**SQL 인젝션 공격**

**1. SQL 인젝션 공격 개요**

- SQL 인젝션 취약점은 웹 서버 영역의 데이터베이스로 전송되는 SQL 쿼리문을 사용자가 임의로 조작할 수 있는 경우에 발생

- 공격자는 취약점을 이용하여 데이터베이스에 저장되어 있는 다른 사용자의 개인정보 등 허가되지 않은 정보에 접근하거나 데이터를 변조

- 두 가지 사례 (회원 ID를 입력하여 회원정보를 조회하는 웹 페이지)

- SQL 쿼리문을 구성하는 웹 애플리케이션의 소스코드를 다음과 같다고 가정

$id = $\_REQUEST[ ‘id’ ];

$query = “SELECT name, email FROM users WHERE id = ‘$id’;”;

- 위 경우에서 SQL 인젝션 취약점 발생

**1) WHERE 구문 우회**

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

- 사용자는 ID가 1인 사용자 정보 요청

- 요청을 받은 웹 애플리케이션은 내부의 데이터베이스로 다음과 같은 SQL 쿼리문 전송

SELECT name, email FROM users WHERE ID=’1’

- 쿼리문에는 WHERE 조건문에서 사용자가 입력한 ID 값 1이 조건으로 입력

- 이 쿼리문은 users 테이블에서 ID가 1인 사용자의 이름과 이메일을 가져오는 쿼리문

- 쿼리문을 받은 데이터베이스는 쿼리문을 실행하여 ID가 1인 사용자 정보 반환

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

- 공격자는 원래의 ID 값 1 대신 SQL 쿼리문 조작을 위해 다음과 같이 입력

1’ or ‘1’=’1

- 이 값을 SQL 쿼리문에 그대로 삽입하면 다음과 같이 구성

SELECT name, email FROM users WHERE id =’1’ or ‘1’=’1’;

- WHERE 이하를 보면 “or” 키워드와 함께 삽입되어 항상 참이 되는 조건으로 변경

- ID 값이 1인 사용자의 데이터뿐만 아니라 모든 사용자의 이름과 이메일 전달

- 공격자가 입력할 수 있는 입력 값에 제한이 없고, 그 입력값이 SQL 쿼리문에서 그대로 실행된다면 SQL 인젝션 성공

**2) UNION 공격**

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

- 공격자는 “or” 키워드를 이용한 구문 대신에 UNION 키워드를 삽입 후 사용자 이름과 패스워드를 요청하는 SELECT 구문을 삽입

- 다음과 같은 SQL 쿼리문이 실행

SELECT name, email FROM users WHERE ID=’1’ UNION SELECT name, pw FROM users#’

