

## SQL Injection 2.0

### Protecting Web Sites from Automated and Advanced SQL Injection

SQL Injection continues to be one of the most predominant Web application threats. In the first half of 2008, SQL injection was the number one attack vector for malware, having compromised tens of thousands of Web sites including those of some high profile companies<sup>1</sup>.

Considering the widespread availability of valuable data on the Web, the popularity of ecommerce and dependency on the Web for all kinds of information, attackers are motivated to implement faster, more advanced SQL injection methods to launch high profile, widespread attacks on targeted Web sites.

This paper provides an overview of SQL Injection 2.0, the new wave of SQL injection threats that dominates the application landscape. Specific attack techniques described include automation of SQL injection through usage of tools and popular search engines, SQL injection for Web site defacement, malware distribution and Denial of Service (DoS) attacks, and direct database SQL injection. There is also an overview of mitigation techniques organizations can use to address these types of application threats.

Most of the information contained herewith is based on the advanced knowledge base of Imperva's Application Defense Center, a premier research organization for security analysis, vulnerability discovery, and compliance expertise.

---

<sup>1</sup> <http://www.securityfocus.com/brief/729>

## Overview of SQL Injection

This white paper assumes you have a working knowledge of SQL injection. However, a brief refresher on SQL injection is provided here. For additional information, you may refer to previous Imperva white papers on this topic: "[SecureSphere SQL Injection Protection White Paper](#)" and "[Blindfolded SQL Injection](#)."

SQL injection is an attack technique that takes advantage of non-validated input vulnerabilities to pass SQL commands through a Web application for execution by a back-end database. Most SQL injection attacks are executed through an application that accepts user-supplied input for query parameters. The attacker supplies a carefully crafted string within a form field or URL parameter to form a new query with results that are very different from what the application developer intended.

For example, consider a login form on a Web site that takes a username and password parameter to enable access to a certain section of the Web site by validating the username and password against entries in the back-end database. A very simple attack may be possible by simply providing something, like 'a' OR 1=1 -- in the username field. Here we illustrate the SQL query that is generated when a user logs into a Web application:

```
SELECT * FROM users WHERE username = '$uid' AND password = '$pwd';
```

Suppose the hacker enters the username as 'a' OR 1=1 --. Then the SQL query becomes

```
SELECT * FROM users WHERE username = 'a' OR 1=1 --' AND password = 'pwd'
```

Because the double hyphens character is interpreted as the beginning of a comment by the SQL server, everything after '--' is ignored. So the query effectively becomes:

```
SELECT * FROM users WHERE username = 'a' OR 1=1;
```

Since 1=1 is always true, the condition is always true and the login is always validated. Once the attacker has gained access to the application, he may then perform reconnaissance to find out the structure of the database, learn about sensitive data tables and ultimately steal sensitive information.

While the example above shows how SQL injection can be abused to gain unauthorized application access, most SQL Injection attacks are aimed at gaining unauthorized access to data within the backend database.

## Automation of SQL Injection

The sheer number of compromised servers this year leaves no doubt that the hackers have stepped up their efficiency using automated tools. Automation, from a hacker's viewpoint, may expedite a number of distinct processes: discovery of attack targets, construction of a working exploit and extraction of data (in particular when a binary search technique is required). Two approaches are increasingly combined together for providing automation in SQL injection attacks: dedicated desktop tools and search engine hacking.

### SQL Injection through Automation with Tools

Tools are mainly used for automating two tasks within the SQL injection attack process: construction of a working exploit and extracting information. Having automated tools at hand not only dramatically increase the efficiency of an experienced hacker but also extends the potential attacker population. Software tools equipped with a crisp and appealing GUI, allow inexperienced "script kiddies" to mount complex SQL injection attacks against targets previously requiring in-depth knowledge of hacking techniques (e.g. blind SQL injection) and acquaintance with the backend database vendor technology (including structure of data dictionary, specific SQL strand, etc.). The overall effect of automation can be clearly depicted by looking at information regarding SQL incidents.

