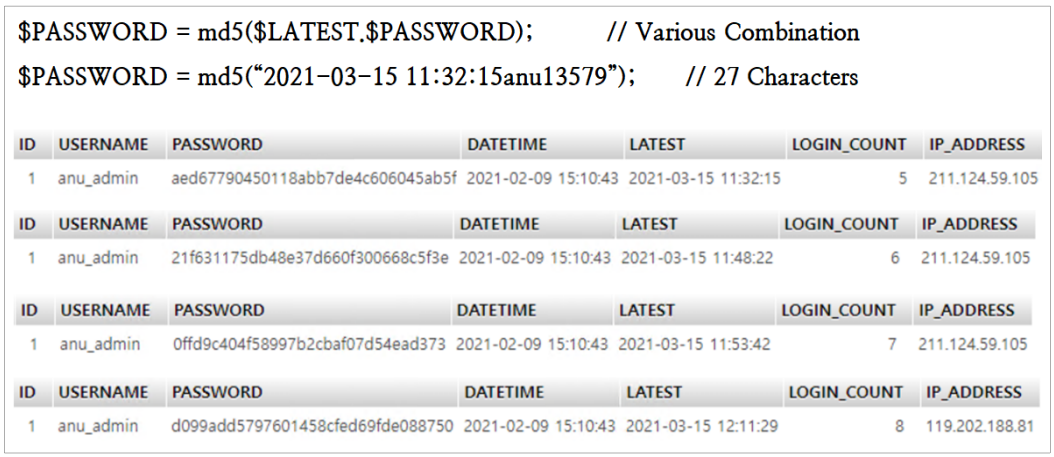
2.2 접속로그 기반의 솔트

Shannon의 One-Time Pad는 완벽한 패스워드임이 입증이 되어 있다[13]. One-Time Pad의 요소인 “완벽한 랜덤(perfectly random)”과 같이 데이터베이스에 저장되는 솔트와 패스워드 해시값도 계속 랜덤하게 변한다면, 무차별공격과 사전공격으로부터 패스워드 보안이 보다 강화될 것이다.

데이터베이스에 저장된 사용자들의 패스워드 해시값이 오랜기간 고정되지 않게 하기위하여 국내·외 개인정보보호정책은 사용자에게 정기적으로 패스워드를 변경할 것을 요구하였다[14]. 하지만 2016년과 2017년에 영국 NCSC와 미국 NIST에서는 정기적인 패스워드 변경요구로 인하여, 사용자가 외우기 쉽고 단순한 패스워드를 사용하게 만들어 오히려 해킹에 취약하다고 발표하였다[15][16]. 이에 따라 NIST와 KISA는 2019년 새로 공표한 패스워드 가이드라인에서 정기적인 패스워드 변경 요구항목을 삭제하였다[17][18].

국내·외 웹사이트의 개인정보처리자는 개인정보보호정책에 따라 방문자의 접속기록을 반드시 데이터베이스에 저장하여야 한다[1][19]. 로그인 시에 변경되는 접속 기록들을 솔트로 사용하는 접속로그 기반의 솔트(Access log based salt) 방식[20]은 개인정보보호 정책도 따르면서, One-Time Pad[14]의 완벽한 랜덤까지는 아니더라도 패스워드 해시값을 자주 바꿀 수 있게 된다. 즉, 접속로그 기반의 솔트는 사용자가 로그인할 때마다 패스워드 해시값이 변경되므로 데이터베이스를 볼 수 있는 관리자나 공격자가 보았을 때, 해당 사용자가 패스워드를 자주 변경하는 것처럼 보이게 할 수 있다.

그림 4는 접속로그를 활용한 솔트를 PHP 7.3, MYSQL 5.0 환경에서 구현하고, PHPMYADMIN으로 시각화한 것이다. 이처럼 접속로그를 솔트로 사용하면 데이터베이스가 노출되어도 솔트가 무엇인지 알 수 없으며, 해시값을 생성함에 있어 솔트 사용의 여부조차 알 수 없게 된다.



```
$PASSWORD = md5($LATEST.$PASSWORD);        // Various Combination
$PASSWORD = md5("2021-03-15 11:32:15anu13579");      // 27 Characters
```

ID	USERNAME	PASSWORD	DATETIME	LATEST	LOGIN_COUNT	IP_ADDRESS
1	anu_admin	aed67790450118abb7de4c606045ab5f	2021-02-09 15:10:43	2021-03-15 11:32:15	5	211.124.59.105

ID	USERNAME	PASSWORD	DATETIME	LATEST	LOGIN_COUNT	IP_ADDRESS
1	anu_admin	21f631175db48e37d660f300668c5f3e	2021-02-09 15:10:43	2021-03-15 11:48:22	6	211.124.59.105

