



build | integrate | secure

Grow with Denim Group



The Second Most Secure
Database

OWASP San Antonio

February 16th, 2006

Mailing List

Please sign up:
owasp-sanantonio@lists.sourceforge.net

Overview

- Introduction
- The Most Secure Database
- The Second Most Secure Database
- Scenario
- STRIDE for Threat Modeling
- Threat Model and Countermeasures
- Conclusions / Questions

Introduction

- Dan Cornell
- Principal with Denim Group, Ltd.
 - *Texas-based*
 - *Software Development*
 - *Systems Integration*
 - *Web Application Security*
- Java 2 Certified Programmer, MCSD

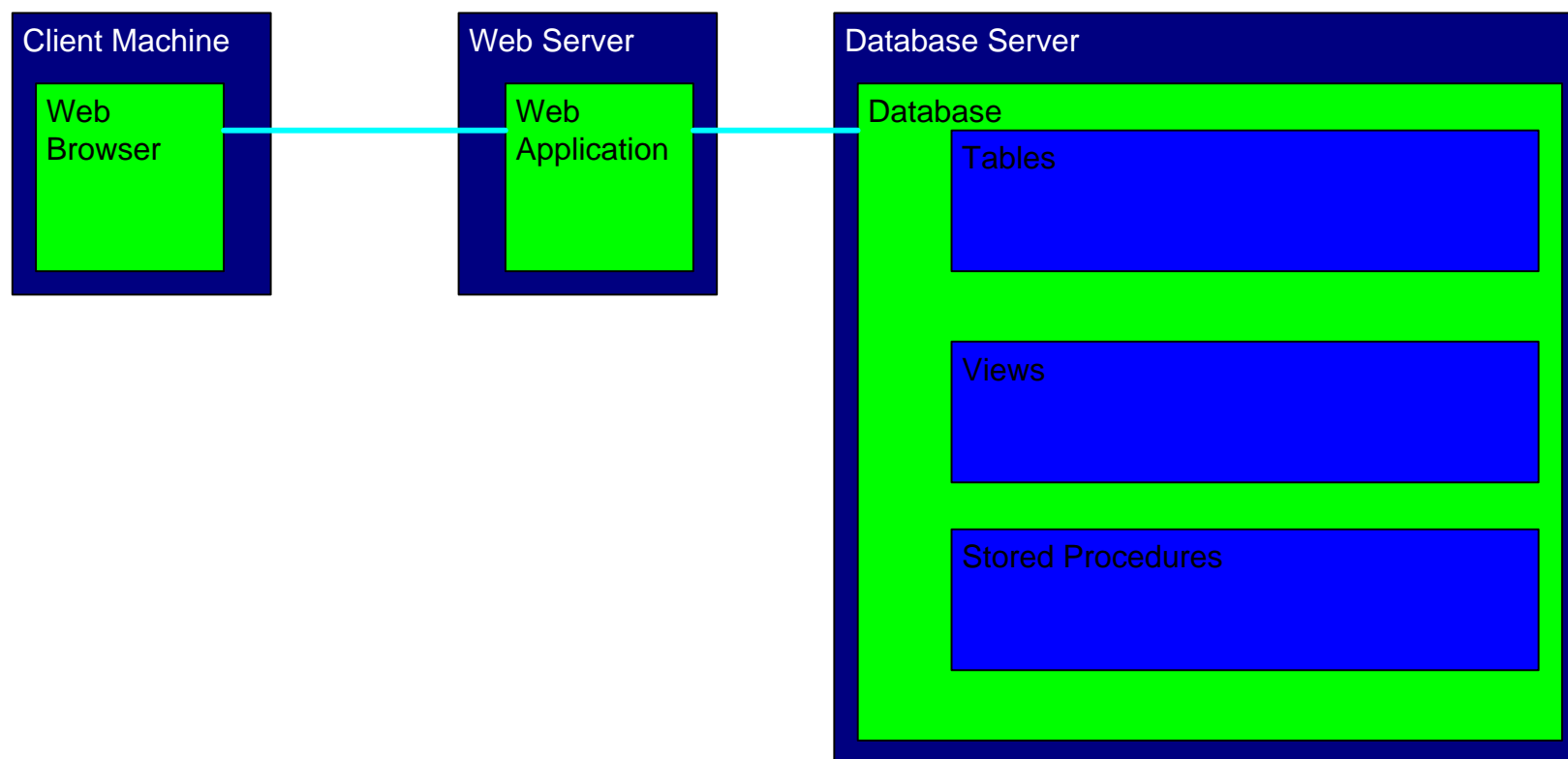
The Most Secure Database



The Second Most Secure Database

- Real world: risk mitigation, not risk elimination
- What are the threats?
- What tools and technologies can be used to address these threats?