- UNION을 사이에 두고 SELECT 구문 두 개가 위치

- UNION으로 합집합으로 두 개의 SELECT 구문의 결과를 모두 포함

- ID가 1인 사용자 정보 외, users 테이블의 모든 사용자의 이름과 비밀번호 같이 반환

- 쿼리문 끝에 “#”은 MySQL DB에서 “#” 뒤에 오는 내용 주석 처리 위함

- “#”이 없다면 짝이 없는 가장 끝의 문자 “ ‘ “로 인해 에러 발생 가능성

**2. SQL 인젝션 공격 실습**

**1) WHERE 구문 우회 실습**

텍스트, 전자제품, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

<DVWA level-low / SQL Injection>

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

- “User ID”에 1을 입력하면 ID가 1인 유저 admin이 출력

- SQL 인젝션 공격에 취약한지 테스트 할 수 있는 기본적인 방법은 “ ‘ “ 입력

텍스트, 스크린샷, 멀티미디어 소프트웨어, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

- 취약한 경우 위와 같은 에러 발생

- “ ‘ “을 입력하면 따옴표 쌍이 맞지 않아 형식 에러가 발생한 것 -> SQL 쿼리문에 영향을 줄 수 있음

- 처음과 마지막에 작은 따옴표를 입력하지 않는 이유는 입력 값을 문자열로 처리하는 SQL 쿼리문의 소스코드에 이미 작은 따옴표가 존재하기 때문

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 디스플레이, 번호이(가) 표시된 사진

자동 생성된 설명

- 위와 같이 입력하면 admin 사용자 외에도 다른 사용자의 정보가 같이 출력

**2) ORDER BY 및 UNION 공격 실습**

- UNION은 합집합이기 때문에 원래의 조회하는 SELECT문의 컬럼 개수와 UNION 뒤의 SELECT 문에서 요청하는 컬럼의 개수가 같아야 함.

- 공격 성공을 위해 원래의 SQL 쿼리문이 몇 개의 컬럼을 반환하는지 알아야 함.

- 컬럼의 개수는 ORDER BY 구문으로 알아낼 수 있음

- 컬럼 이름을 ORDER BY 구문으로 지정하거나 위치 인덱스를 이용하여 정렬 가능

Ex) ORDER BY 1은 첫 번째 컬럼을 기준으로 정렬. ORDER BY 2는 두 번째 컬럼을 기준으로 정렬

- 만약, 전체 컬럼 수보다 큰 수를 입력하면 에러 발생

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

- 첫 번째 컬럼을 기준으로 정렬 시도하였을 때 정상적인 표시

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 멀티미디어 소프트웨어, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

- 세 번째 컬럼을 기준으로 정렬하면 에러 발생 -> 컬럼의 총 개수 = 2

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

- “1' UNION SELECT 1,2#”을 입력하면 사용자 ID가 1인 사용자 정보와 함께 UNION으로 추가한 결과 표시

- SELECT 문 뒤에 상수를 입력하는 것이 가능 -> 해당 숫자가 나타나는 위치로 어떤 컬럼의 정보에 웹 페이지의 어느 위치에 나타나는지 쉽게 파악 가능 (모두 웹 페이지에 표시되는 것은 아님)

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

- Mysql DB에는 information\_schema라는 DB에서 DB 이름, 테이블, 컬럼 정보 관리

- information\_schema의 schemata 테이블로부터 schema\_name을 가져오는 SQL 쿼리문을 이용하여 DB 이름을 알아낼 수 있음

- 데이터베이스의 이름이 모두 First name 뒤에 출력

- DVWA에서 사용하는 데이터베이스는 dvwa라는 것을 추측 가능

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

- table\_schema가 dvwa인 조건을 주어 dvwa 테이블만 출력

- guestbook과 users라는 테이블 출력

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

- users 테이블의 컬럼 목록을 알아냄

- users와 password 컬럼 주목

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

- users 테이블에서 user와 password 컬럼을 가져온다.

- 사용자의 아이디와 비밀번호가 그대로 노출된다.

- 현재 비밀번호는 해시 값으로 암호화되어 있는 상태이므로 복호화를 진행해줘야 함

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

- 웹 브라우저의 검색창에 해시 값을 입력하자 복호화된 값을 알려주는 사이트 등장

- admin 유저의 비밀번호는 password라는 것을 알아냄

**3. 블라인드 SQL 인젝션 공격**

- 블라인드 SQL 인젝션 공격은 일반적인 SQL 인젝션 공격과 달리 출력되는 결과 값이 입력 값의 존재 여부만 알려줌

- 에러가 발생하지 않도록 처리되어 있음

- SQL 쿼리문의 처리가 어떤 방식으로 되는지 모름

- SQL 쿼리문이 실행된다는 것을 알고, 입력 값에 따라 존재 유무에 대한 결과가 달라지면 블라인드 SQL 인젝션 공격을 시도해볼 수 있음

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

<1’ AND 1=1# 입력 화면>

- 위와 같은 입력을 했을 때, 정상적인 처리가 된다는 것을 확인 -> AND 조건이 같이 처리되었다는 의미, 처리되지 않으면 비정상적인 값으로 판단하여 사용자가 없다는 메시지가 나와야 함.