ID	USERNAME	PASSWORD	DATETIME	LATEST	LOGIN_COUNT	IP_ADDRESS
1	anu_admin	0ffd9c404f58997b2cbaf07d54ead373	2021-02-09 15:10:43	2021-03-15 11:53:42	7	211.124.59.105

ID	USERNAME	PASSWORD	DATETIME	LATEST	LOGIN_COUNT	IP_ADDRESS
1	anu_admin	d099add5797601458cfed69fde088750	2021-02-09 15:10:43	2021-03-15 12:11:29	8	119.202.188.81

그림 4. 접속로그 기반의 솔트에서 접속시간을 이용한 패스워드 해시값 생성
Fig. 4. Generation of password hash value using latest in access log based salt

또한, 공격자가 패스워드를 크래킹 중일 때 사용자가 로그인을 하면, 접속로그가 변함에 따라 해시값이 변경되므로 공격자는 크래킹을 새로 시도하여야 한다. 하지만 접속로그 기반의 솔트 방식에서는 로그인과 관련된 소스코드가 해킹된다면 사용된 솔트가 무엇인지 노출되는 취약점이 있다[20].

III. 접속로그와 패스워드 별 가변-동적솔트 매커니즘

로그인 체크 시에 패스워드 해시값을 생성하는데 이용하는 솔트의 값과 조합은 사용방법에 따라 표 1에서와 같이 4가지로 분류할 수 있다. 솔트의 값은 고정 또는 가변일 수 있으며, 솔트로 사용되는 접속로그의 조합도 정적이거나 동적으로 변경될 수 있다.

표 1. 값과 조합에 따른 솔트 사용방법의 분류
Table 1. Classification of salt usage by value and combination

Salt	Classification	
Value	Fixed	Variable
Combination	Static	Dynamic

Pritesh가 제안한 방식[6]에서는 Salt라는 데이터베이스 컬럼에 솔트가 고정값으로 저장된다. 접속로그 기반의 솔트[20]는 로그인 시마다 접속기록들이 변하므로 솔트의 값이 가변적이다. 그러나 매번 동일한 접속로그의 조합이 사용되므로 접속로그 기반의 솔트는 가변-정적솔트 방식이다. 솔트의 조합을 동적으로 변하게 할 수 있다면 소스코드가 노출되더라도 패스워드의 해시값을 보다 안전하게 보호할 수 있게된다.

접속로그와 패스워드 별 가변-동적솔트는 사용자별로 서로 다른 접속로그의 솔트조합이 동적으로 선택되는 방식이다. 사용자가 입력한 패스워드를 숫자화한 후 솔트조합의 개수로 모듈러 연산하여 얻어진 수를 통해 솔트조합을 선택하므로, 사용자 별로 서로 다른 솔트조합이 동적으로 선택되어 패스워드 해시값을 생성하는 방식이다. 그림 5는 접속기록 N개를 전부 솔트로 사용하는 경우에 패스워드와 접속기록들의 전체 조합을 나타낸다. 패스워드를 포함하여 N개의 접속기록을 모두 사용하는 경우에 가능한 솔트조합의 수는 (N+1)의 계승 개수가 된다. 그림에 나타난 각 조합에서 왼쪽에 표시된 x는 패스워드를 숫자화 한 후 (N+1)!로 모듈러 연산을 수행한 결과 값이다.

가변-동적솔트 방식에서 공격자가 패스워드 해시값 생성에 사용된 조합을 얻기 위해서는 실제 해당 사용자의 패스워드를 숫자화한 후 전체 조합의 수로 모듈러 연산을 수행하여야 한다. 접속로그 기반 솔트[20]와 다르게 가변-동적솔트 방식에서는 소스코드가 노출되더라도 사용자의 패스워드를 알지 못하는 공격자는 패스워드 해시값 생성에 어떤 솔트조합이 사용되었는지 알 수 없다. 한동안 로그인이 없어 패스워드 해시값이 바뀌지 않더라도 공격자는 소스코드에 나타나는 모든 경우의 솔트조합을 이용하여 패스워드 해시값을 크래킹하여야 한다. 접속기록을 포함한 N개의 솔트를 모두 조합에 사용하는 경우에 패스워드와 솔트 조합의 전체 경우의 수는 (N+1)! 이다. 이 경우에 평균 (N+1)! /2번의 크래킹이 요구되며, 솔트 개수의 증가에 따라 조합의 수는 기하급수적으로 늘어나게 된다.