Scenario



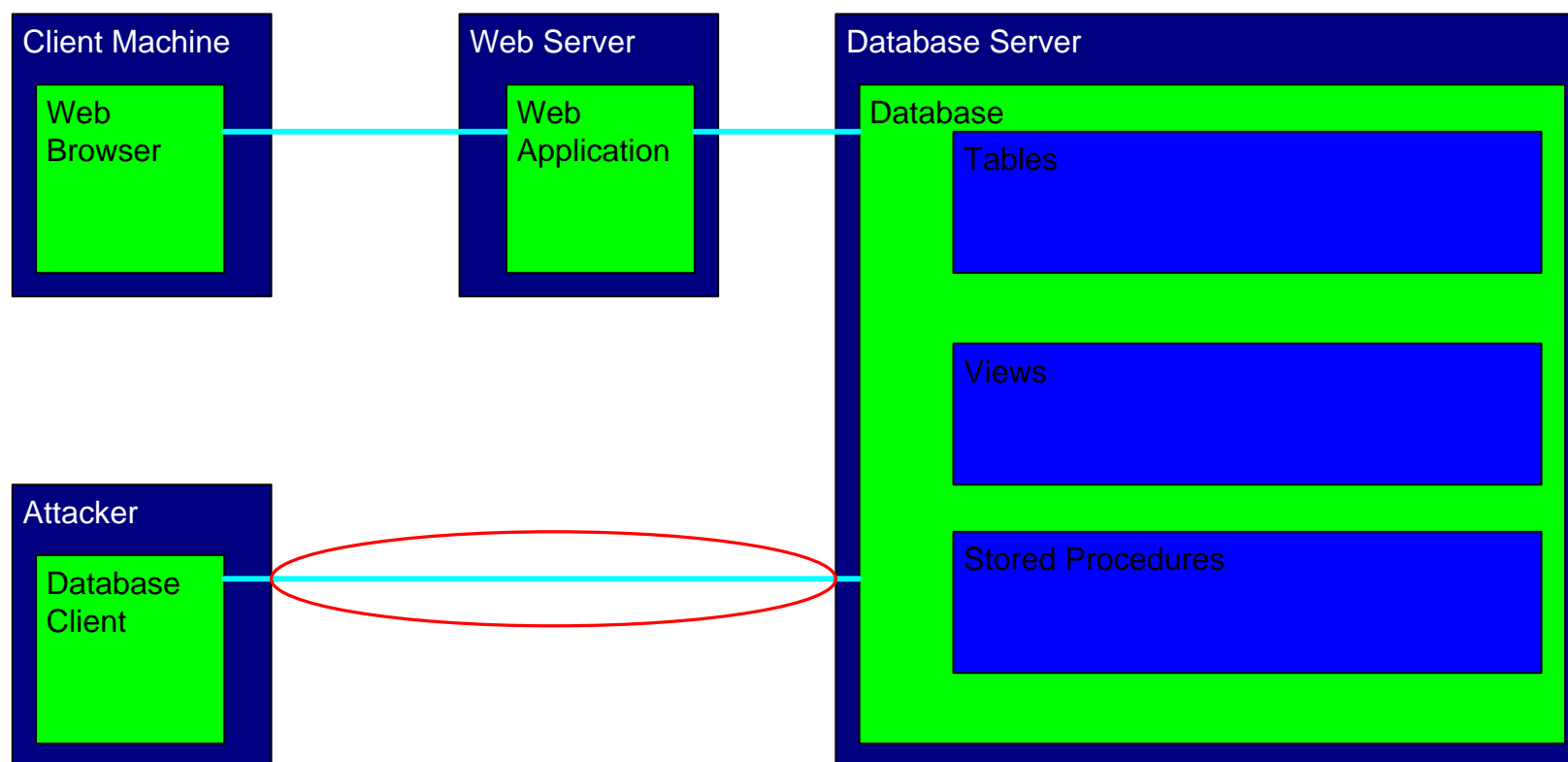
STRIDE for Threat Modeling

- Spoofing Identity
- Tampering
- Repudiation
- Information Disclosure
- Denial of Service
- Elevation of Privilege

Threat Model and Countermeasures

- Malicious Attacker Direct Access to Database
- SQL Injection
- Network Eavesdropping
- Database Compromise via Cracker or Backup Tape Loss
- Unauthorized Writes to Data
- Unauthorized Reads from Data

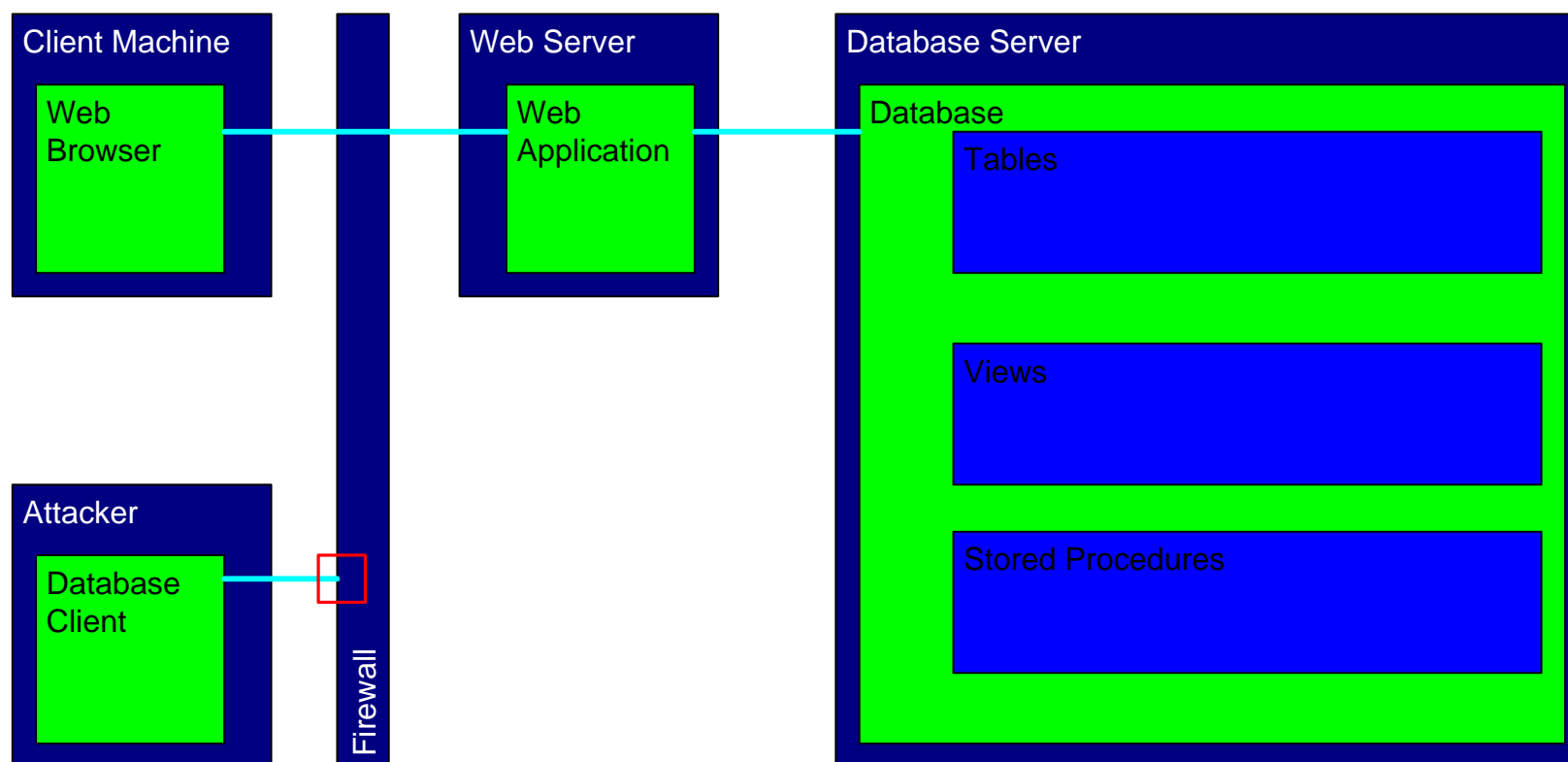
Threat: Malicious Attacker Direct Access to Database



Threat: Malicious Attacker Direct Access to Database

- Spoofing Identity
- Tampering
- Repudiation
- Information Disclosure
- Denial of Service
- Elevation of Privilege

