이스케이프 처리 함수, 그리고 개념
PHP 환경에서 이스케이프 처리 시 사용되는 addslashes() 함수 혹은 mysql_real_escape_string() 함수(객체 사용 시 real_escape_string()), magic_quotes_gpc 가 있으며, 이는 싱글 쿼터, 더블 쿼터, 역 슬러시 같은 문자를 이스케이프 처리 시켜서 문자의 기능을 하지 못하게 한다. 예를 들어, ' 문자는 \', " 문자는 \", \ 문자는 \\로 처리가 된다.

이스케이프 처리 용도
SQL 인젝션 취약점을 방어하기 위해 대부분 사용되며, 사용자 입력 값에 대해 이스케이프 처리 후 DB 쿼리에 바인딩 하여 DB에 질의를 한다.

멀티 바이트 문자 처리 버그
PHP에서 멀티 바이트 문자 처리 시 버그가 발생되는데 멀티 바이트 문자 뒤에 0x5C, %5C 문자인 \ 문자가 있으면 하나의 문자로 처리가 된다. 이렇게 되면 이스케이프 처리가 된 \', \", \\ 문자들 앞에 멀티 바이트 문자가 붙을 경우
이스케이프 처리가 무효화된다는 것이다. 즉, 0xbf5C27는 0xbf5c가 한문자, 0x27 한문자로 해석이 된다.

취약한 환경
그러나 모든 환경에서 발생되는 것은 아니며, "GBK 인코딩을 사용할 경우", "EUC-KR을 UTF-8로 인코딩 변환을 할 경우" 발생된다.

취약점 실습
간단한 테스트를 통해 본 취약점을 살펴보자.
테스트 환경은 아래와 같다.
운영체제 : Windows 10
PHP 버전 : PHP 5.2.12
MYSQL 버전 : 5.1.41
취약한 환경 예시(1)
먼저 인코딩 변환 시 취약한 예시를 먼저 살펴보자.

아래는 취약한 소스코드이다. 물론 개발 설계 시부터 인코딩을 통일 시키는 것이 기본이지만, 간혹 웹상에서 사용되는 인코딩 방식과 DB에서 사용되는 인코딩 방식이 다를 경우가 있다. 이럴 경우 인코딩 변환 함수를 통해 변환 후 DB에 저장한다.

```
<?
    $db_conn = new mysqli ("localhost", "아이디", "패스워드", "DB명");
    $id = $db_conn->real_escape_string($_GET["id"]);
    $pw = md5($_GET["pw"]);

    // EUC-KR을 UTF-8로 인코딩 변환
    $id = mb_convert_encoding($id, "utf-8", "euc-kr");
```

```
    $pw = mb_convert_encoding($pw, "utf-8", "euc-kr");

    $query = "select * from members where id='{$id}' and pw='{$pw}'";

    $temp = $db_conn->query($query) or die($db_conn->error);
    $info = $temp->fetch_assoc();

    if(empty($info)) {
        echo "Login Failed~!";
    } else {
        echo "Login Success~!";
    }
?>
```

테스트를 위한 테이블 생성 쿼리 및 테이터 삽입 쿼리이다.

```
create table members(
id varchar(20),
pw varchar(50)
);
insert into members values('admin', md5('admin123'));
```

환경 구성 후 GET 방식으로 아이디, 패스워드를 입력할 경우 정상적으로 로그인이 되는 것을 알 수 있다.

그림 1. 로그인 화면

앞서 설명한 바와 같이 싱글 쿼터(%27) 앞에 멀티 바이트 문자(%a1~%fe) 입력 시 다음과 같이 DB 에러가 발생되는 것을 알 수 있다. 이는 멀티 바이트 문자로 인해 싱글 쿼터 앞의 역슬러시가 무효화되고 결국 싱글 쿼터만 남게 되어 발생된 에러이다.

그림 2. 멀티 바이트 문자 삽입 시 DB 에러 발생

강제적으로 결과를 참으로 유도하여 다음과 같이 인증 우회가 되는 것을 알 수 있다.