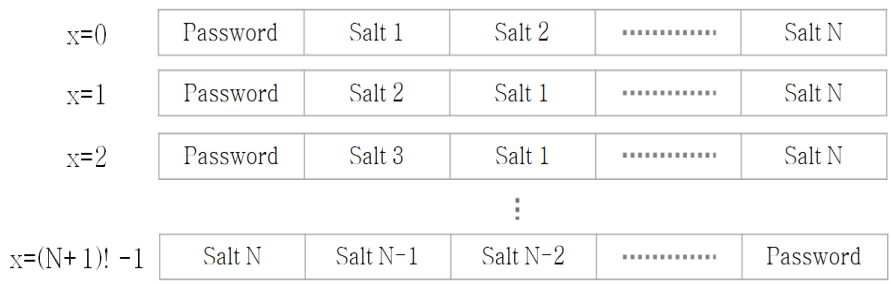


그림 5. 패스워드와 N개 솔트의 조합
Fig. 5. Combination of password and N salts

패스워드 만을 숫자화하여 솔트조합을 선택하는 기존 동적솔트 방식[21]의 경우에, 사용자의 패스워드가 변하지 않는 한 솔트의 조합을 선택하는 x값도 변하지 않게 된다. 소스코드와 데이터베이스가 노출된 경우에 새로운 로그인 시도가 있더라도 공격자는 모든 솔트조합에 대하여 크래킹을 시도해 볼 수 있다. 크래킹 시도 중 사용자가 로그인 시 데이터베이스에는 최근 로그인 시간과 로그인 IP 등이 새롭게 기록되며, 해당 패스워드의 솔트조합으로 패스워드 해시값이 변경된다.

기존 방식[21]에서는 크래킹 중 새로운 접속기록의 변경을 발견한 공격자는, 변경된 패스워드 해시값을 가지고 첫 번째 조합이 아닌 로그인이 되었던 시점의 솔트조합에서 크래킹을 계속하여 시도하여도 되는 취약점이 있다.

패스워드만 숫자화하는 방식의 취약점을 개선하기 위하여 본 논문에서는 사용자가 입력한 패스워드의 앞이나 뒤에 접속로그를 붙여 숫자화 한다. 사용자가 패스워드를 변경하지 않더라도 입력받은 패스워드에 붙은 접속로그로 인하여 로그인 시마다 숫자화된 값은 지속적으로 변하게 된다. 이는 (N+1)! 모듈러 연산의 x값도 변경되어 솔트조합도 로그인 시마다 동적으로 변하게 된다. 크래킹을 시도 중인 해커는 사용자가 새로운 로그인 시에 선택되는 솔트조합이 바뀌어 버리므로 첫 번째 솔트조합부터 다시 크래킹을 시도하여야 한다.

그림 6은 N개의 접속기록을 모두 솔트로 사용하는 경우에 대한 가변-동적솔트 방식의 로그인 인증절차를 나타낸다. 사용자로부터 수신한 패스워드의 앞이나 뒤에 데이터베이스에 저장된 접속로그를 붙여 숫자화한 후, (N+1)!로 모듈러 연산을 수행한다. 연산의 결과로 x번째 솔트조합을 선택하여 패스워드 해시값을 생성하며, 데이터베이스에 저장되어 있는 해시값과 비교한다. 두 해시값이 다르면 로그인 실패를, 같으면 성공을 반환한다. 로그인 성공 시에는 새롭게 생성된 접속로그를 입력된 패스워드에 붙여 숫자화한 후, (N+1)!로 모듈러 연산을 수행하여 y번째 솔트조합을 선택한다. 새로운 접속로그와 y번째 조합으로 생성된 패스워드 해시값을 다음 로그인 시의 인증을 위하여 데이터베이스에 저장한다.

