

# Database Security

## *The Past, the Present, the Future*

Mark Kraynak

Director of Product Marketing

Imperva

[mark@imperva.com](mailto:mark@imperva.com)



# Who we are

- Venture-backed, privately-owned company with operations and HQ in the US and Israel
- Leadership of **Shlomo Kramer**
  - Check Point co-founder
  - Co-developer of Stateful Inspection
  - Imperva Co-founder and CEO
- Deep expertise in security
  - Application, Database and Data Center Security Elite Specialists
    - Israeli Defense Force cyber warfare team
    - Private sector application, network, database and data center penetration testing and security consultants

# Agenda

- Intro Demo: SQL Injection
- Noteworthy Data Thefts
- A Multi-Dimensional Problem
- SQL Injection Revisited
- Countermeasures Demonstration
- Effective countermeasures
- Q&A

## Data Theft

# Publicity & Governmental Action

### ➤ Legislation fuels publicity

- 15 states with security breach laws with 4-10 more expected in 2005

**CNN Money**

#### 40M credit cards hacked

Breach at third party payment processor affects 22 million Visa cards and 14 million MasterCard.

**THE WALL STREET JOURNAL.**

#### DSW Shoe Says Theft of Data Involved 1.4 Million Credit Cards

#### Online thieves get personal information on 310,000 in US

NEW YORK (AFP) - Some 310,000 people could have lost their personal identification data to online thieves who broke into the computers of



#### 145,000 AMERICANS' IDENTITY DATA STOLEN

Source: MARK SCHWANHAUSSER, Mercury News

A company that sells personal data on consumers said Wednesday that it's alerting Californians -- that they might be vulnerable to identity theft after a crime ring paid for the data. ChoicePoint, a Georgia company that boasts it has said Tuesday that it had alerted 35,000 Californians that they were vulnerable, as

#### CSU Breach Exposes 59,000 to Hackers



Erika Morphy, [www.enterprise-security-today.com](http://www.enterprise-security-today.com)

Identity theft and computer crimes continue to soar. The latest example is the California fact

**THE WALL STREET JOURNAL.**

#### LexisNexis Reveals Further Breaches of Database

LexisNexis said 310,000 Americans, nearly 10 times its original estimate, have had their personal data accessed by unauthorized individuals via its computer systems, raising fresh concerns about the data-collection in identity

#### MSN flaw put Hotmail accounts at risk

By Joris Evers, CNET News.com  
Published on ZDNet News: June 6, 2005, 5:53 PM



**washingtonpost.com**

#### FDIC Alerts Employees of Data Breach

In letters dated last Friday, the agency told roughly 6,000 people to be "vigilant over the next 12 to 24 months" in monitoring their financial

### ➤ Publicity fuels more legislation

- 20 more states considering additional security breach legislations
- 10 US Senate bills introduced in 2005 (Identify Theft Protection Act)

# Costs are Real for Businesses

## FTC Consent Agreements

- Penalty - 20 yrs of bi-annual audits by outside security consultants
- **Microsoft, Petco**, and **Guess** for “deceptive claims” about security
- **BJ Wholesale** for "unfair" business practices of lax computer security and major credit card breach in 2004

## Hard Dollar Estimates

- BJ Wholesaler - **\$16M** reserve
- DSW - **\$6.5M to \$9.5M** set aside
- Polo Ralph Lauren - **\$1.6M** claimed
- Chipotle's Mexican Grill - **\$.75M** claim

"There is going to be a flood of lawsuits,"  
- former Justice Department prosecutor