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

<1’ AND 1=2# 입력 화면>

- 사용자가 없다는 출력 메시지가 뜸 -> AND 뒤의 1=2가 거짓이 되기 때문

- 입력하는 값이 SQL 쿼리문을 통해 처리되는 것을 확인

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

- 실제 소스코드를 보면 SQL 쿼리문이 사용되는 것을 확인할 수 있음.

- 위와 같은 과정을 통해 “어떤 사용자가 웹사이트에 존재하는가?”와 같은 명제의 참·거짓을 가려낼 수 있음

- 참과 거짓일 때의 결과 차이를 분석하여 어떤 정보를 알아내는 기법 -> 블라인드 SQL 인젝션 공격

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

스크린샷, 텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

텍스트, 소프트웨어, 멀티미디어 소프트웨어, 스크린샷이(가) 표시된 사진

자동 생성된 설명

- 웹사이트에 출력되는 메시지가 참과 거짓일 때, 모두 같으면 응답 시간으로 구별 시도

- SLEEP() 함수를 통해 참일 경우, 함수가 같이 동작하도록 구성

- 존재하는 사용자 ID 1을 입력했을 때는 출력이 5초간 지연되는 것을 확인

- 존재하지 않는 사용자 ID 6을 입력했을 때는 지연 없이 바로 메시지가 출력되는 것을 확인

- 블라인드 SQL 인젝션 공격의 경우 자동화 프로그램을 통해 수많은 쿼리문을 실행하여 공격

**4. sqlmap 자동화 공격**

- SQL 인젝션 공격 프로그램

- Python으로 개발된 CLI 기반 프로그램

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

- 여러 가지 옵션 중 자주 사용되는 옵션

1) -u: 필수 옵션으로 공격을 시도할 URL 지정

2) --cookie: 로그인이 필요한 경우, 로그인하고 난 다음 발급된 세션 쿠키 값 지정

3) --data: POST 요청의 폼을 공격하고자 할 때, 바디로 전달되는 데이터 지정

4) -p: 테스트할 파라미터 지정

5) --dbms: 데이터베이스의 종류를 알고 있는 경우 지정

6) --current-db: 현재 데이터베이스의 이름 알아내기

7) -D: 데이터를 입수할 데이터베이스 지정

8) -T: 데이터를 입수할 테이블 이름 지정

9) -C: 데이터를 입수할 컬럼 지정

10) --tables: 데이터베이스의 테이블 알아내기 (주로 -D 옵션과 같이 사용)

11) --columns: 데이터베이스의 컬럼들 알아내기 (주로 -D, -T 옵션과 같이 사용)

12) --dump: 데이터베이스의 정보 알아내기

- sqlmap 프로그램을 사용하기 위해 필요한 필수 옵션은 -u URL 정보

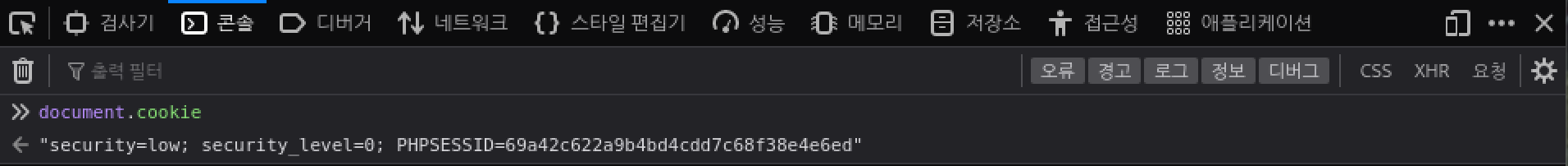
- DVWA처럼 로그인된 페이지에 SQL 인젝션 공격을 시도하는 경우 쿠키 정보도 필요