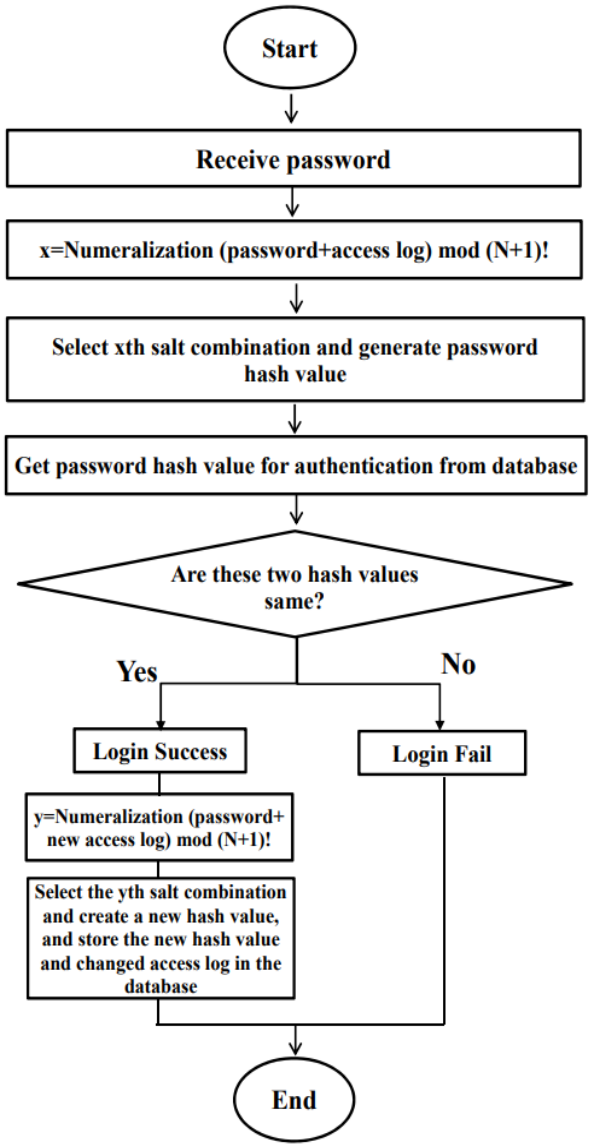


그림 6. 가변-동적솔트 방식의 로그인 인증절차
Fig. 6. Login authentication procedure of variable-dynamic salt

Ⅳ. 구현과 시험

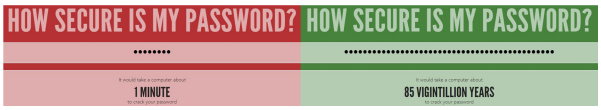
접속기록 중 최근 접속시간(Latest), 접속한 IP주소(Ip), 그리고 총 로그인 횟수(Login_count)를 솔트로 사용하는 경우에 총 24개의 서로 다른 패스워드와 솔트의 조합을 생성할 수 있다. 그림 7은 접속로그와 패스워드 별 가변-동적솔트에 대한 PHP구현의 예이다. PHP 코드에서 unpack 함수의 첫 번째 인자 I는 사용자 패스워드(Password) 문자열을 정수로 변환한다.

```php
$new_password = $password.$latest;

$x = unpack("I", $new_password)[1]%24;

if($x == 0)
    $combination = $latest.$ip.$login_count.$password;
else if($x == 1)
    $combination = $latest.$login_count.$ip.$password;
else if($x == 2)
    $combination = $latest.$login_count.$password.$ip;
else if($x == 3)
    $combination = $latest.$ip.$password.$login_count;
else if($x == 4)
    $combination = $latest.$password.$ip.$login_count;
else if($x == 5)
    $combination = $latest.$password.$login_count.$ip;
                    .
                    .
                    .
else if($x == 19)
    $combination = $latest.$ip.$login_count.$password;
else if($x == 20)
    $combination = $latest.$ip.$login_count.$password;
else if($x == 21)
    $combination = $latest.$ip.$login_count.$password;
else if($x == 22)
    $combination = $latest.$ip.$login_count.$password;
else if($x == 23)
    $combination = $latest.$ip.$login_count.$password;

$hash = md5($combination);
```

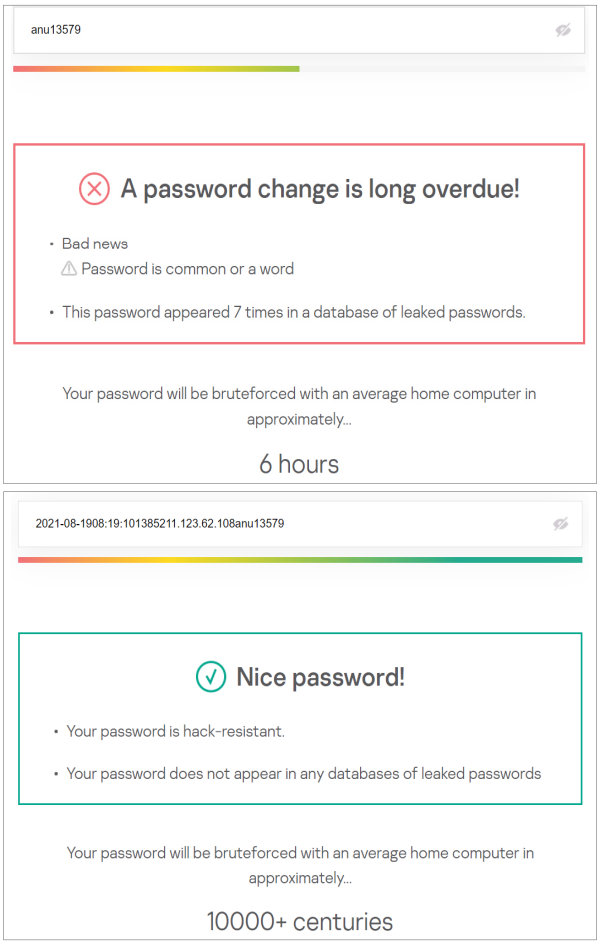
그림 7. 가변-동적 솔트의 PHP코드
Fig. 7. PHP code of variable-dynamic salt