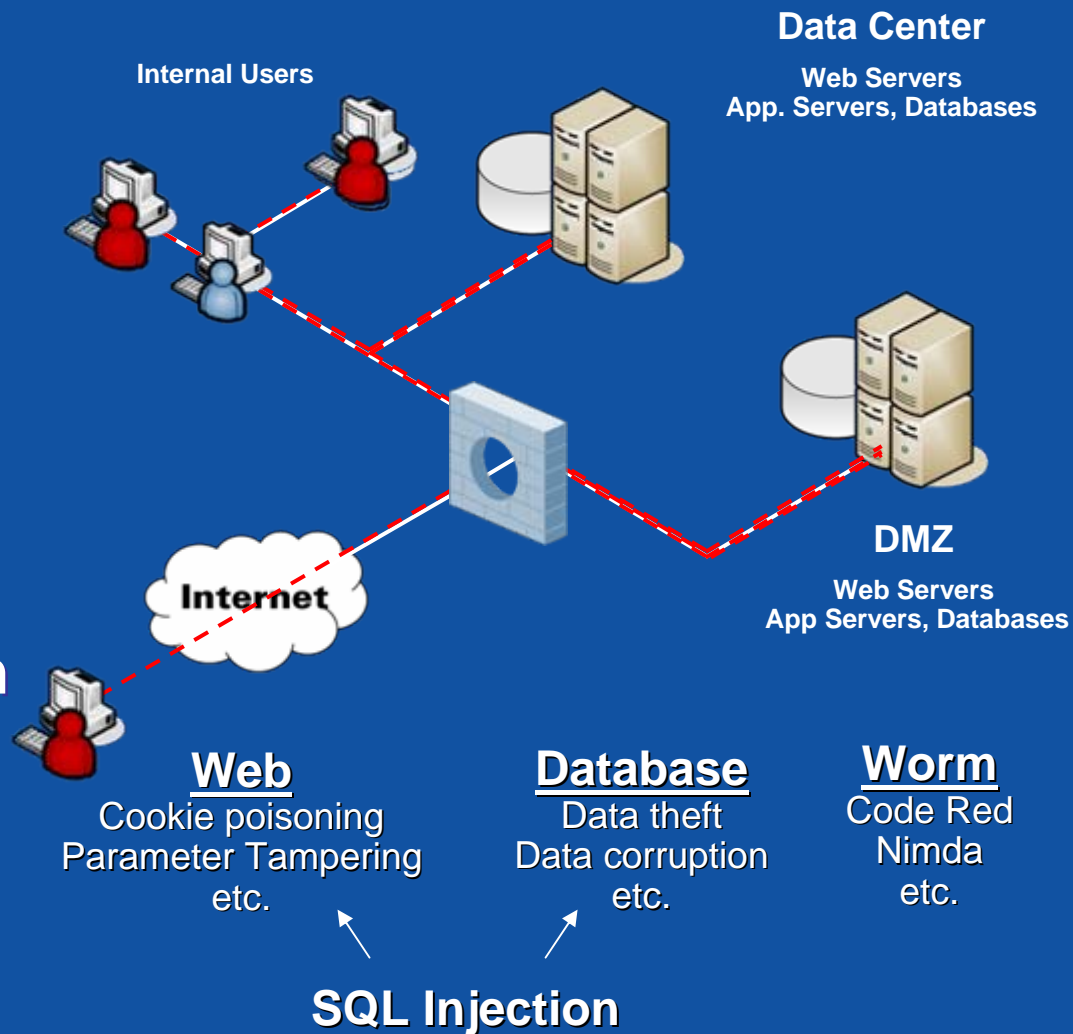
## Data Breach Lawsuits

- Class action - CA law on "reasonable" security for customer information
  - **CardSystems Solutions (\$120M)**
  - **LexisNexis**
  - **ChoicePoint**
- Ohio Attorney General - "Implied warranty" to protect consumers info
  - **DSW Shoe Warehouse**

# Database Threats and Vulnerabilities

A multi-dimensional problem

- Direct: Database breach
  - Internal sources
  - Very high value target
- Indirect: Web attacks
  - Targeted
  - External sources
  - “Custom” vulnerabilities
- Platform: Worm infection
  - External and internal sources
  - Generic attack



# SQL Injection: A Pervasive Attack to Compromise Data

# SQL Injection: What is it

- An attack methodology
  - Allows the attacker to alter SQL statements generated by an application (due to the lack of input validation)
  - SQL Injection opens up the full semantics of database access languages (so the attacker has a LOT of tools available)
- An application is vulnerable to SQL Injection as a result of the *programming* of the application itself
- Built-in database security and traditional network security solutions are hard-pressed to correct this issue
  - (we will demonstrate some of the reasons why...)