With the daunting, time-consuming task of creating a working exploit taken care of by automated tools the time frame between target discovery and full exploitation is greatly diminished. This has two favorable effects for potential attackers. First and foremost, the risk for detection is lower, as the attack duration shortens. This is as true in real life heists as it is for the cyberspace ones. Secondly, it is much easier for attackers to find a window of opportunity to launch their attack, even against systems that are carefully watched and not always available for remote exploit.

Not all tools are created alike. Some are targeted towards specific vendor brands; some perform only a single task while others are more comprehensive. Some are command-line tools while others have a lavish GUI. Generally speaking, there are now tools that provide the following capabilities:

- » Given an injection point (URL and parameters), create a working exploit.
- » Achieve the above goal even when blind SQL injection is required
- » Identify type of backend database
- » Extract metadata from backend database (list of tables, columns and users)
- » Extract information through GUI
- » Extract information through binary search
- » Compromise remote server by deploying backdoors

These are some of the tools being employed by hackers to automate SQL injection attacks:

- » **Priamos** can be used to find vulnerabilities in applications. The tool requires manual configuration of the injection point, but thereafter, the user can apply the vulnerable character string into the injector module in Priamos to retrieve all database names, tables and column data. It presents a GUI to the user for direct database interaction.
- » **Power Injector** does some automatic detection of the injection point and provides a GUI to take the SQL injection attack further from that point in order to extract desired information from the database. Its power lies in its capacity to automate tedious blindfolded SQL injection methods with several threads.
- » **SQL Ninja** can be used to exploit SQL injection vulnerabilities on a Web application that uses Microsoft SQL Server as its back-end database. It requires manual configuration of the injection point. It is dedicated to exploiting injections into a specific stored procedure of the database server called Xp\_cmdshell.
- » **SQL Map** does some of its automated detection of the injection point based on Google searches. It performs automatic blind SQL injection, capable of capturing an active database management system fingerprint, enumerating entire remote databases and much more.

These are just a few of the more common and publicized tools, but there are many more tools that have been and continue to be developed by professional attackers with the purpose of automating SQL injection attacks.

An attacker with the appropriate toolbox and an idea about potential injection points in an application needs no more than click and point to get away with credit card numbers, social security numbers and other personal information stored in a backend database.

## SQL Injection through Automation with Search Engines

While tools are mostly used for reducing the time from vulnerability detection to exploit, another type of automation is required for quickly detecting large quantities of potential targets. The growing trend is the increased abuse of Internet search engines for spotting potential SQL injection targets.

Using specially crafted search terms, an attacker can quickly get a list of applications, accessible through the Internet, that are potentially vulnerable to a specific instance of SQL injection. Search terms can be created to detect a specific vulnerable application (e.g. one that was identified as vulnerable by a CVE entry) or a specific error code indicative of SQL injection vulnerability. The results include not only a domain name for the target application but the actual injection points within that application. Thus, an attack can be quickly mounted by taking the information from the search engine results and feeding them to one of the automated tools mentioned above.

Attackers can craft their own search terms but can also use resources available on the Internet that provide lists of search terms for almost any type of attack, SQL injection included. One of the most famous resources in this respect is Johnny Long's Google Hacking Database maintained at [johnny.ihackstuff.com](http://johnny.ihackstuff.com). The search terms on Johnny's database, called "Google Dorks" are categorized by attack type and the interface to the database is very intuitive.

Further increasing the potential effect of the Google Hacking technique on improving the overall efficiency of an attack is the use of automated search tools. These simple robots take as input a list of search terms, run them all at once and give back a formatted set of results. Attackers can use their own tools (creating such a tool can be a really simple programming exercise) or use available tools from the Internet such as Goolag Scanner (by The Cult of the Dead Cow). This tool in particular takes its list of search terms from Johnny Long's database and makes them available to the non-technical user through a pleasant, easy-to-use GUI.