변환된 정수값을 패스워드와 솔트조합의 전체 개수인 24로 모듈러 연산을 수행한 후, 연산 결과에 부합하는 솔트조합을 선택하여 MD5 해시함수에 입력하고 패스워드 해시값을 생성한다. 사용자의 패스워드를 “anu13579” 라는 8자리 문자열로 가정하였을때, 8자리 패스워드는 Securityledger사의 크래킹 PC기준 약 10분만에 크래킹 될 수 있다[7]. 그러나 위의 접속기록 3가지와 패스워드를 포함한 길이는 45자리가 되어 11자리 패스워드 크래킹에 걸리는 4년보다 훨씬 더 많은 시간이 소요된다.

그림 8은 howsecureismypassword.net[22]와 Kaspersky의 패스워드 안전성 체크툴[23]에서 측정한 패스워드와 솔트조합의 크래킹 예상시간을 나타낸다. howsecureismypassword.net에서는 예시로 제시한 패스워드 “anu13579”는 1분만에 크래킹 된다고 나왔다. 반면, 패스워드와 솔트들의 조합 중 하나인 “2021-08-19 08:19:101385211.123.62.108anu13579”는 크래킹이 불가능한 85 VIGINTILLION YEARS 라는 큰 시간이 측정되었다. Kaspersky에서는 “anu13579”가 일반 가정용 컴퓨터로 약 6시간만에 크래킹 될 수 있다고 나왔으며, 조합 중 하나인 “2021-08-19 08:19:101385211.123.62.108anu13579”는 10000+centuries 라는 큰 시간이 측정되었다.



(a) How secure is my password
(a) How secure Is my password



(b) Kaspersky의 패스워드 안전성 체크툴
(b) Kaspersky password checker
그림 8. 패스워드와 솔트조합의 크래킹 예상시간
Fig. 8. Estimated cracking time for password and Salt combination

표 2는 MD5 해시함수와 현재 안전하다고 알려진 SHA-256 해시함수의 속도를 비교한 표이다. 측정을 위한 코드는 PHP로 구현하였고, PHP 7.3버전, MYSQL 5.0 데이터베이스, 그리고 CPU 3.40GHz 환경에서 각 해시함수의 실행시간을 50회씩 측정하여 평균 계산하였다. 해시함수의 입력값은 솔트를 붙이지 않은 패스워드 “anu13579”와 예시로 사용된 솔트조합 중 하나인 “2021-08-19 08:19:101385211.123.62.108anu13579”를 사용하였다. “anu13579”를 입력값으로 사용하여 MD5 해시함수의 실행시간은

0.000276ms가 나왔고, SHA-256 해시함수의 실행시간은 0.000522ms가 나왔다. MD5와 SHA-256 해시함수에 “2021-08-19 08:19:101385211.123.62.108anu13579”를 실행한 소요시간은 "anu13579"를 사용하였을 때와 동일하였다.

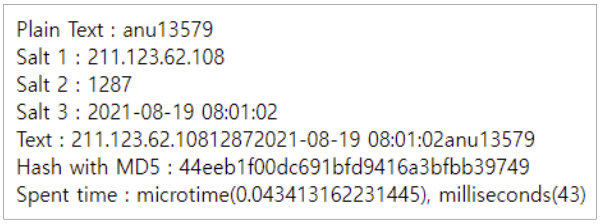
표 2. MD5와 SHA-256의 속도비교
Table 2. Speed comparison of MD5 and SHA-256

Hash function	Password only	Password with salt
MD5	0.000276 ms	0.000276 ms
SHA-256	0.000522 ms	0.000522 ms