텍스트, 전자제품, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

<User ID에 임의의 값 1을 입력한 화면>

- 주소창에 요청을 전송한 URL이 표시됨 -> -u 옵션에 사용



- 쿠키 값은 개발자 도구의 콘솔 탭에서 “document.cookie” 입력

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 소프트웨어, 폰트이(가) 표시된 사진

자동 생성된 설명

<터미널에서 값을 입력한 모습>

- 옵션 값 중 &나 # 같은 특수문자가 있으면 터미널에서 명령어를 내릴 때 특수한 기능 문자로 사용되기 떄문에 따옴표로 묶어줘야 함

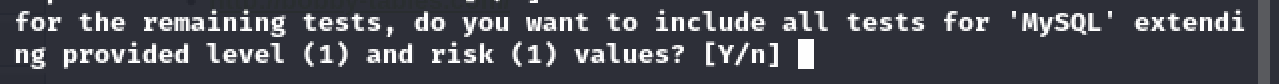
- 표시된 메시지 중 굵은 글씨로 표시된 부분

1) id 파라미터가 SQL 인젝션 공격 가능성이 있다.

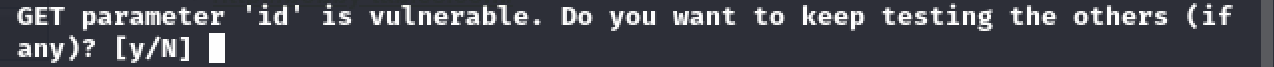
2) id 파라미터가 크로스 사이트 스크립팅 공격 가능성이 있다.

3) DBMS의 종류가 MySQL로 보여 다른 DBMS에 대한 테스트 제외를 원하는지

(MySQL인 것을 알고 있다면 제외)



- MySQL 관련 테스트를 모두 수행할 것인지 물어보는 질문



- id 파라미터가 취약하다는 결과와 함께 다른 파라미터를 계속 테스트하기를 원하는지 물어보는 질문

- 이미 id 파라미터가 취약하다는 것을 알았으므로 N을 선택

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

<최종적으로 출력된 결과>

- 160개의 요청을 사용하여 SQL 인젝션 취약점이 가능한 부분을 찾아냄

- id 파라미터가 다음 종류의 SQL 인젝션 공격에 취약

1) 에러 기반(error-based)의 SQL 인젝션

2) 참/거짓 기반(Boolean-based)의 블라인드 SQL 인젝션

3) SLEEP을 이용한 시간 기반의 블라인드 SQL 인젝션 공격

4) 컬럼 2개를 이용한 UNION 쿼리

- 백엔드의 DBMS는 MySQL이고 버전은 4.1 이상

- 웹 서버의 운영체제는 우분투 8.04 버전

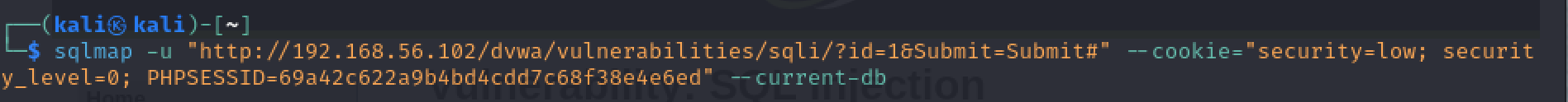
- 사용된 웹 관련 기술은 PHP 5.2.4, 아파치 2.2.8

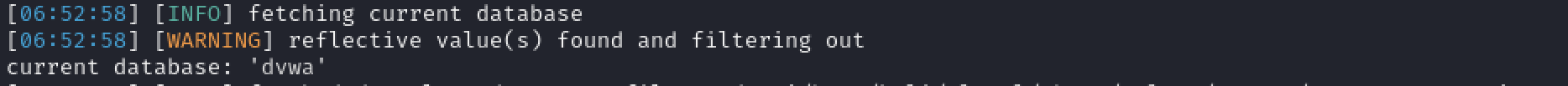
- /home/kali/.local/share/sqlmap/output/192.168.56.102 아래에 데이터 저장됨