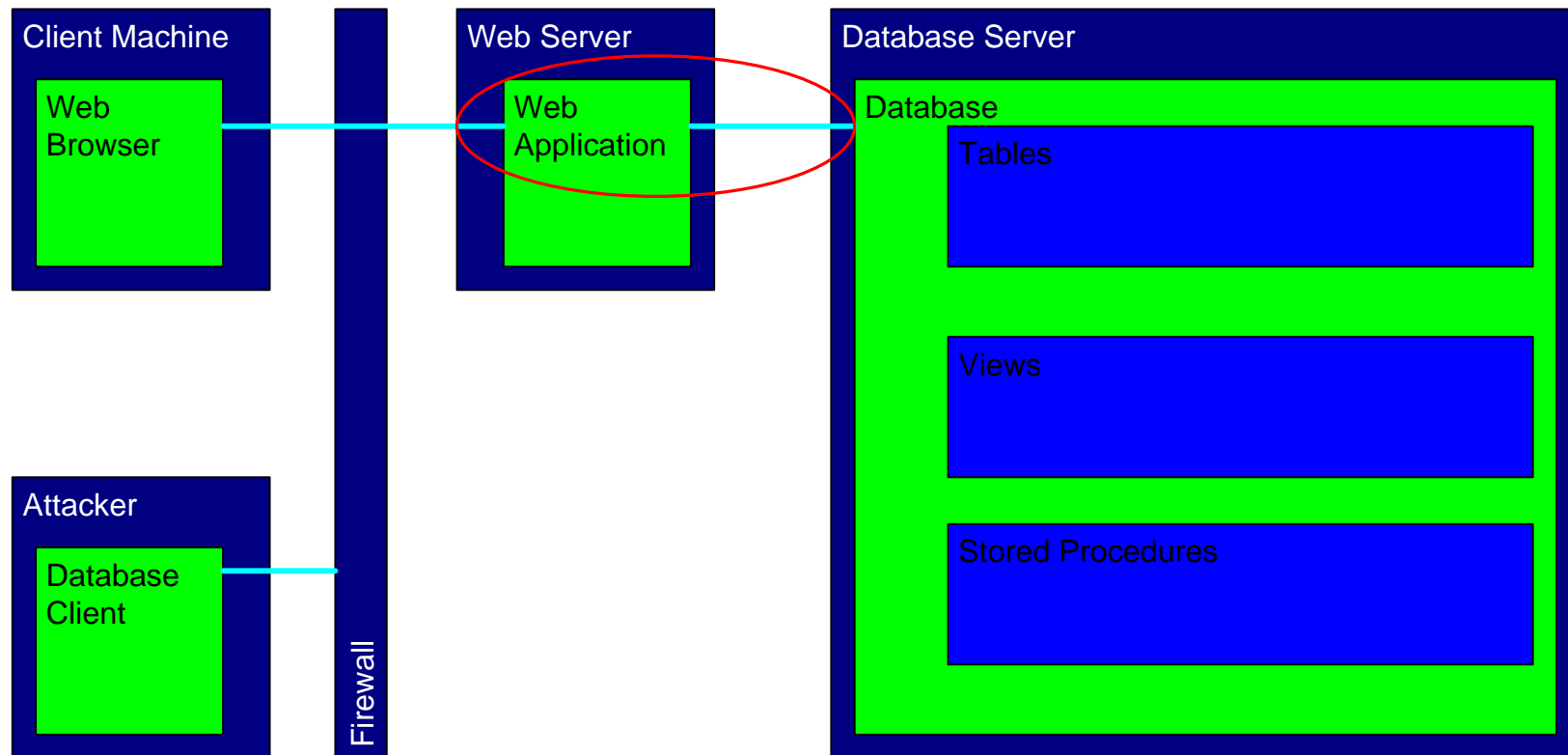
Threat: Malicious Attacker Direct Access to Database



Countermeasures: Malicious Attacker Direct Access to Database

- There are almost no cases where a database server should be Internet-addressable
- Install a firewall
- Change the default (blank?) administrative password
- SQLSlammer worm

Threat: SQL Injection

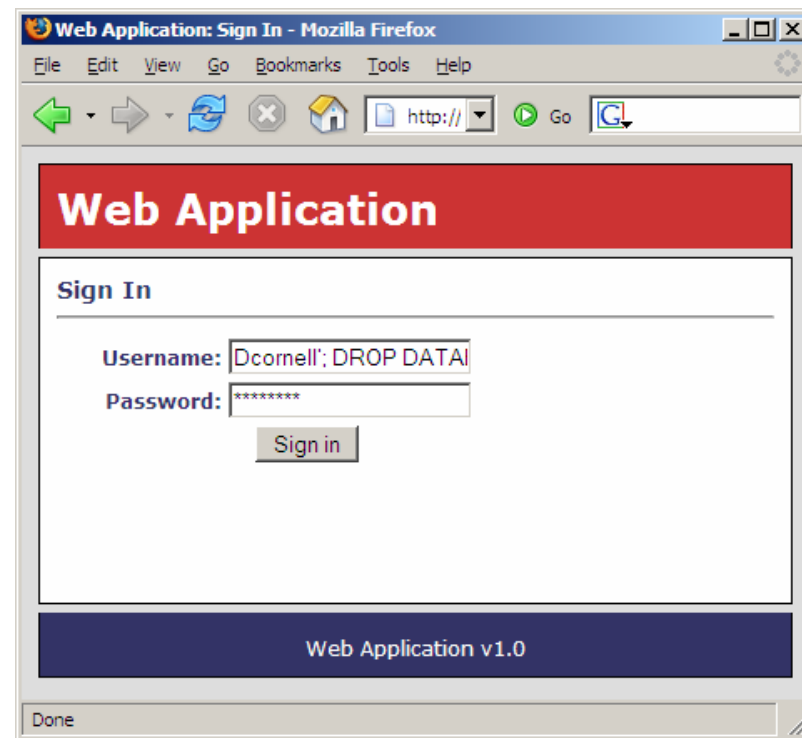


Threat: SQL Injection

```
try {  
    string username =  
    request.getParameter("username");  
    string password =  
    request.getParameter("password");  
    string sSql = "SELECT * FROM User WHERE  
    username = " + username + " AND password = " +  
    password + """;  
    Statement stmt = con.createStatement();  
    ResultSet rs = stmt.executeQuery(sSql);  
    ...  
} catch(Exception ex) {}
```

Threat: SQL Injection

- Specially crafted input contains SQL control characters



Threat: SQL Injection

- Malicious user sends in a username parameter of: Dcornell'; DROP DATABASE Ecommerce; --

SQL Executed is:

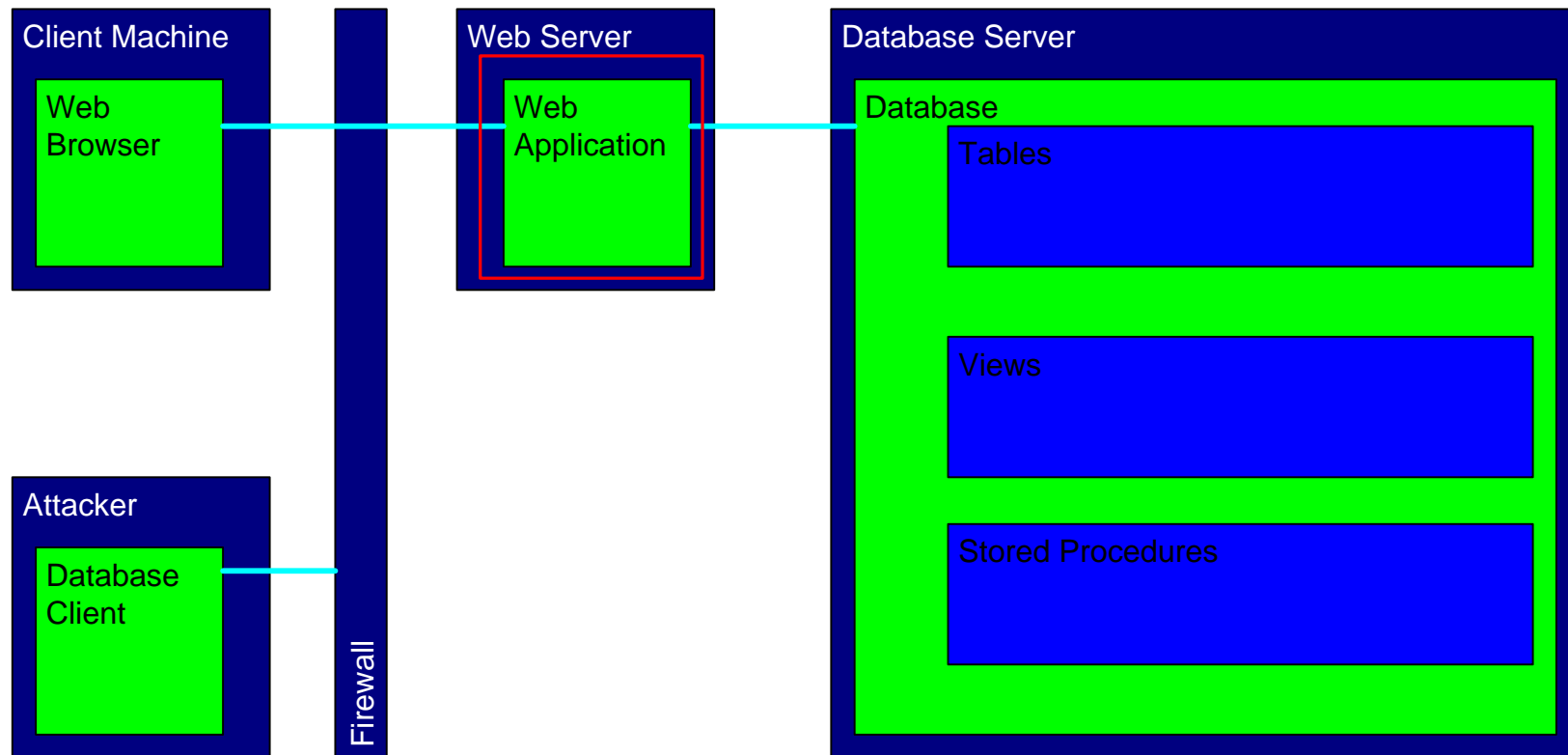
```
SELECT * FROM User WHERE username = 'Dcornell';  
DROP DATABASE Ecommerce; -- AND password =  
'whocares'
```