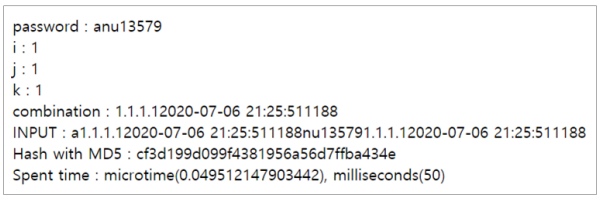
실험의 조건과 결과로 나온 수치로만 비교하였을 때, MD5는 SHA-256 보다 약 1.9배 빠르다고 볼 수 있다. 이에 따라 “anu13579”를 사용하여 생성된 해시값에 패스워드 크래킹 공격을 한다고 하였을 때, 충돌이 일어나지 않는다면 SHA-256은 MD5에 비해 약 1.9배 늦게 크래킹 된다고 볼 수 있다. 따라서 Securityledger사의 크래킹 PC[7] 기준으로 8자리를 MD5의 입력값으로 사용하여 생성된 해시값의 크래킹 성공 시간이 약 10분이므로, SHA-256은 1.9배인 약 19분이 걸리게 되어 여전히 취약하다. 그러나 “anu13579”에 솔트를 붙인 예시는 45자리이므로, Securityledger사의 크래킹 PC 기준으로 11자리 패스워드의 MD5 해시값 크래킹에 걸리는 4년보다 훨씬 더 많은 시간이 걸리게 된다. 따라서 MD5와 같이 취약하다고 알려진 해시함수를 이용하여도 입력값이 45자리의 예시처럼 길어지면, SHA-256과 같이 안전하다고 알려진 해시함수에 솔트를 사용하지 않는 짧은 입력값보다 크래킹에 훨씬 안전할 수 있다.

그림 9는 웹에서 로그인 시에 MD5 해시함수를 이용하는 경우의 접속로그 기반 가변-정적솔트[20]와 기존 가변-동적솔트 방식[21], 그리고 본 논문에서 제안한 가변-동적솔트 방식에 대한 패스워드 해시값의 생성시간을 비교한 것이다. 그림 7에서 사용한 3가지 접속기록을 솔트로 사용하였으며, MD5와 SHA-256 해시함수의 속도를 비교하였던 환경과 동일한 환경에서 50회 측정한 시간의 평균을 계산하였다.

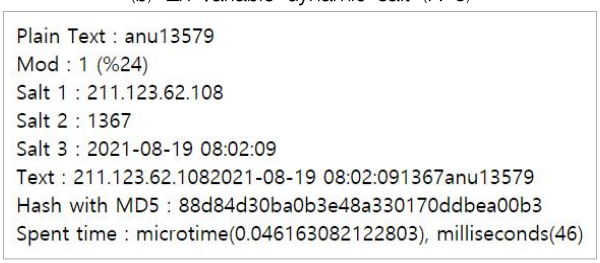
접속로그 기반의 가변-정적솔트 방식에서 패스워드 해시값의 생성시간은 43ms로 측정되었다. 가변-동적솔트에서는 사용자가 입력한 패스워드에 접속로그를 붙여 숫자화하며, 모듈러 연산으로 솔트조합을 선택하는 기능이 추가되므로, 접속로그 기반의 가변-정적솔트에 비하여 패스워드 해시값의 생성시간이 증가하였다. 본 논문의 가변-동적솔트 방식에서는 패스워드의 앞이나 뒤에 접속로그를 붙여 숫자화한 후, 모듈러 연산의 결과로 x번째 솔트조합을 선택하여 패스워드 해시값을 생성한다. 반면에 기존 방식[21]에서는 솔트조합을 선택하는 j, 선택된 조합을 패스워드 내에 위치시키는 i, 그리고 패스워드의 앞 또는 뒤에 삽입시키는 k의 처리를 수행하는 다소 복잡한 단계를 거쳐서 해시값을 생성한다. 이에 따라 본 논문에서 제시한 가변-동적솔트 방식의 패스워드 해시값 생성시간은 46ms로 기존 가변-동적솔트 방식[21]의 50ms에 비하여 약 8%의 성능향상을 보였다.



(a) 가변-정적솔트
(a) Variable-static salt



(b) 기존 가변-동적솔트
(b) Ex variable-dynamic salt (N=3)



(c) 가변-동적솔트
(c) Variable-dynamic salt (N=3)

그림 9. 접속로그 기반 패스워드 해시값의 생성시간 비교
Fig. 9. Comparison of generation times for access-log based password hash values

표 3. 솔트 방식의 비교
Table 3. Comparison of Salt methods

	Pritesh salt	Variable-static salt	Ex vvariable-dynamic	New variable-dynamic salt
Salt value	Fixed	Variable	Variable	Variable
Type	Static	Static	Dynamic	Dynamic
Vulnerability	Vulnerable in DB exposure	Vulnerable in DB and source code exposure	Safe in DB and source code exposure, but fixed salt combination is chosen	Safe in DB and source code exposure, and cracking from the first combination
Hash generation time (Salt count = 3)	43ms	43ms	50ms	46ms
Maximum number of attacks	2	2	(Password length+1) * N! * 2	(N+1)!