Earlier this year, hackers used search engines to quickly find ASP and ASPX (.net) pages that accepted a set of carefully chosen input parameters (e.g. an article ID, product ID and others) which they suspected are indicative of a number of poorly written modules available worldwide (most probably code samples widely used by programmers). This served as the injection point for the successful upload of SQL injection code in over 500,000 Web pages. The code caused a user who visited the site to be redirected to a malware serving site. Affected companies were forced to quickly investigate and sanitize the Web sites by removing the injected code<sup>2</sup>.

## SQL Injection: Going Beyond Data Theft

Although SQL injection has most commonly been associated with the extraction of valuable data through Web applications, it is also increasingly being utilized as a platform for launching other types of Web based attacks including Web site defacement, malware distribution (aka "Drive-by download attacks"), and Denial of Service (DoS).

### SQL Injection for Web site defacement

Web site defacement traditionally occurred when a hacker obtained administrative privileges to a Web site and then altered the content of the Web site with potentially offensive or erroneous graphics and text. While Web site owners have bolstered the security of Web configuration tools, malicious users have discovered a new technique to deface Web sites: SQL injection.

In 2007, there were several high profile incidents in which SQL injection was used for Web site defacement. In fact, the Microsoft UK<sup>3</sup> site and the United Nations<sup>4</sup> (UN) sites were defaced within just months of each other in 2007. This type of attack is made possible when the injection point allows not only tampering with the criteria of a SELECT statement but also appending additional SQL statements such as INSERT or UPDATE. In particular, an attacker can construct an UPDATE statement to tamper with the contents of database columns that are later embedded in HTML pages. The attacker would replace the original content of such columns with HTML code to either change the appearance of a page (by embedding offensive images) or silently redirecting a client to a malware hosting server (by embedding IFRAME tags).

Because this type of Web site defacement affects the backend database rather than the static Web application files, even if there is a static change tracking system or change management system being used, neither of these mechanisms would detect the attack.

### SQL Injection for Malware Distribution

Over the past year, there have been a number of publicized malware distribution incidents due to SQL injection attacks. Many of the affected Web sites were high profile, high traffic volume based sites.

SQL injection for malware distribution is an evolution of the same technique that is used for Web site defacement. This time however, the appended HTML element is not of a visible nature.

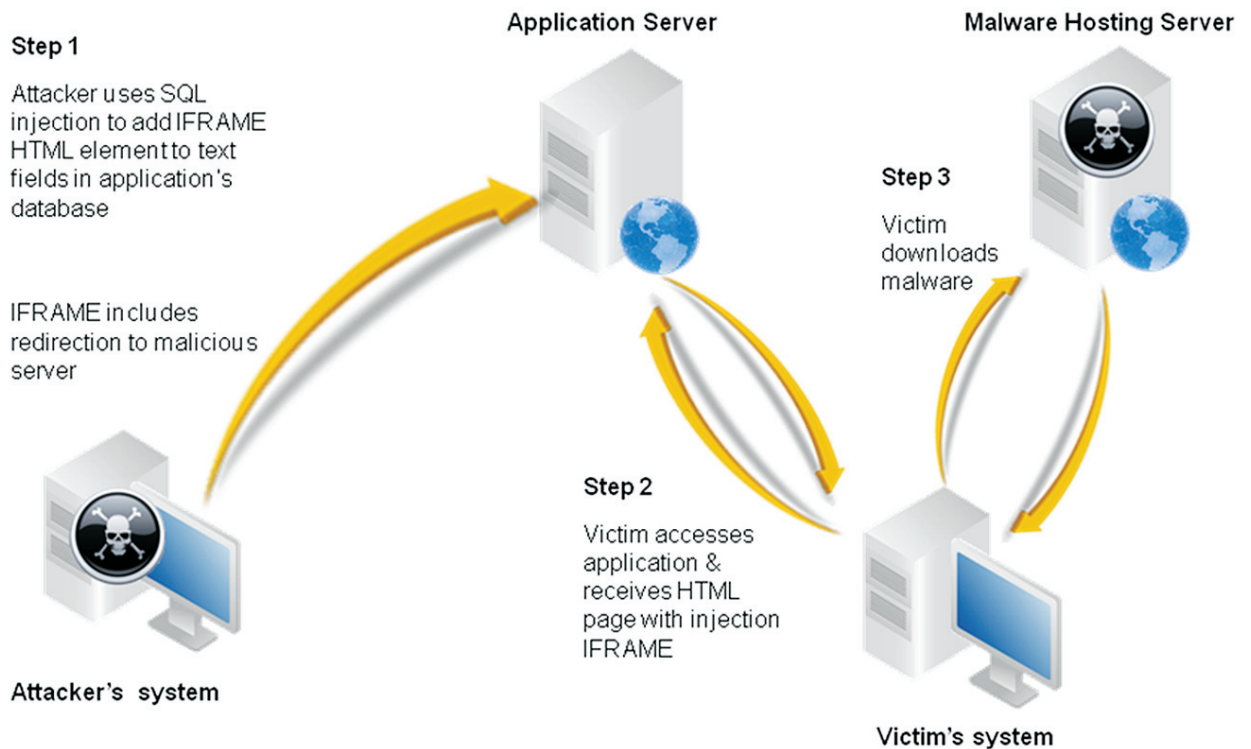
Below we illustrate how this type of attack can take place. In this example, we will focus on Web site defacement, not how the SQL injection point was found in the application.