- Attacker can execute arbitrary database queries with the same permissions as the application
 - *View sensitive data*
 - *Modify data*
 - *Destroy data*

Threat: SQL Injection

- Spoofing Identity
- Tampering
- Repudiation
- Information Disclosure
- Denial of Service
- Elevation of Privilege

Countermeasures: SQL Injection



Countermeasures: SQL Injection

- Do not create SQL statements from a combination of unfiltered user input and static text
- Use prepared statements or stored procedures
 - *Even these are potentially vulnerable if used incorrectly*
 - *Beware of building SQL statements in stored procedures and using EXEC*
- Do not use a highly-privileged account for web application database access
 - *Separate duties between multiple, role-based SQL accounts*

Countermeasures: SQL Injection

.NET

- Use Stored Procedures:

```
SqlConnection PubsConn = new SqlConnection ("Data Source=server;integrated " + "Security=sspi;initial  
    catalog=pubs;");  
SqlCommand testCMD = new SqlCommand ("TestProcedure", PubsConn);  
testCMD.CommandType = CommandType.StoredProcedure;  
SqlParameter RetVal = testCMD.Parameters.Add ("RetVal", SqlDbType.Int);  
RetVal.Direction = ParameterDirection.ReturnValue;  
SqlParameter IdIn = testCMD.Parameters.Add ("@au_idIN", SqlDbType.VarChar, 11);  
IdIn.Direction = ParameterDirection.Input;  
SqlParameter NumTitles = testCMD.Parameters.Add ("@numtitlesout", SqlDbType.VarChar, 11);  
NumTitles.Direction = ParameterDirection.Output ;  
IdIn.Value = "213-46-8915";  
PubsConn.Open();  
SqlDataReader myReader = testCMD.ExecuteReader();  
...  
myReader.close();
```

Countermeasures: SQL Injection

.NET

- Use Parameterized Queries:

```
string ConnectionString = "...";
con = new SqlConnection(ConnectionString);
con.Open();
string CommandText = "SELECT FirstName, LastName" +
    " FROM Employees" +
    " WHERE (LastName LIKE @Find)";

cmd = new SqlCommand(CommandText);
cmd.Connection = con;
cmd.Parameters.Add(
    new SqlParameter("@Find",
                    System.Data.SqlDbType.NVarChar,
                    20,
                    "LastName"));
cmd.Parameters["@Find"].Value = txtFind.Text;
rdr = cmd.ExecuteReader();
...
rdr.Close();
```

Countermeasures: SQL Injection

J2EE

- Use stored procedures:

```
CallableStatement cs = con.prepareCall("{call SHOW_SUPPLIERS}");  
cs.setInt(1, 75);  
cs.setString(2, "Colombian");  
ResultSet rs = cs.executeQuery();
```

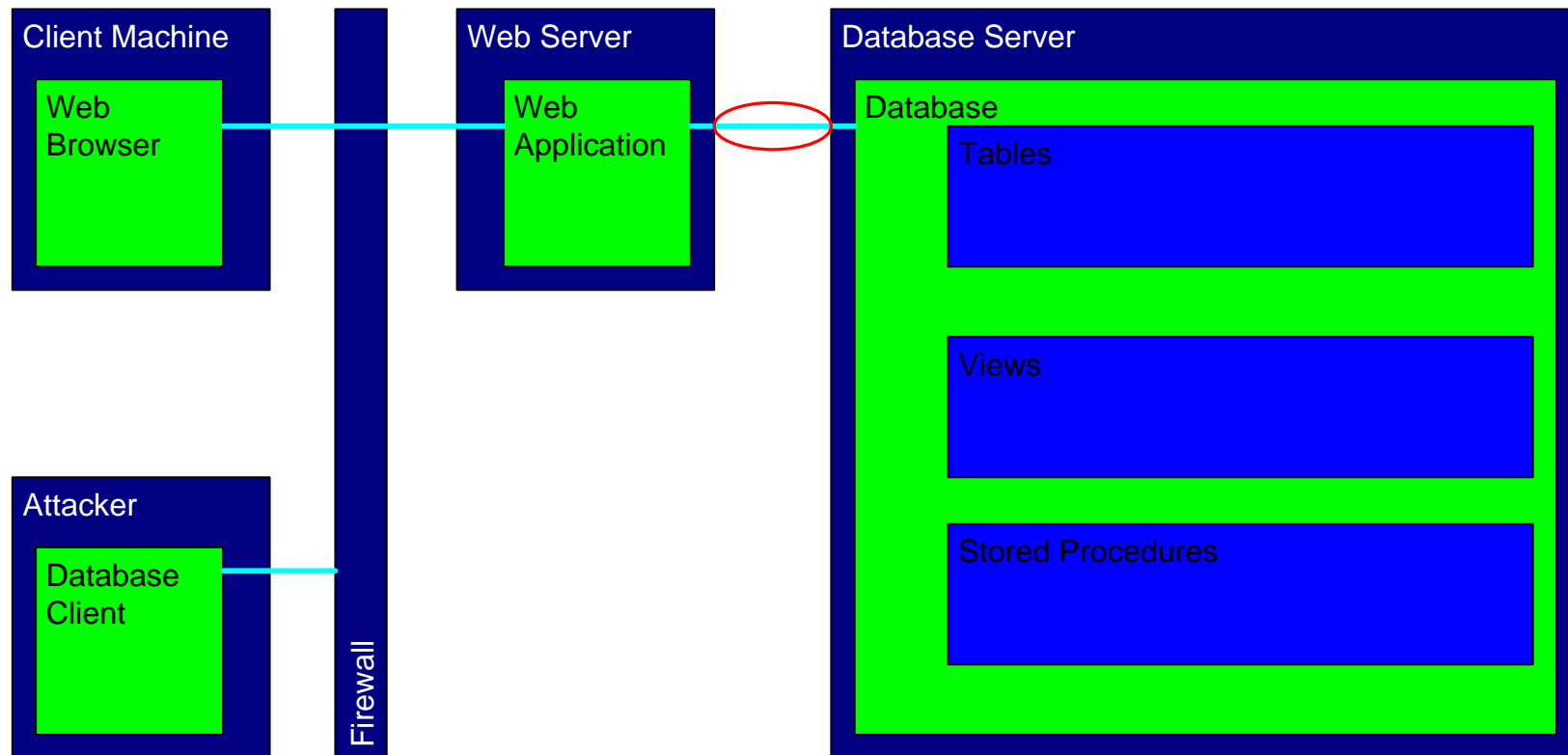
Countermeasures: SQL Injection

J2EE

- Use parameterized queries:

```
PreparedStatement updateSales = con.prepareStatement(  
"UPDATE COFFEES SET SALES = ? WHERE COF_NAME LIKE ?");  
  
updateSales.setInt(1, 75);  
updateSales.setString(2, "Colombian");  
updateSales.executeUpdate();
```