표 3은 기존 솔트방식과 본 논문에서 제안한 솔트방식을 솔트 값과 유형, 취약성, 해시값 생성시간, 그리고 요구되는 최대 공격 횟수를 비교한다. 솔트의 개수가 N일 때 기존 가변-동적솔트 방식[21]에서의 최대 공격횟수는 (Password length+1) * N! * 2가 된다. 본 논문에서 제시한 가변-동적솔트 방식에서는 패스워드의 길이와 무관하게 (N+1)!의 최대 공격횟수를 갖는다. 그러나 기존 방식에서는 사용자가 패스워드를 변경하지 않을 시에 i(패스워드 내 솔트조합의 위치), j(선택되는 솔트조합), k(패스워드의 앞이나 뒤)가 고정 값이 된다는 취약점이 있었다. 따라서 사용자가 패스워드를 변경하지 않을 시 공격자는 크래킹 시도가 가능하며, 로그인하더라도 로그인한 시점에서 계속 크래킹을 해 나가면 결국 패스워드는 크래킹 될 수 있다.

반면에 본 논문의 가변-동적솔트 방식에서는 사용자가 새로운 로그인 시에 선택되는 솔트조합이 바뀌므로, 크래킹 중인 해커는 첫 번째 솔트조합부터 크래킹을 다시 시도하여야 한다. 솔트개수가 3인 경우에 기존 가변-동적솔트에서[21]의 해시값 생성시간은 50ms이며, 본 논문의 가변-동적솔트 해시값 생성시간은 46ms로 기존에 비하여 8%의 개선을 보이고 있으며, 기존 가변-동적솔트 방식의 크래킹에 대한 취약성도 강화하였다.

가변-동적솔트는 가변-정적솔트에 비해 여러 기능이 추가되었지만 해시값의 생성시간은 약 6%인 3ms 정도만이 증가되었다. 가변-정적솔트에서는 공격자가 소스코드에 노출된 한 개의 정해진 솔트조합으로 크래킹을 시도한다. 그러나 가변-동적솔트에서는 소스코드가 노출되더라 N개의 솔트를 사용하는 경우 (N+1)!개의 솔트조합들에 대하여 평균 N!/2번 이상의 크래킹을 시도해야 한다. 솔트의 개수가 10개인 경우에 가변-동적 솔트에서는 평균 19,958,400(=11!/2)번의 크래킹을 시도하여야 하며, 공격자가 크래킹 시도 중 사용자가 로그인에 성공하면 공격자는 첫 번째 솔트조합부터 다시 크래킹을 시도하여야 한다.

최적의 N은 각 조직에서의 보안 요구사항에 의하여 결정된다. N이 8인 경우에 패스워드와 솔트의 전체 조합 수는 362,880(=9!)개이며, Securityledger사의 크래킹 PC[7] 기준으로 8자리 패스워드를 크래킹하는데 소요되는 시간은 약 10분이므로, 총 소요시간은 평균 1,814,400분(=약 3년)이 된다. 한편 개인정보 보호법 제 39조의 6에 따라 1년 동안 로그인하지 않은 이용자의 개인정보는 파기하거나 별도로 분리 저장하여야 한다[24]. 따라서 공격자가 크래킹 하는 동안 1년이 경과하면 개인정보는 파기되어 로그인 할 수 없으며, 만약 그 기간 안에 사용자가 로그인을 한다면 공격자는 솔트조합의 처음부터 크래킹을 다시 시도하여야 한다.

V. 결 론

Pritesh[6] 방식은 Salt라는 데이터베이스 컬럼에 솔트가 고정값으로 저장되는 반면, 접속로그 기반의 솔트[20]는 데이터베이스가 노출되더라도 공격자는 솔트가 무엇인지 알 수 없게 된다. 또한, 로그인 시마다 솔트 값이 변하기 때문에 패스워드 해시값도

같이 변하게 되어 패스워드를 안전하게 보호할 수 있다. 본 논문에서 제안한 접속로그와 패스워드 별 가변-동적솔트 방식은 사용자가 입력한 패스워드에 접속로그를 붙여 숫자화한 후 모듈러 연산으로 솔트조합을 선택하고, 선택된 솔트와 패스워드의 조합으로 해시값을 생성한다. 제안한 방식은 데이터베이스 뿐만 아니라 소스코드가 노출되어도 사용자의 패스워드 값을 알 수 없기 때문에, 공격자는 어떤 솔트조합을 사용하였는지 알 수 없게 된다. 또한 로그인 시마다 접속로그가 변하므로 선택되는 솔트조합도 지속적으로 변하게 되어 패스워드를 더욱 안전하게 보호할 수 있다.