---

<sup>2</sup> [http://www.pcworld.com/article/145151/huge\\_web\\_hack\\_attack\\_infects\\_500000\\_pages.html](http://www.pcworld.com/article/145151/huge_web_hack_attack_infects_500000_pages.html)

<sup>3</sup> Microsoft UK site defacement: [http://news.cnet.com/2100-7349\\_3-6194705.html](http://news.cnet.com/2100-7349_3-6194705.html)

<sup>4</sup> United Nations site defacement: [http://www.pcworld.com/article/135879/hackers\\_deface\\_un\\_site.html](http://www.pcworld.com/article/135879/hackers_deface_un_site.html)

**Step 1:**

The attacker modifies a SQL query by adding an IFRAME HTML element to all or some of the text fields in the application's database. The IFRAME includes a redirection to a server holding malware.

**Step 2:**

Once a victim accesses the compromised application, he would receive an HTML page with the injection IFRAME.

**Step 3:**

The victim downloads malware from the attacker controlled server. Once the user visits the malware serving site, Trojans or keystroke loggers are downloaded and automatically installed on the user's machine. Once a keystroke logger software application is on the machine, any personal information such as credit card numbers, bank passwords, or social security numbers can easily be extracted and subsequently misused by the attacker.

What makes this attack particularly difficult to detect is that it is executed without leaving behind any visual trace. When the HTML page is rendered by the victim's Web browser, what happens behind the scenes is that the victim ends up downloading malware from the attacker-controlled server thinking that it is part of the legitimate Web application.

As part of a malware distribution attack, infecting all items in the database is not necessary. The attacker may choose to infect a specific record that is known to be displayed for all users. Infecting all items of a database is accomplished through the combination of a new attack target (data integrity) with a new method (automation).

Because of SQL injection automation tools and search engines, hackers have been able to successfully inject malware links in hundreds of thousands of Web sites. In fact, in January 2008 alone, at least 70,000 Web sites had been affected by one particular automated SQL injection attack that exploited several common vulnerabilities of these sites, including a Microsoft SQL server vulnerability<sup>5</sup>.