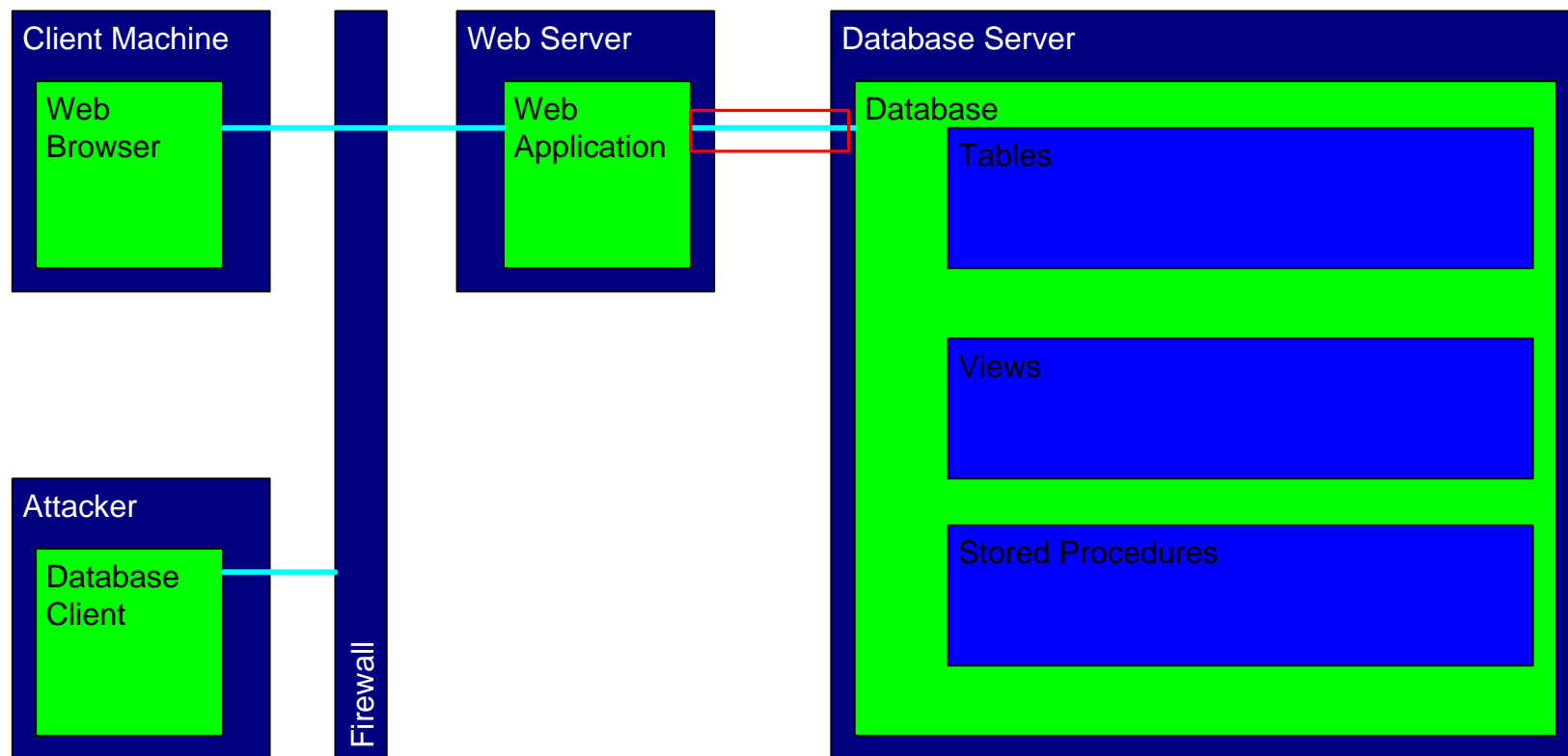
Threat: Network Eavesdropping



Threat: Network Eavesdropping

- Information Disclosure

Countermeasures: Network Eavesdropping



Countermeasures: Network Eavesdropping

- Encrypt traffic between the web application and the database server
- Weight the threat versus performance costs
 - *How much of a threat is there inside of a “protected” network?*
 - *A MUST if database access traffic runs over the Internet*
 - But you wouldn't be doing that anyway, right?