# SQL Injection: Example 1

## Authentication Circumvention

The Code:

```
...  
Sql Qry = "SELECT * FROM Users WHERE Username = ' " &  
Request.QueryString("User") & "' AND Password = ' " &  
Request.QueryString("Pass") & "' "  
Logi nRS.Open Sql Qry, MyConn  
If Logi nRS.EOF Then Response.Wri te("Inval id Logi n")  
...
```

When a normal user logs in, the following query is created:

```
SELECT * FROM Users WHERE Username = ' John'  
AND Password = ' Smi th'
```

The attacker, however, inserts `X' OR '1'='1` as the password, altering the query into the following (non empty) one:

```
SELECT * FROM Users WHERE Username = ' John'  
AND Password = ' X' OR '1'='1'
```

# SQL Injection: Example 2

## Data Retrieval

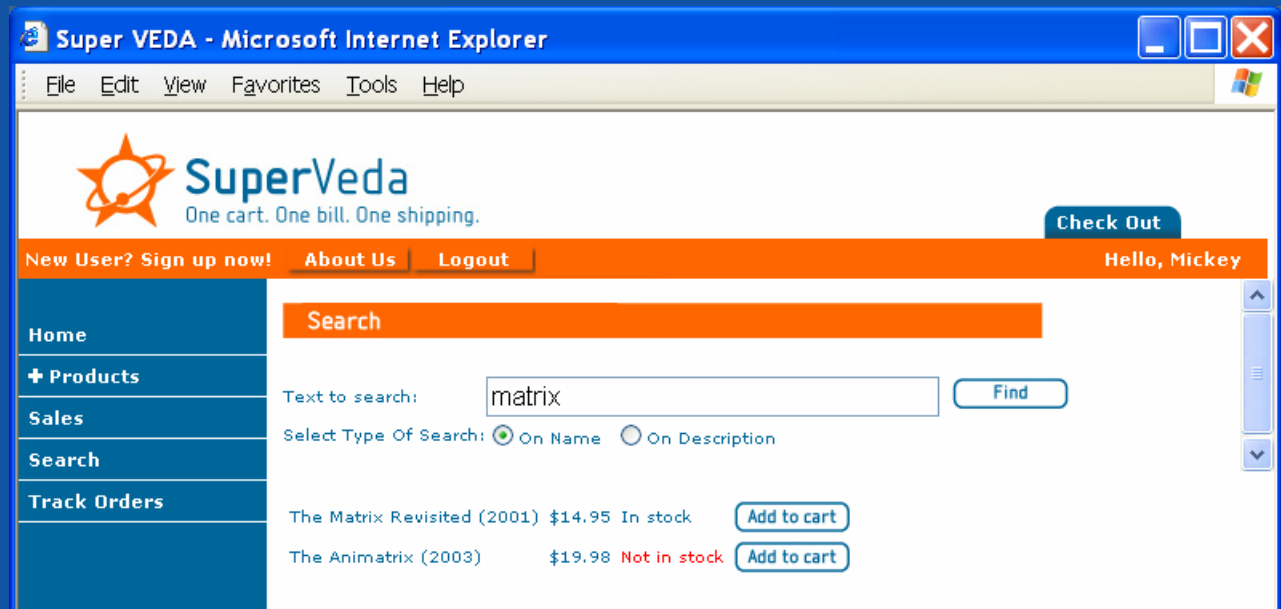
The Code:

```
Sql Qry = "SELECT * FROM Products WHERE ProdDesc LIKE " & "'%' " & Request.QueryString("SearchTerm") & "%' "
ProdsRS.Open Sql Qry, MyConn
```

The query that is normally created when using the form is:

```
SELECT * FROM Products WHERE ProdDesc LIKE '%matrix%
```

Showing all matching results:



The screenshot shows a web browser window titled "Super VEDA - Microsoft Internet Explorer". The page displays the SuperVeda logo and navigation links. A search bar contains the text "matrix". Below the search bar, two search results are shown:

| Product Name                | Price   | Stock Status | Action                      |
|-----------------------------|---------|--------------|-----------------------------|
| The Matrix Revisited (2001) | \$14.95 | In stock     | <a href="#">Add to cart</a> |
| The Animatrix (2003)        | \$19.98 | Not in stock | <a href="#">Add to cart</a> |

# SQL Injection: Example 2

## Data Retrieval (Continued)

The attacker now uses the following string as the search term:

```
99' UNION SELECT null , null , username || ';' || password  
|| ';' || ccnumber || ';' || ccdatetime, null , null , 0, null  
FROM Users--
```

Causing the original query to be altered into the following one:

```
SELECT * FROM Products WHERE ProdName LIKE '%99' UNION  
SELECT null , null , username || ';' || password ||  
';' || ccnumber || ';' || ccdatetime, null , null , 0, null  
FROM Users --%'
```

As a result, the query now returns all products whose name terminates with '99' (probably none), as well as the list of the users, their passwords, and their credit card numbers

# Countermeasures: Common Approaches That Don't Work

## Candidate 1: Error message hiding

Simplest and most common countermeasure against SQL Injection

- Achieved by simple configuration options (e.g. suppress error messages or set a custom error message)
- A classic ***Security By Obscurity*** approach

... why it won't work ...

# Blindfolded SQL Injection

# Blindfolded SQL Injection

- What is it?
  - A set of techniques for the detection and exploitation of SQL Injection vulnerabilities
  - Eliminates the reliance on error messages
- The attacker employs Boolean tests determine whether an error has occurred

## Blindfolded SQL Injection: Identifying an Opportunity

- Testing for the existence of SQL Injection can be done simply by replacing a field with equivalent SQL syntax:
  - The number **5** can be represented in SQL as **(6-1)**
  - The string **'test'** can be represented as **'te'+ 'st'** (in MS SQL) or **'te'// 'st'** (in Oracle)
  - A date can be replaced with the database's date function
    - getdate() (MS SQL) or sysdate (Oracle)
- Matching results indicate that the system is vulnerable, while an error indicates that the syntax was not parsed by an SQL parser



## *Blindfolded SQL Injection:*

# Targeting the Attack Parameters

- Since errors are hidden / identical some form of differentiation is required
- Step #1 – Enumerating the number of columns
  - Done using an ORDER BY statement, which sorts by specific field
  - When an existing field is chosen, the result is sorted according to it. However, when a non-existent field is chosen, an error occurs
- Step #2 – Enumerating the type of fields
  - Create an initial request with all fields set to NULL
  - Type detection is done by guessing one field at a time
- Once field types are known, exploit is trivial

## *Blindfolded SQL Injection:* **Identifying a Column**

- Union Select null,null,null,null,null,null
  - Error = Syntax isn't right. We have a type issue.
- It takes some time, but we find the right combo:
- Union Select null,null,null,1,null,null...
  - No Error = Syntax & basic typing is right.
- Union Select 1,null,null,1,null,null...
  - No Error = 1st column is integer.
- Union Select 1,2,null,1,null,null
  - ERROR! = 2nd column is NOT integer.
- Union Select 1,'2',null,1,null,null
  - No Error = 2nd column is String.
- Continue until you understand the column types

# Candidate 2: Signature Protection

- Relies on the existing IDS/IPS infrastructure or on an easily installed signature protection component
- Attempts to detect common SQL Injection strings such as: **UNION SELECT**, **OR 1=1**, etc.

## BUT

- Signatures can only be practically applied to HTTP traffic
  - SQL Injection strings are not different than valid SQL statements.
- Placing strict signatures on keywords such as **INSERT**, **SELECT** and **DELETE**, and characters such as **'**, **=** and **--** will cause the security mechanism to block valid requests

... why it won't work ...