<sup>5</sup> Multiple Web sites affected by SQL injection based malware attack:

<http://www.scmagazineus.com/Mass-SQL-injection-attack-compromises-70000-websites/article/100497/>

## SQL Injection for DoS

Another application threat related to SQL injection is Denial of Service (DoS), which, in its most extreme form, can bring the Web application to a halt by shutting down its backend database. It takes place when an attacker appends the SHUTDOWN command to a SQL statement, or when the attacker creates complex queries over self-joins of large database tables with the intention of sending the database into time consuming loops over lots of data. This ends up consuming precious CPU time within the database servers. These attack techniques are generally well known – they require true SQL injection and usually such complex queries require the existence of very large tables within the target database.

What is relatively new is the use of search fields that perform “LIKE” comparisons of the search terms to induce DoS without users actually inputting an additional command or clause. So even if the application code is written correctly, but the application fetches results based on just an approximated search of the backend database, the application might be vulnerable. This phenomenon is due to the way commercial databases implement the LIKE operator. In particular this operator gives special “wild card” meaning to some characters (e.g., “%”, “[ ]”, “[^]”, etc.) up to the point of providing full regular expression capabilities.

One of the most obvious implications of using specially crafted approximated search expressions is avoiding the use of database indexes. This, by itself, consumes more processing resources in the database server since an entire table needs to be scanned sequentially rather than a small index being partially traversed. In addition, the database engine applies complex regular expressions to every row in the table. Due to very long processing times, the application slows down because the application server’s resources are being fully consumed. Some studies have shown that by using even a relatively short search term of a few characters over a relatively small table with just a couple thousand records, the search can easily consume all of the CPU’s power for thirty seconds or more. This is clearly a relatively simple, yet powerful method that can be used to induce DoS in an application even if the application is not directly susceptible to SQL injection.

## Direct Database SQL Injection

Another type of attack, one that is less prevalent in real world scenarios, but nevertheless, can cause major damage to an organization if it succeeds, is a direct database SQL injection attack through database stored procedures. A stored procedure is a collection of procedural SQL statements put together for the completion of a single logical task and stored in the database server under a distinctive name. A person who writes a stored procedure can grant execution privileges to other database users without having to grant them any privileges over the underlying objects. An implication of this model is that the code within the stored procedure is (usually) executed in the security context of the procedure writer rather than the procedure caller. Hence if a stored procedure is vulnerable to SQL injection, an attacker with execute privileges on that procedure can inject SQL statements that will be executed with the privileges of the procedure writer, most often an administrative user. In the following code snippet we can see that through the parameter RESULT\_TABLE an attacker can inject changes to the update statement.

```

PROCEDURE VALIDATE_LAYER(LAYER IN VARCHAR2, RESULT_TABLE IN
VARCHAR2) IS
...
UPDATE_STR := 'INSERT INTO ' || RESULT_TABLE || ' VALUES(:gid,
:gid_result)' ;
...
LOOP
    BEGIN
        FETCH QUERY_CRS INTO GID ;
        EXIT WHEN QUERY_CRS%NOTFOUND ;
        GID_RESULT := MDSYS.MD2.VALIDATE_GEOM(UPPER(
        DBMS_ASSERT.QUALIFIED_SQL_NAME(LAYER) ), GID, NULL);
        EXECUTE IMMEDIATE UPDATE_STR USING GID, GID_RESULT ;
    
```



Commercial database platforms offer more and more functionality to their users through stored procedures, granting by default execution privileges to any database user over a growing set of those. Consequently, the risk of an internal user gaining elevated privileges is growing higher. However, direct database SQL injection poses risk not only with respect to internal threats. Many Web applications make use of a database stored procedure to complete their tasks. As a consequence a SQL injection vulnerability in a stored procedure can be invoked through a Web facing application exposing it to external threats. One such example is a vulnerability reported by David Litchfield earlier in 2008, coded CVE-2008-2589. It is a SQL injection vulnerability in a built-in Oracle stored procedure called `WWV_RENDER_REPORT`. The stored procedure forms part of the Oracle Portal software suite and is accessible through a Portal URL. Therefore external attackers can easily abuse the stored procedure to gain unauthorized access to all Portal data.