- sqlmap 프로그램을 이용해 위와 같이 URL에 포함된 각 파라미터를 자동으로 테스트하여 SQL 인젝션 공격 가능 여부 확인

- 웹 애플리케이션 개발 언어와 버전, 웹 서버, DBMS 버전 등의 유용한 정보도 확인 가능

- 실행 결과를 저장하여 데이터 수집 관련 명령어 추가 실행시, 기존의 결과를 참조하여 빠르게 수집 가능

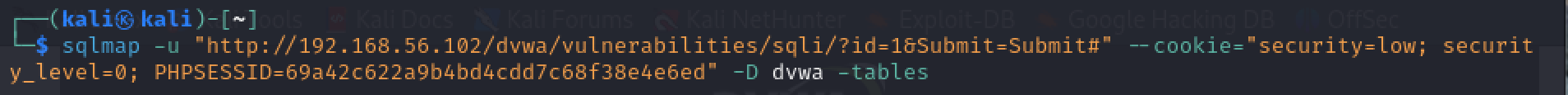




- 이전에 입력한 명령 뒤에 --current-db 옵션을 주어 현재 데이터베이스의 이름을 알아낼 수 있음.

- “dvwa”라는 데이터베이스 이름을 알아냄

- 만약, 데이터베이스 이름이 “None”이라고 표시되며 찾지 못하면 DVWA 로그인이 종료되었는지 확인  
 (일정 시간 이후, 로그인이 풀리기 때문. 재로그인시, 새로운 쿠키 값 사용)

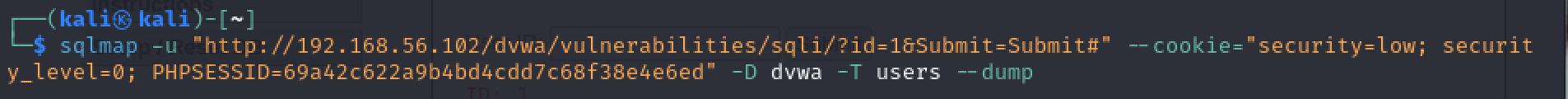


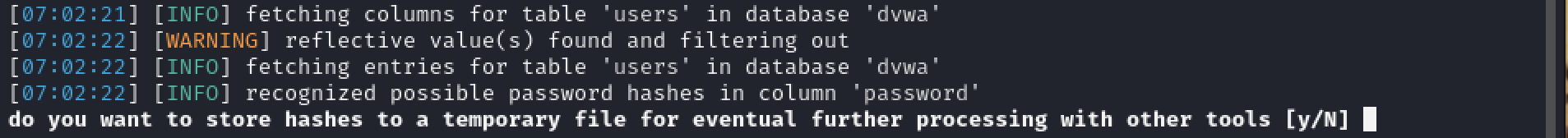
텍스트, 소프트웨어, 멀티미디어 소프트웨어, 폰트이(가) 표시된 사진

자동 생성된 설명

- dvwa 데이터베이스의 테이블 이름을 알아내기 위해 이전 명령에서 -D 옵션으로 dvwa라는 데이터베이스 이름을 설정해주고, --tables 옵션을 사용하여 테이블을 알아냄.