솔트의 개수가 10개인 경우에 공격자는 최대 39,916,800(=11!)개의 조합에 대하여 크래킹을 시도하여야 하며, 크래킹 도중에 사용자가 로그인 시 공격자는 첫 번째 솔트조합부터 다시 크래킹을 시도하여야한다. 접속로그와 패스워드 별 가변-동적솔트는 이전 솔트 방식[20]에 비해 패스워드 보안이 훨씬 강화되었지만 패스워드 해시값의 생성시간은 이전 방식에 비해 약 6% 정도만 증가하였다. 컴퓨터의 성능이 발전함에 따라 모든 해시함수가 안전하다고 보장할 수 없는 현시점에서 가변-동적솔트 방식은 패스워드를 기존 방식보다 안전하게 보호할 수 있는 방안이 될 것이다.

패스워드를 더욱 안전하게 보호하기 위하여 로그인을 하지 않아도 패스워드 해시값이 지속적으로 변하게 하는 방법을 추후 연구할 계획이다. 이는 Shannon의 One-time Pad와 같이 완벽한 랜덤에 근접한 성능을 가지면서, 개발자와 사용자가 사용하기 편리한 패스워드 보호방법이 될 것이다. 그리하여 컴퓨터의 발전이 개인정보에 부정적인 영향을 끼치지 않는 사이버 세계를 만드는 데 기여하고자 한다.

References

[1] The Ministry of Public Administration and Security (Personal Information Protection Policy), "Personal Information Safeguard and Security Standard", 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] CWE, https://cwe.mitre.org/data/definitions/759.html, [accessed: Aug. 10. 2021]

[6] 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.

[7] The security ledger, https://securityledger.com/2012/12/new-25-gpu-monster-devours-passwords-in-seconds/, [accessed Aug 10, 2021]

[8] Statista, https://www.statista.com/statistics/744216/worldwide-distribution-of-password-length/, [accessed Aug 10, 2021]

[9] CWE, https://cwe.mitre.org/data/definitions/759.html, [accessed Aug 10, 2021]

[10] Samsung Electronics Co. Ltd., "User Device Performing Password Based Authentication and password Registration and Authentication Method Thereof", Korean Patent No. 10-2014-0024427.

[11] Sirapat Boonkrong and Chaowalit Somboonpattanakit, "Dynamic salt generation and placement for secure password storing", IAENG International Journal of Computer Science, 43, pp. 27-36, 2016.

[12] Dr.Abdelrahman Karrar, Talal Almutiri, Sultan Algrafi, Naif Alalwi, and Ammar Alharbi, "Enhancing Salted Password Hashing Technique Using Swapping Elements in an Array Algorithm", International Journal of Computer Science and Technology, Vol. 9, No. 1, pp. 21-25, 2018.

[13] 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.
[14] Personal Information Protection Total Portal, https://www.privacy.go.kr/a3sc/per/chk/examInfoViewCQ4.do, [accessed: Aug. 10, 2021]
[15] NCSC, https://www.ncsc.gov.uk/blog-post/problems-forcing-regular-password-expiry, [accessed: Aug. 10, 2021]
[16] Boannews, https://www.boannews.com/media/view.asp?idx=56986, [accessed: Aug. 10, 2021]
[17] Alvaka Networks, https://www.alvaka.net/new-password-guidelines-us-federal-government-via-nist/, [accessed: Aug. 10, 2021]
[18] Korea Internet & Security Agency (KISA), "Password Selection and User Guide", 2019.
[19] NIST, https://www.nist.gov/privacy-policy, [accessed: Aug. 10. 2021]
[20] J. Jeong, D. Woo, and Y. Cha, "Enhancement of Website Password Security by Using Access Log-based Salt", 2019 International Conference on Systems of Collaboration Big Data, Internet of Things & Security (SysCoBIoTS), Casablanca, Morocco, pp. 1-3, 2019.
[21] Jinho Jeong, Youngwook Cha, and Choonhee Kim "A Study on the Variable and Dynamic Salt According to Access Log and Password", Journal of Korea Multimedia Society Vol. 24, No. 1, pp. 58-66, 2021.
[22] How Secure Is My Password, https://howsecureismypassword.net, [accessed: Aug. 10, 2021]
[23] Kaspersky Password Check, https://password.kaspersky.com, [accessed: Aug. 10, 2021]
[24] Personal Information Protection Committee (Personal Information Protection Policy Division), "Personal Information Protection Act", 2020.

저자소개

정 진 호 (Jin-Ho Jeong)

2019년 : 안동대학교 컴퓨터공학과(공학사)
2021년 : 안동대학교 컴퓨터공학과(공학석사)
2021년 ~ 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
2019년 ~ 현재 : 안동대 사이버보안 센터장
1999년 ~ 현재 : 안동대학교 컴퓨터공학과 교수
관심분야 : 망/시스템 제어 및 관리, 블록체인 및 보안, 개방형통신망