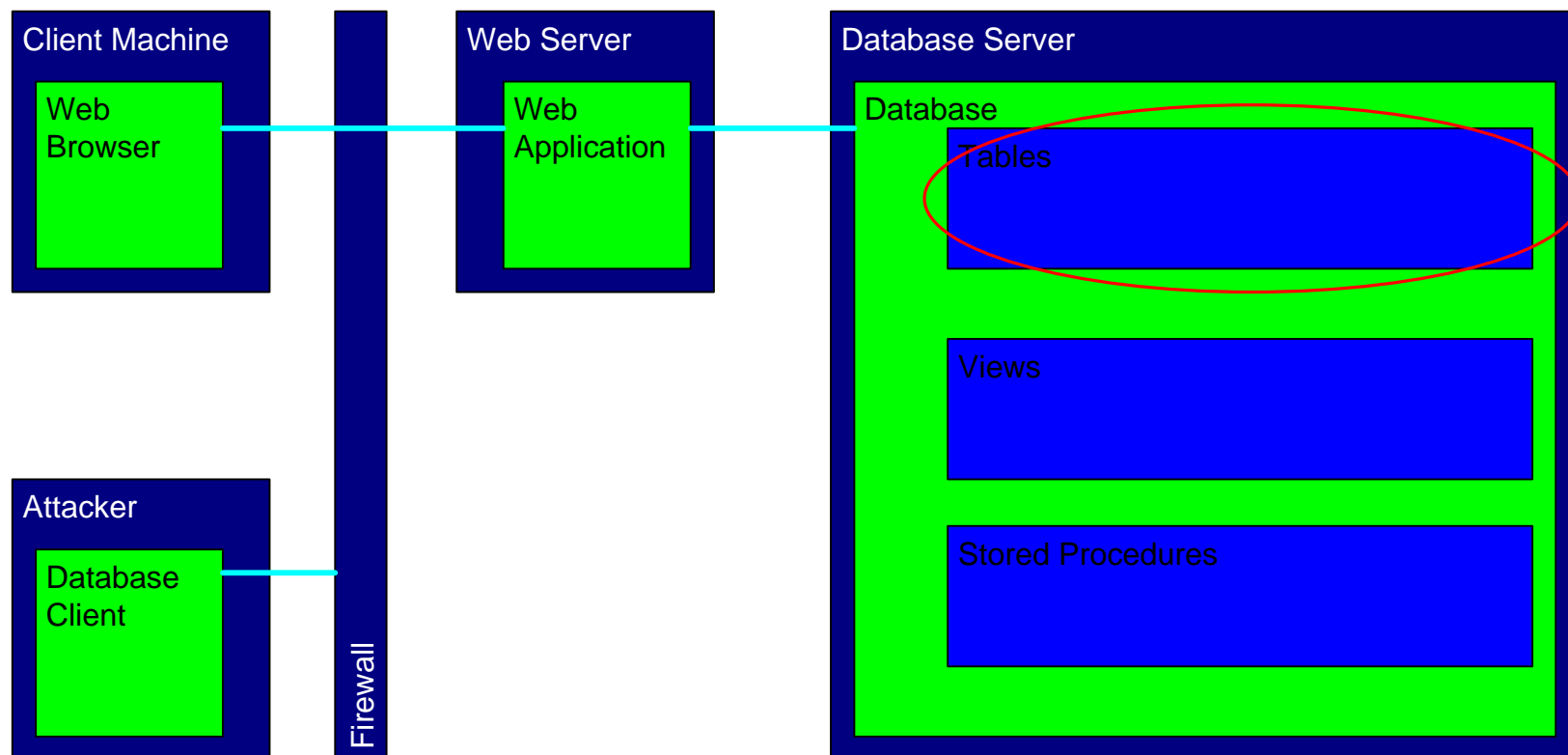
Countermeasures: Network Eavesdropping

- SQL Server 2005:
 - *Relies on having a valid Certificate Server in your Windows network environment*
 - *Force Protocol Encryption option for both servers and clients*

Countermeasures: Network Eavesdropping

- MySQL:
 - <http://dev.mysql.com/doc/refman/5.0/en/secure-connections.html>
 - *Somewhat difficult to enable on the server-side because it is a build-time option*
 - TMTOWTDI: OpenSSL or yaSSL

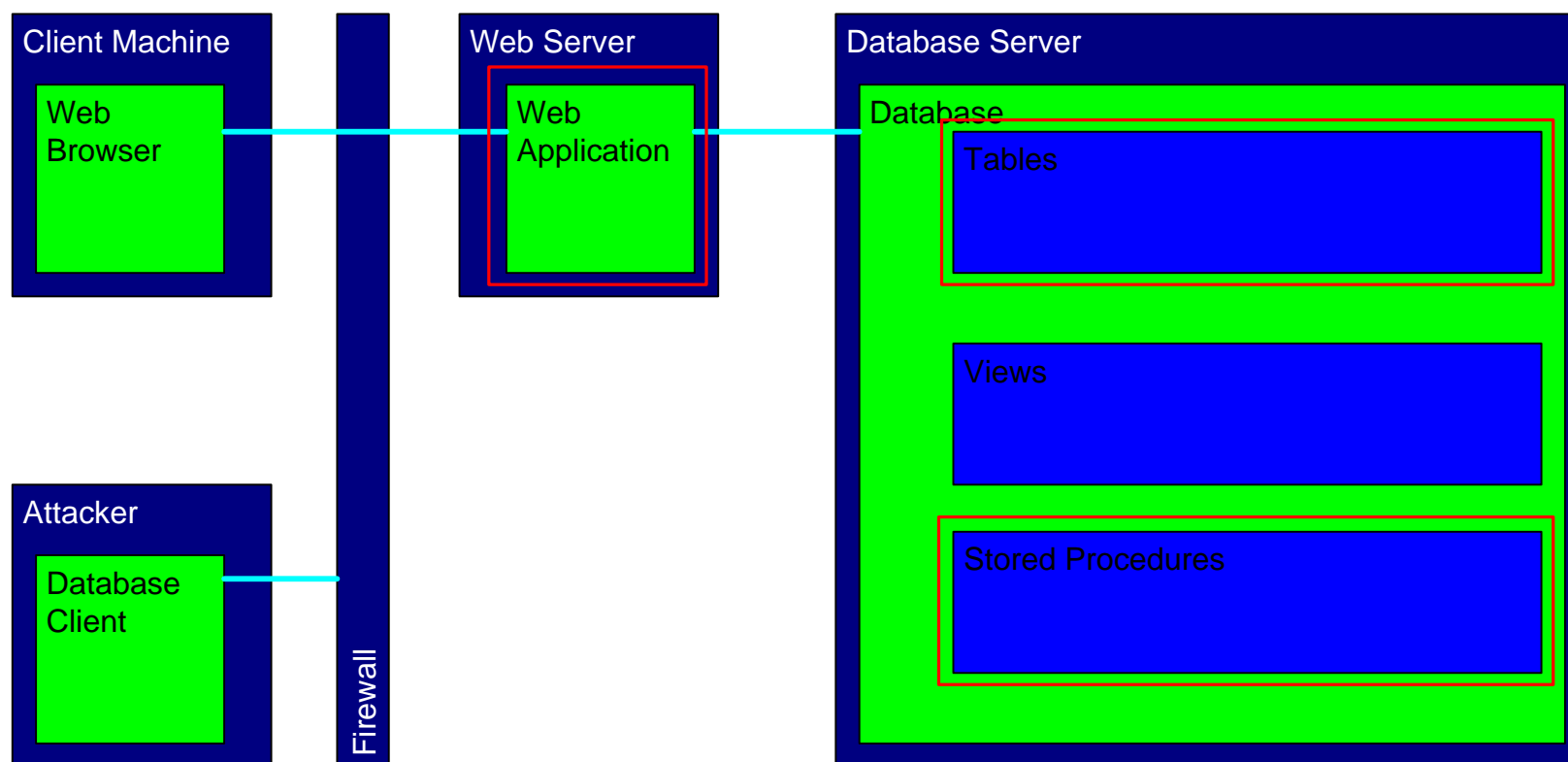
Threat: Database Compromise via Cracker or Backup Tape Loss