Recent research<sup>6</sup> has also shown that while traditionally direct database SQL injection vulnerabilities are attributed to textual parameters they can also be attributed to parameters of type DATE, and even to procedures with no parameters that use the `SYSDATE` function to get the current date.

## Mitigation Techniques

When dealing with SQL injection vulnerabilities, in a perfect world, organizations would: fix the application code for discovered vulnerabilities by using input sanitation and using prepared statements with bind variables, test that the application is no longer vulnerable, and then redeploy the application. The drawback of this approach is that it takes a lot of time to fix application code. Also, in some cases, the application code is not even available because the original application developers no longer work for the organization or the application was developed by subcontractors.

### Code Fix is Not a Solution for Every Organization

Every organization typically has a process that it must follow for fixing SQL injection and other types of application vulnerabilities. Finding the vulnerabilities is only the beginning. The software code fix process involves working in synchronization with application developers and the QA team and coming to a consensus on a software development process designed to prevent developers from introducing flaws into critical applications. The following is an example of such a process:

1. Investigate the application and identify the vulnerable code.
2. Architect a solution in the source code to fix the vulnerability.
3. Organizations with a change management process in place must also document the fix, perform a risk assessment and get sign off before they can implement any changes to the application.
4. Code the fix into the application.
5. Test the application to make sure the fix solved the problem and didn't introduce new vulnerabilities.
6. Follow a maintenance window – most organizations have a specific window and generally the larger the organization, the smaller the window is. In certain cases, for critical vulnerabilities, an application may have to be brought down for a very brief period and must be able to regress during that period of the deployment (when the application is not running).
7. Deploy the application after it has been fixed for the vulnerability.

Clearly, the above process is very resource and time consuming and following it for each and every vulnerability as soon as it is discovered is not a realistic option.

A serious drawback of the code fix approach is that during the entire length of the process the application remains vulnerable to attacks. It is only after the application has been redeployed in step seven that the application is protected against that particular vulnerability. Another drawback is that changing the code may introduce new vulnerabilities, for which the code would have to be re-scanned, and possibly changed and re-tested.

---

<sup>6</sup> <http://www.databasesecurity.com/dbsec/lateral-sql-injection.pdf>

Another important point is that certain organizations use packaged business applications such as Oracle E-Business Suite, SAP and PeopleSoft, and must depend on the vendors themselves to release a patch for the vulnerability fix. The organizations themselves cannot even perform a code fix. Therefore, they would probably need an alternative security solution to protect their applications on an ongoing basis.

### **Traditional Security Methods are Insufficient**

Network firewalls and Intrusion Prevention Systems (IPS) attempt to identify SQL injection via traditional signature-based protections. The most common way of detecting SQL injection attacks is by looking for SQL signatures in the incoming HTTP stream. For example, looking for SQL commands such as UNION, SELECT or xp\_. The problem with this approach is the high rate of false positives. Most SQL commands are legitimate words that could normally appear in the incoming HTTP stream. Ultimately, the security administrator will either disable or ignore any SQL alert reported. In order to overcome this problem to an extent, the product should learn where it should and shouldn't expect SQL signatures to appear. The ability to discern parameter values from the entire HTTP request and the ability to handle various encoding scenarios are an important requirement. Certain Web application firewall products address this need.

Some organizations consider usage of vulnerability scanning technology, either static (code review) or dynamic (application vulnerability scanning), to address SQL injection and other application vulnerabilities. It should be noted that although these types of vulnerability scanning technologies assist in discovering vulnerabilities, they still leave the organizations with the problem of mitigating them.

To address SQL injection and other advanced application threats, a different solution, a Web application firewall, is required. Unlike network firewalls and IPS solutions, a Web application firewall understands how applications work and provides specific protection for applications against attacks based on both known and unknown vulnerabilities. Even an organization that has the resources available to fix application code will benefit from deploying a Web application firewall. With a Web application firewall deployed in front of critical application and Web servers, the organization gets a larger window within which to fix its application code, rather than having to schedule rush fixes each time a new application threat or variant of a known application threat is introduced. In addition to deploying a Web application firewall, the organization can schedule code fixes and third party patching of the most critical areas of the application, much like an organization would do with any other functional flow.