# SQL Injection Signature Evasion

# SQL Injection Signature Evasion

- A set of techniques which allow an attacker to evade signature protection mechanisms
- Methods include
  - Detecting signature protection (EASY)
  - Generic evasion techniques
  - SQL language specific evasion techniques

# *SQL Injection Signature Evasion:* **Generic Evasion Techniques**

- Non-SQL Specific
- Employs common IDS evasion techniques, such as:
  - IP Fragmentation
  - TCP Segmentation
  - White Space Diversification
  - Various Encodings (HTTP/UTF8/Unicode/etc)
- Vulnerability to these techniques is a result of poor implementation rather than an inherent problem

# SQL Injection Signature Evasion

## SQL-Based Techniques

- Technique #1 – Value equivalence (instead of OR 1=1)
  - OR 'Simple' = 'Simple'
  - Make the expression look different but still be the same.
    - Adding N will make the value an nvarchar:
    - OR 'Simple' = N'Simple'
  - Concatenation at the SQL level:
    - OR 'Simple' = 'Sim'+ 'ple' (MS-SQL)
    - OR 'Simple' = 'Sim' || 'ple' (Oracle)
- What if the signature detection is looking at a much wider expression like **OR followed by =** ?
  - OR 'Simple' LIKE 'Sim%'
  - OR 'Simple' > 'S'
- SQL is a rich toolset: there are unlimited numbers of examples:
  - OR 'Simple' IN ('Simple')
  - OR 'S' BETWEEN 'R' AND 'T'

# SQL Based Techniques

- Technique #2 – White Space Equivalence / Comments
  - Used to evade signatures that contain white spaces, such as
    - OR 1=1
    - UNI ON SELECT
    - EXEC SP\_
- Using Comments
  - `http://localhost/showproducts.asp?CatID=99'UNI/**/ON /**/SE/**/LECT`



# SQL Injection Signature Evasion

## SQL Based Techniques

- Technique #3 – String Equivalence
  - Basic string equivalence is done by executing a concatenated string (Most DBs have more than one way of doing so), such as:
    - ; EXEC('INS'+ERT INTO...')
    - ; EXECUTE('INS'||ERT INTO...')
  - A possible string equivalence is through its hexadecimal representation, allowing the keyword SELECT to be represented as 0x73656c656374

## Countermeasures

### Candidate 3: DB Access Control Lists (ACLs)

- Least privileges applied to the application account
- Protects the database against system level attacks that require special system privileges, such as the following:

(Oracle examples)

```
; DROP USER <name>  
; DROP TABLE <name>  
; GRANT CONNECT, RESOURCES  
; SHUTDOWN ABORT
```

(MS-SQL examples)

```
; EXEC MASTER.XP_CMDSHELL('cmd.exe /e dir') --  
; SHUTDOWN --  
; DROP DATABASE MyApp --
```

... why it won't (completely) work ...

# SQL Injection Denial of Service

# SQL Injection Denial of Service

- A set of techniques to launch Denial of Service attacks against databases
  - Direct or through SQL Injection
- Basic SQL DoS techniques require the application to be running a privileged user account
- Advanced techniques allow the attacker to perform various destructive activities through a user account with limited privileges
  - Making the server unavailable
  - Corrupting data

## SQL Denial of Service

# Data Corruption/Destruction

- While not a classic DoS attack, Data destruction/corruption may often render the application useless
- Recovery time may be significant
  - Instead of a reboot, data restoration is required
- Attacker looks for pages which perform DELETE or UPDATE statements based on a parameter provided by the user
- Injecting an *OR 1=1* (or equivalent) string will cause the query to delete or alter the entire contents of the table.
  - For instance, injecting into a password change form:

```
UPDATE Users SET Password='BOGUS' WHERE Username='User'  
OR '1'='1'
```

# SQL Denial of Service

## Resource Consumption

- Resource consumption attacks can be achieved by a read-only user
- Classic DoS: Attacker can prevent others from using the server
- Can be performed through several techniques, such as:
  - Creating a very large record set created from a correlated query:

```
SELECT A1.* , B1.* FROM A AS A1, B AS B1
WHERE EXISTS (SELECT A2.* , B3.* FROM A AS A2, B AS B3
              WHERE A1.AID = A2.AID)
AND EXISTS (SELECT B2.* , A3.* FROM B AS B2, A AS A3
           WHERE B1.BID = B2.BID)
```