Threat: Database Compromise via Cracker or Backup Tape Loss

- Information Disclosure

Countermeasures: Database Compromise via Cracker or Backup Tape Loss



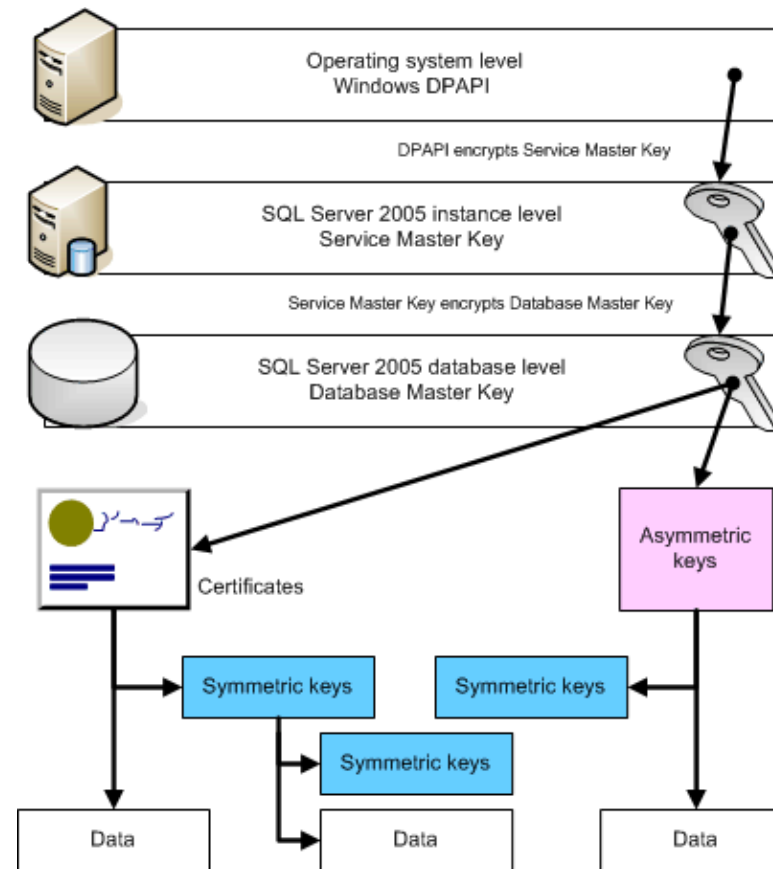
Countermeasures: Database Compromise via Cracker or Backup Tape Loss

- Encrypt data when stored in database tables
 - *Simple access to the table data is not enough to view sensitive information. An attacker must also know the appropriate key and be able to run decryption code.*
- Changes to implement encryption and decryption can be made in several areas:
 - *Web application*
 - *Stored procedures*
- Most modern databases have built-in encryption routines

Countermeasures: Database Compromise via Cracker or Backup Tape Loss

- SQL Server 2005
 - *Uses a hierarchy of encryption keys starting at the Windows DPAPI level (machine key), through the service key and database key and to specific keys*
 - *Has implementations of a variety of symmetric encryption algorithms*
 - RC4, RC2, DES, AES
 - *Has implementation of asymmetric encryption*
 - RSA (variety of key lengths from 512 to 2048)
 - *EncryptByKey and DecryptByKey functions*
 - *EncryptByKeyAutoCert, DecryptByKeyAutoCert*
 - *OPEN SYMMETRIC KEY, CLOSE SYMMETRIC KEY SQL syntax*
 - *See:*
<http://www.microsoft.com/technet/itsolutions/msit/security/sqlatsec.mspx>

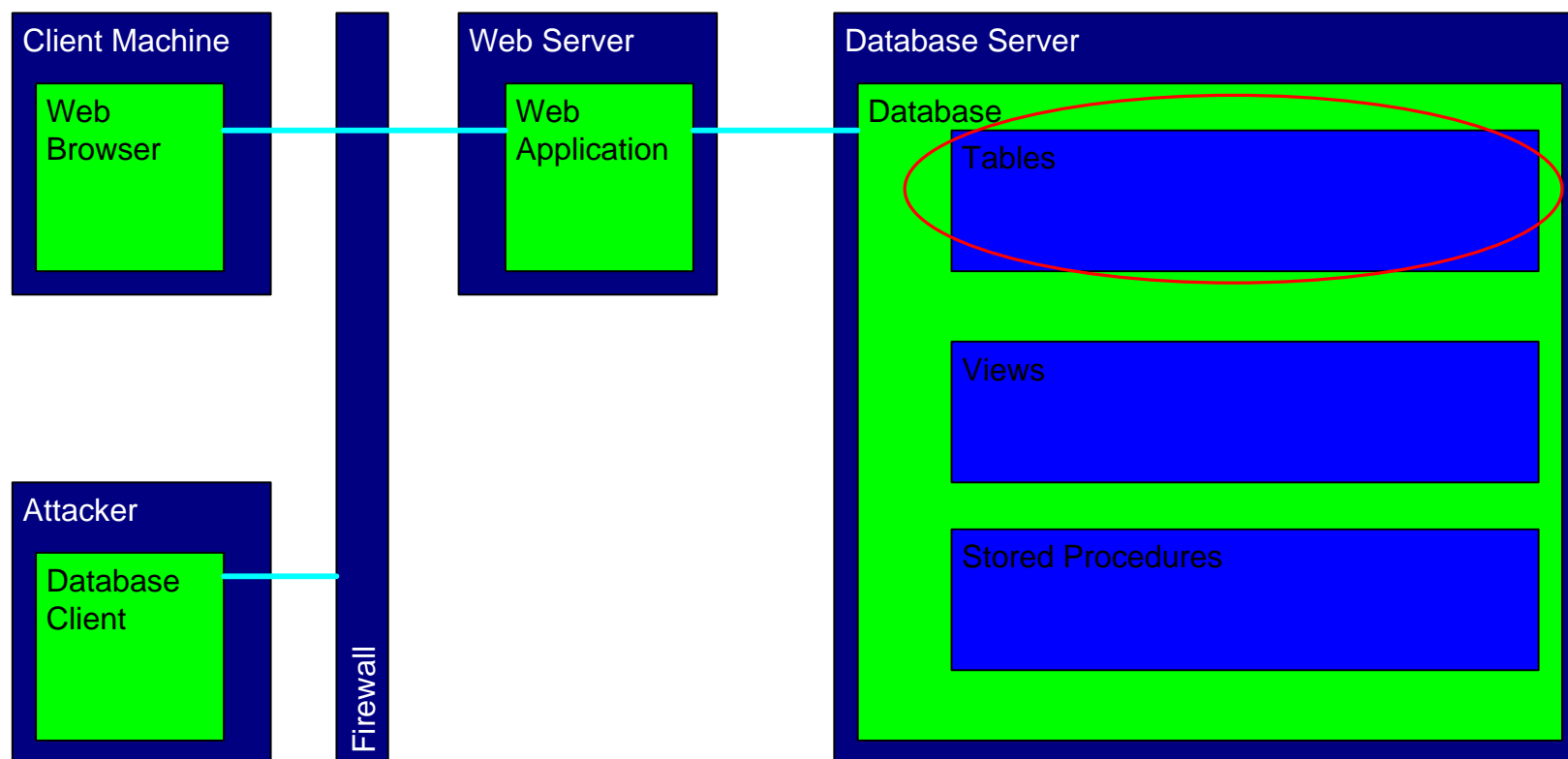
Countermeasures: Database Compromise via Cracker or Backup Tape Loss



Countermeasures: Database Compromise via Cracker or Backup Tape Loss

- MySQL 5.0
 - *Variety of encryption and decryption functions*
 - AES_ENCRYPT, AES_DECRYPT
 - DES_ENCRYPT, DES_DECRYPT
 - ENCODE, DECODE
 - *Variety of hashing functions*
 - ENCRYPT, PASSWORD, OLD_PASSWORD, MD5, SHA1
 - See: <http://dev.mysql.com/doc/refman/5.0/en/encryption-functions.html>