One of the greatest challenges organizations face in SQL injection security solutions is properly tracking actual users. The fundamental problem is that almost all Web applications use "pool accounts" so that while users are authenticated to the Web application, they appear only as a single pooled account user to the database. Security and compliance mandates make it critical to have a solution that will uniquely identify the end-users who are performing activities via applications, even when connection pooling mechanisms are used for communication between the Web application and associated database. The lack of visibility in these situations has serious disclosure and audit accountability issues. This capability is difficult to find in most products in the market, since they either address application or database security, but not both. However, Imperva's SecureSphere products address this important need.

## **Preventing SQL Injection with SecureSphere Application Data Security Solutions**

The Imperva SecureSphere Web application firewall features advanced protection against SQL injection attacks and incorporates a multi-layer security model that enables precise attack protection from SQL injection without the need for manual tuning. SecureSphere's security architecture incorporates both dynamic positive (white list) and dynamic negative (black list) security models. Robust enforcement algorithms draw on both security models to identify and block even the most sophisticated attacks. In addition, customers can easily upgrade from the SecureSphere Web application firewall to the SecureSphere Database Security Gateway, which adds advanced protection against direct database SQL injection and other attacks targeted at the database.



## Multi-Layered Approach for Protection against SQL Injection

SecureSphere utilizes a multi-tiered approach for detecting of SQL injection, HTTP protocol validation, IPS based signatures, Dynamic Profiling, and Correlated Attack Validation.

Imperva has developed specific protection against SQL injection through a dedicated SQL Injection alert that is part of the SecureSphere security engine. By combining behavioral indications (character type / length violation, special characters, etc.) with anti-evasion mechanisms, simple pattern matching and fully fledged regular expressions, SecureSphere is able to accurately defend application data against a wide range of SQL injection threats.



- » **HTTP Protocol Validation** prevents protocol exploits including buffer overflow, malicious encoding, HTTP smuggling, and illegal server operations. Flexible policies enable strict adherence to RFC standards while allowing minor variations for specific applications.

SecureSphere integrates Dynamic Profiling, IPS, and Correlated Attack Validation technologies to identify SQL injection with unmatched accuracy.

- » **Dynamic Profiling** delivers query-level access control by automatically creating profiles of each user and application's normal query patterns. Any query (such as a SQL injection attack query) that does not match previously established user or application patterns are immediately identified.

SecureSphere's Dynamic Profiling technology examines live traffic to automatically create a comprehensive model or "profile" of the site. Specific elements of the profile include dynamic URLs, HTTP methods, cookies, parameter names, parameter lengths, and parameter types. The profile then serves as a positive security model for the Web application. By continuously comparing user interactions to the profile, SecureSphere can detect any unusual Web activity. As the Web site changes over time, advanced learning algorithms automatically update the profiles to eliminate any need for manual tuning.

Figure 1 presents a SecureSphere Dynamic Profile of a "Credit Card Number" parameter corresponding to the appropriate length of the Credit Card number that is entered into an online form of an ecommerce site:

URL Parameters							
Name	 	Value Type	Min	Max	Required	Read Only	Prefix
CCNumber		Numeric	16	16	<input checked="" type="checkbox"/>	<input type="checkbox"/>	

**Figure 1: SecureSphere automatically builds a model of each Web application parameter**

The SecureSphere positive-security model identifies the parameter as a required parameter consisting of Numeric characters with a minimum and maximum length of sixteen characters. The insertion of more or less than sixteen characters into the parameter conflicts with the profile and a SecureSphere Parameter Length Violation Alert is triggered.

There are a number of SQL injection vulnerabilities that are well known, such as SQL query string patterns related to the basic "OR 1=1" evasion technique described earlier. These types of known threats can be mitigated through signature pattern matching. Recognizing this, Imperva has incorporated IPS as part of its advanced security engine to mitigate common vulnerabilities.