그림 3. 멀티 바이트 문자 삽입을 통한 이스케이프 처리 우회

취약한 환경 예시 (2)

다음은 GBK 인코딩 시 발생 예시이다.

아래는 취약한 소스코드이다.

```
<?
    $db_conn = new mysqli ("localhost", "아이디", "패스워드", "DB명");
    $id = $db_conn->real_escape_string($_GET["id"]);
    $pw = md5($_GET["pw"]);

    $query = "select * from members where id='{$id}' and pw='{$pw}'";
```

```
    // GBK 인코딩
    $db_conn->query("SET NAMES gbk");
    $temp = $db_conn->query($query) or die($db_conn->error);
    $info = $temp->fetch_assoc();

    if(empty($info)) {
        echo "Login Failed~!";
    } else {
        echo "Login Success~!";
    }
?>
```

테스트를 위한 테이블 생성 쿼리 및 테이터 삽입 쿼리이다. 이때 인코딩을 GBK로 설정한다.

```
create table members(
id varchar(20) character set gbk,
pw varchar(50) character set gbk
);
insert into members values('admin', md5('admin123'));
```

멀티 바이트 문자 사용 시 동일하게 우회가 되는 것을 알 수 있다.

그림 4. 멀티 바이트 문자 삽입을 통한 이스케이프 처리 우회

그리고 대응 방안

SQL 인젝션에서 가장 안전한 방식은 역시 Prepared Statement 방식일 것이다. 취약한 환경 (1), (2) 소스코드에 대해 Prepared 적용 후 테스트를 진행해 보자.

취약한 환경 예시(1)에 대한 안전한 소스코드

```
<?
    $db_conn = new mysqli ("localhost", "아이디", "패스워드", "DB명");
    $id = $_GET["id"];
    $pw = md5($_GET["pw"]);

    // EUC-KR을 UTF-8로 인코딩 변환
    $id = mb_convert_encoding($id, "utf-8", "euc-kr");
    $pw = mb_convert_encoding($pw, "utf-8", "euc-kr");
    $query = "select * from members where id='{$id}' and pw='{$pw}'";

    $stmt = $db_conn->stmt_init();
    $stmt->prepare("select * from members where id=? and pw=?");
    $stmt->bind_param('ss', $id, $pw);

    $stmt->execute();
```

```
    $stmt->bind_result($result_id, $result_pw);
    $stmt->fetch();

    if(empty($result_id)) {
        echo "Login Failed~!";
    } else {
        echo "Login Success~!";
    }
?>
```

멀티 바이트 문자를 삽입 시 우회가 되지 않는 것을 알 수 있다.

그림 5. 취약 환경 예시(1) 대응 방안 적용

취약 환경 예시(2)에 대한 안전한 소스코드

```
<?
    $db_conn = new mysqli ("localhost", "아이디", "패스워드", "DB명");
    $id = $_GET["id"];
    $pw = md5($_GET["pw"]);

    $query = "select * from members where id='{$id}' and pw='{$pw}'";

    $db_conn->query("SET NAMES gbk");
    $stmt = $db_conn->stmt_init();
    $stmt->prepare("select * from members where id=? and pw=?");
    $stmt->bind_param('ss', $id, $pw);

    $stmt->execute();
    $stmt->bind_result($result_id, $result_pw);
    $stmt->fetch();

    if(empty($result_id)) {
        echo "Login Failed~!";
    } else {
        echo "Login Success~!";
    }
?>
```

멀티 바이트 문자를 삽입 시 우회가 되지 않는 것을 알 수 있다.

그림 6. 취약 환경 예시(2) 대응 방안 적용

예전부터 이와 관련하여 많은 글들이 온라인상에 존재한다. 그런데 취약점들에 대한 설명만 많을 뿐 대응 방안에 대한 언급은 하지 않거나, addslashes() 함수 대신 mysql_real_escape_string() 함수를 사용이 안전하다는 글들을 많이 볼 수 있다.

그러나 앞서서 살펴본 바와 같이 addslashes() 함수에 국한된 문제가 아니란 것을 알 수 있었으며, Prepared Statement 방식이 근본적인 방어법이란 것을 알 수 있었다.