- “guestbook”과 “users”라는 테이블이 있는 것을 확인





- “users” 테이블을 살펴보기 위해, 이전 명령에서 –tables 옵션 대시, -T 옵션으로 “users” 테이블을 지정, --dump 옵션을 이용해 테이블의 내용 추출

- 진행 과정 중 sqlmap 프로그램이 패스워드 해시 관련 정보를 찾으면 몇 가지 패스워드 크래킹 관련 질문함

- 위 질문은 나중에 다른 프로그램을 이용해 패스워드 크래킹을 진행할 수 있도록 패스워드 해시를 임시파일로 저장할 것인지 묻는 질문

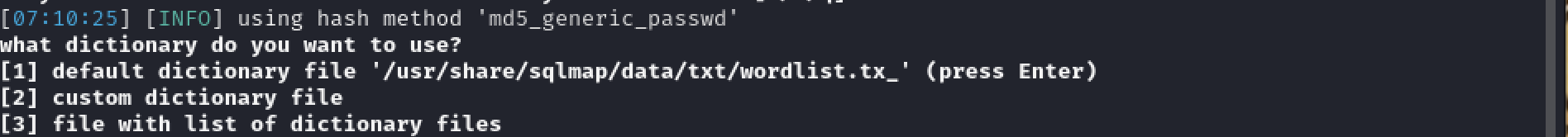
- sqlmap은 패스워드 크래킹 수행 기능 내장. 내장된 기능 사용시, 기본값 N 입력



- 위 질문은 딕셔너리 공격을 이용하여 패스워드 해시를 크래킹할 것인지 묻는 질문

- 딕셔너리 공격 = 패스워드 목록으로 구성된 파일을 이용해 패스워드를 알아내는 것

- Y 입력



- 어떤 사전 파일을 사용할 것인지 묻는 질문

- [1] = sqlmap에 내장된 사전 파일 사용

- [2] = 칼리 리눅스의 사전 파일이나, 다른 파일

- [3] = 복수의 사전 파일 사용

- 해당 과정은 1번 사용



- 공통 패스워드 접미사를 사용할 것인지 묻는 질문

- 접미사 기법 = 공통된 문자열을 기본으로 몇 가지 다른 문자를 추가하여 패스워드를 만드는 기법

- 많은 웹 사용자들은 여러 개의 웹사이트 패스워드를 동일한 패스워드로 사용 (위험!)

- 기본값 N 사용

텍스트, 스크린샷, 소프트웨어, 폰트이(가) 표시된 사진

자동 생성된 설명

- “users” 테이블 내의 모든 정보 출력된 모습

- 사용자 ID와 패스워드 해시, 크래킹된 패스워드 출력

**5. SQL 인젝션 공격 대응**

- SQL 인젝션 공격과 같은 모든 파라미터 입력값 조작으로 이루어지는 공격은 입력값 검증을 통해 대응

(화이트리스트 검증 등)

- 시큐어 코딩을 통해 좀 더 구체적으로 대응 가능

* 사용자가 입력한 값은 SQL 쿼리문에서 오직 데이터로만 사용되어야 함.
* SQL 쿼리문의 구조에 영향을 줄 수 없도록 해야 함.
* 다음과 같은 쿼리문은 SQL 인젝션에 취약

|  |
| --- |
| $query = “SELECT first\_name, last\_name FROM users WHERE user\_id=’$id’;”; |

쿼리문을 구성하고 실행하는 방법을 파라미터 쿼리로 변경해야 함.

|  |
| --- |
|  |

- 10~12번째 줄에서 prepare(), bindParam(), execute()를 각각 호출하여 쿼리문 실행하는 것을 확인

- prepare() 함수에서 미리 실행할 쿼리문의 형태 작성 -> prepared statement

- 사용자 입력값이 들어갈 id 부분은 bindParam() 함수에서 설정

-> 사용자가 입력한 값이 쿼리문의 일부가 될 수 없고 온전히 데이터로만 처리

-> or 또는 UNION 등과 같은 키워드가 입력되어도 무의미한 문자열로 인식

- 결과적으로 사용자가 SQL 쿼리문을 조작할 방법이 없어 SQL 인젝션 공격 방지

- 자바에서는 prepareStatement() 함수, 닷넷에서는 SqlCommand() 함수 등 다른 프로그래밍 언어에서도 파라미터 쿼리와 관련하여 제공하는 함수 많음.