- » **SecureSphere IPS** includes unique database signature dictionaries designed specifically to identify vulnerable stored procedures and SQL injection strings, preventing database platform attacks. SecureSphere also has a database of SQL injection based signatures used for recognition of known SQL injection patterns found in user input to Web applications quick and easy protection from known SQL injection attack, as well as protection from known network, operating system, and Web server software attacks. SecureSphere goes beyond a standalone IPS product because it learns and understands application behavior (Dynamic Profiling) to accurately prevent malicious users from abusing legitimate applications and combines this with Correlated Attack Validation, resulting in a lower rate of false positives.
- » **Correlated Attack Validation** correlates security violations originating from multiple SecureSphere detection layers. By correlating multiple violations from the same user, SecureSphere is able to detect SQL injection with a degree of accuracy that is not possible using any single detection layer.

## End-to-end Coverage for Applications and Databases Enables Stronger Protection

Because SecureSphere provides end-to-end coverage for protection against both database and Web application threats, it enables complete protection against SQL injection attacks irrespective of the origin of the attack.

SecureSphere goes beyond simple signature based mechanisms and IPS solutions. It observes the SQL communication and builds a profile consisting of all allowed SQL queries. Whenever a SQL injection attack occurs, SecureSphere can detect the unauthorized query sent to the database. SecureSphere can also correlate anomalies on the SQL stream with anomalies on the HTTP stream to accurately detect SQL injection attacks. Therefore, SecureSphere offers a unique value in protecting actual applications and databases themselves, as opposed to just the supporting platforms and infrastructure software.

SecureSphere provides effective protection against direct database SQL injection. SecureSphere creates an accurate baseline database usage profile, tracking acceptable usage of the application, and maintains the profile on an ongoing basis on the database server. In real time, SecureSphere detects queries that deviate from the learned profile and correlates them with information about the sensitive tables in the database. If it detects an unauthorized attempt to access sensitive data, SecureSphere will issue an alert and can block the user request.

## Conclusion

SQL injection 2.0 threats impose a real danger to organizations that depend on the Web for information exchange with and for providing valuable services to their partners and customers. SQL injection attacks are constantly evolving, changing their shape and target. Relatively simple forms of SQL injection attacks are becoming more widespread through usage of automation tools and popular search engines, while advanced SQL injection attacks are being crafted to more seamlessly bypass built-in security mechanisms within applications and databases.

To protect critical business data from being compromised by SQL Injection, organizations should mitigate the known vulnerabilities by deploying a Web application firewall in front of their applications, and they should follow a regular code fix procedure for fixing the critical areas of the applications that are vulnerable due to some logical business flaw.

## About the Imperva Application Defense Center

The security and compliance experts at the Imperva Application Defense Center (ADC) conduct research on issues related to application and data security and compliance. The ADC is lead by Amichai Shulman, Imperva's CTO who was named to InfoWorld Magazine's "Top 25 CTOs of 2006" list. Automated feeds from the ADC ensure that SecureSphere is always armed with the latest information about new security developments and regulatory compliance best practices. This specific expertise in application and database security is unmatched in the industry today.



**Imperva**

North America Headquarters  
3400 Bridge Parkway  
Suite 101  
Redwood Shores, CA 94065  
Tel: +1-650-345-9000  
Fax: +1-650-345-9004

International Headquarters  
125 Menachem Begin Street  
Tel-Aviv 67010  
Israel  
Tel: +972-3-6840100  
Fax: +972-3-6840200

Toll Free (U.S. only): +1-866-926-4678  
[www.imperva.com](http://www.imperva.com)

© Copyright 2008, Imperva  
All rights reserved. Imperva and SecureSphere are registered trademarks of Imperva.  
All other brand or product names are trademarks or registered trademarks of their respective holders.  
#WP-SQL\_INJECTION\_2.0-1208rev1