- Executing endless loops:

```
BEGIN DECLARE @A INT;
        WHILE (1=1) BEGIN
            IF (1=2) BEGIN
                SET @A = 1;
            END
        END
END
```

# Effective Countermeasures

## The Right Solution – Data security in 3 layers

- **The Application** – Write secure code
  - Use Prepared Statements/Parametric Queries
  - Use Stored Procedures
  - Validate Input (length, type, character set)
- **The Database** – Apply available features
  - Restrict database user permissions
  - Impose resource quotas/limit profiles
  - Audit database activity and logs
- **External Mechanism**
  - Use solutions that are aware of application context
  - Revalidate some of the security tasks such as input validation and logging
  - Perform tests on incoming requests and outgoing responses based on expected behavior

# Effective Countermeasures: External Mechanism

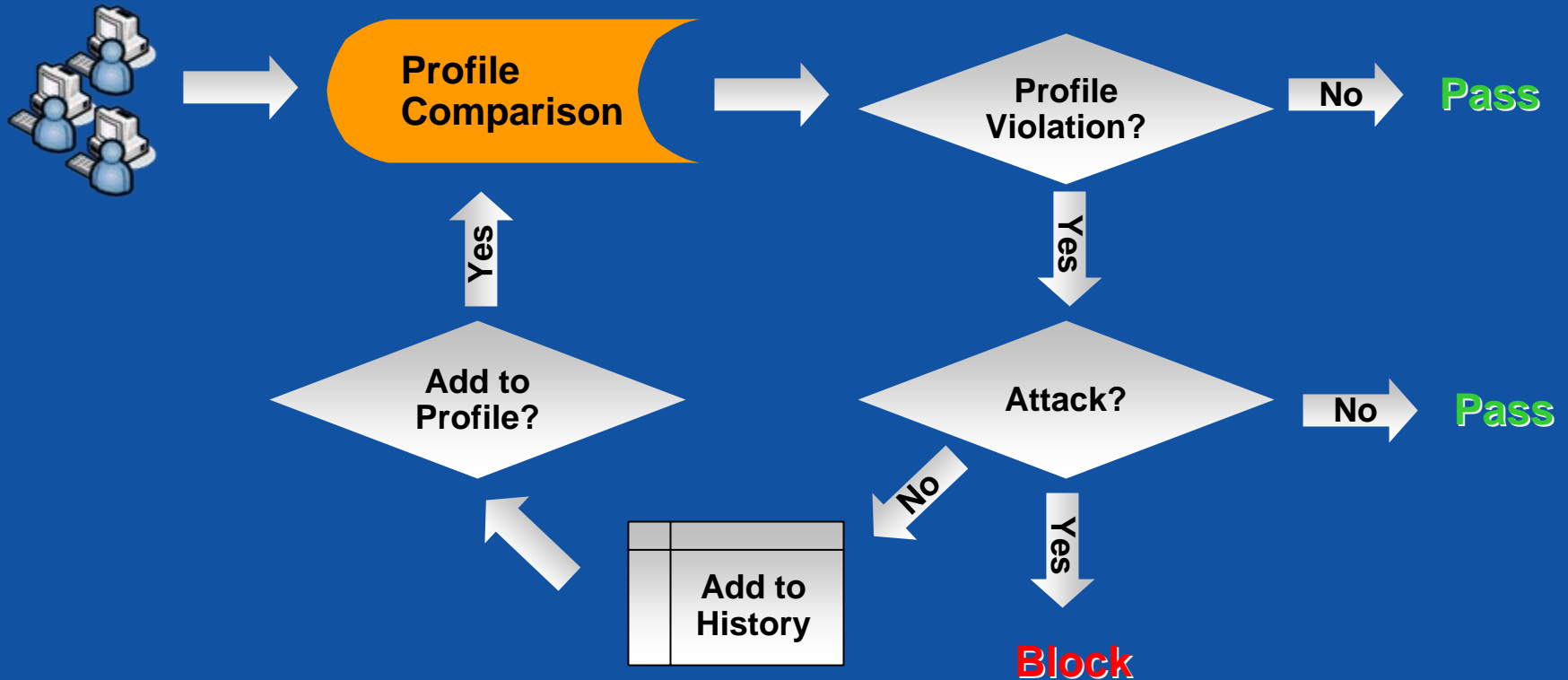
## A Model for Database Security

- *Dynamic Profiling* models appropriate database usage
  - Database objects
    - Queries, stored procedures, privileged operations, system objects, etc
  - Users
    - Auditable trail of user access and activity
  - Business activities and transactions
    - Prevents rogue users from overstepping permissions
  - Time of day and Location
    - Reduces “comfort zone” of rogue users attempting malicious operations outside of normal work locations or work hours
  - Application / Access Method
    - Prevents stolen / abused credentials (i.e. rogue user using an application’s credentials)
  - Requests per second / Data Consumption Rate
    - Prevents DoS attacks and alerts on inappropriate spikes in data use
- Audit and Secure based on usage dynamics
  - Verify real-time usage vs. the baseline
  - Audit deviations from baseline
  - Enforce baseline (as appropriate)



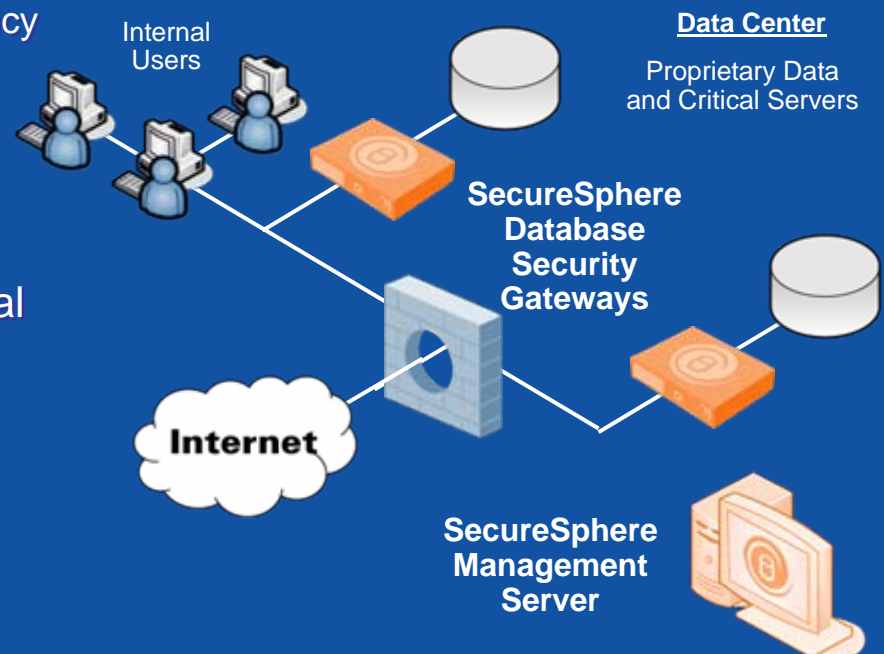
# SQL Profiling

A continuously evolving model of database and application structure, design and deployment



# Imperva SecureSphere Database Security Gateway

- Assessment
  - Models Database Usage
    - Dynamic Profiling learns from traffic
    - Automatically generates security policy
    - Support manual adjustments to policy
  - Identifies Usage Vulnerabilities
- Audit
  - Logs all activity (incl. DBA)
  - Identifies activities that matter in real time
- Protection
  - Alerts (blocks) attacks and policy violations
  - Stops platform attacks
    - Database server software
    - Operating system



# Additional Information

## Live Webinars

Register at [imperva.webex.com](http://imperva.webex.com)

For more information or a copy of the  
“SQL Injection” white paper,  
contact me:

Mark Kraynak  
[mark@imperva.com](mailto:mark@imperva.com)

**THANK YOU!**