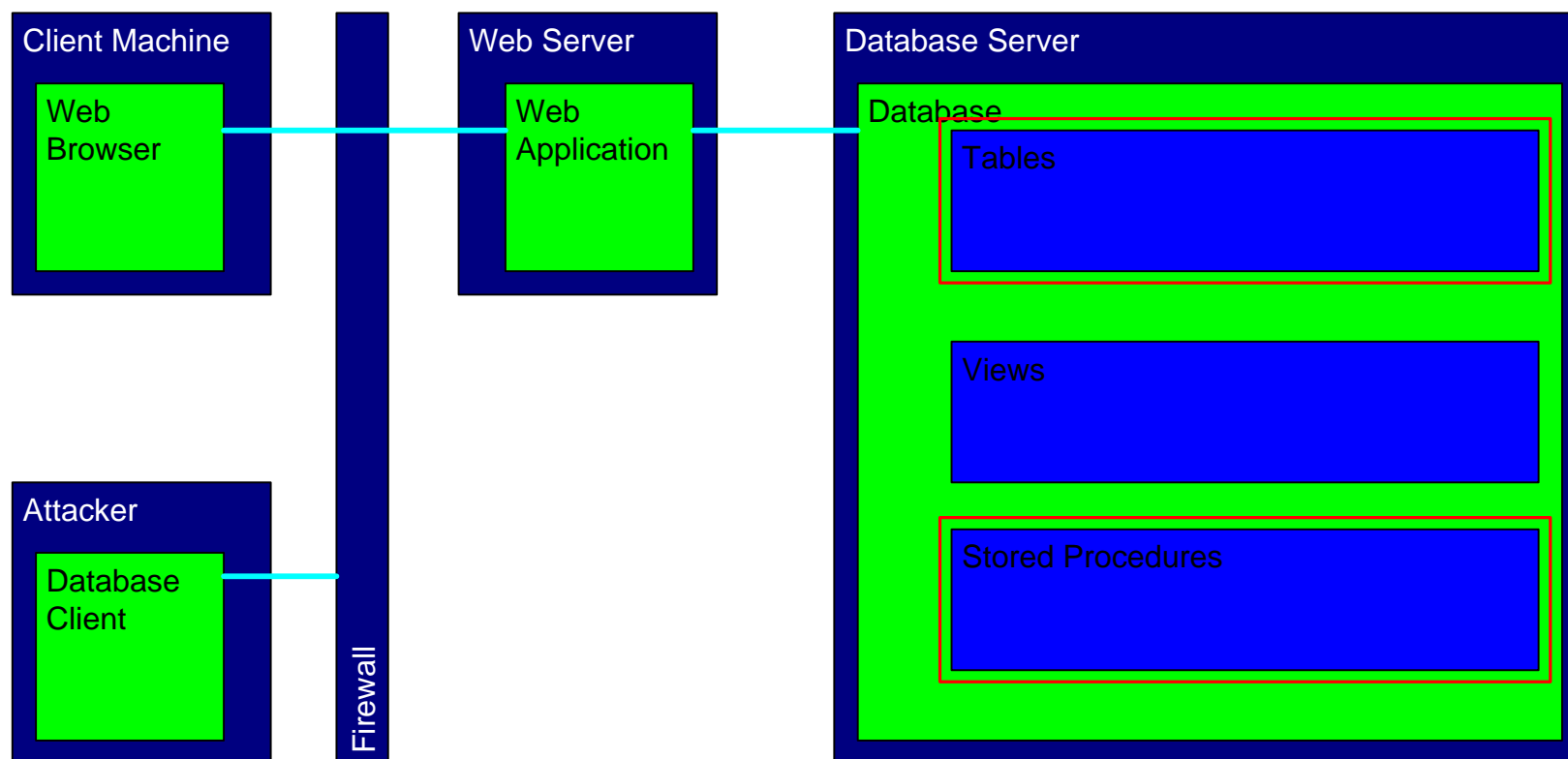
Threat: Unauthorized Writes to Data



Threat: Unauthorized Writes to Data

- Tampering
- Repudiation

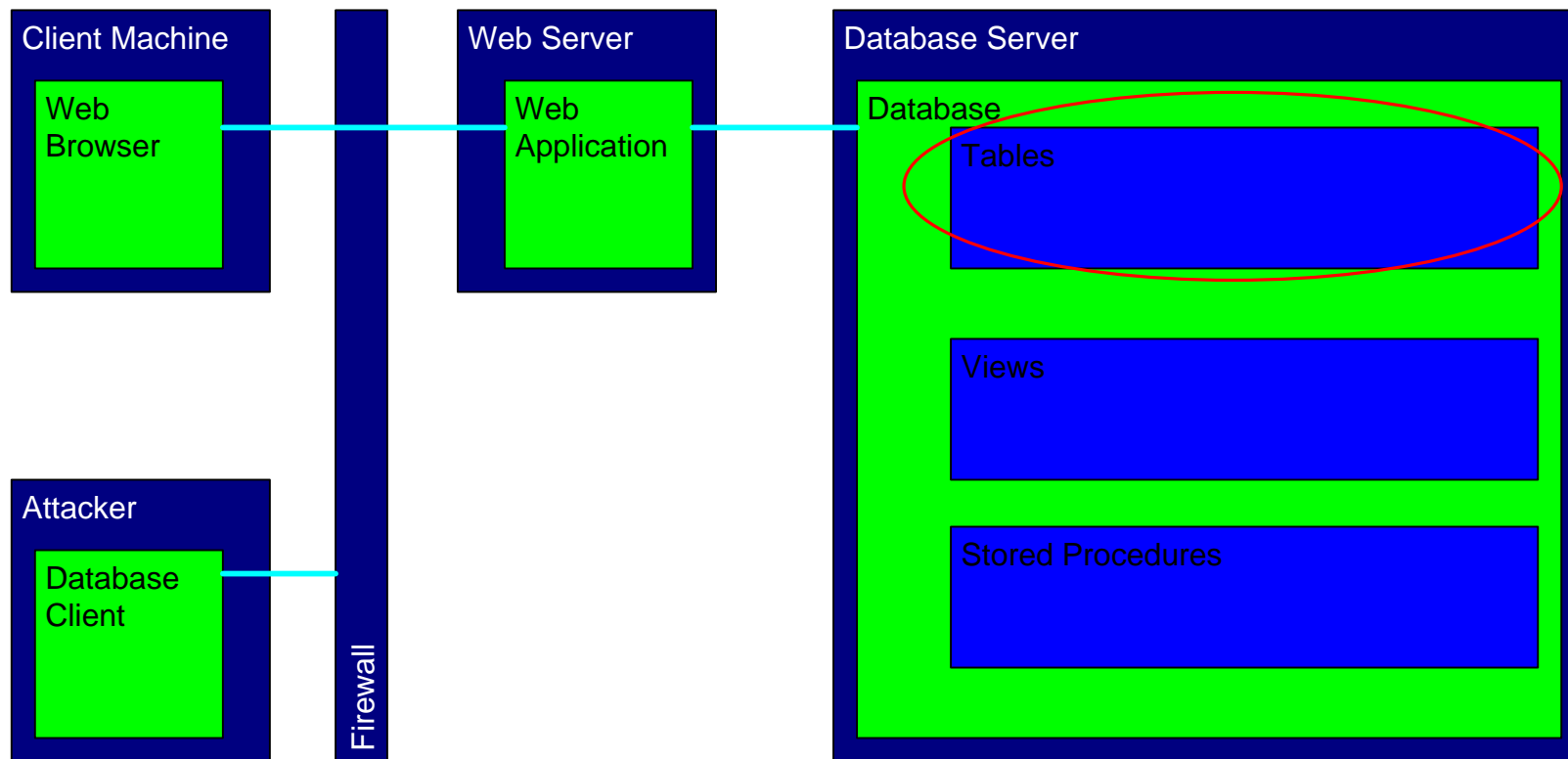
Countermeasures: Unauthorized Writes to Data



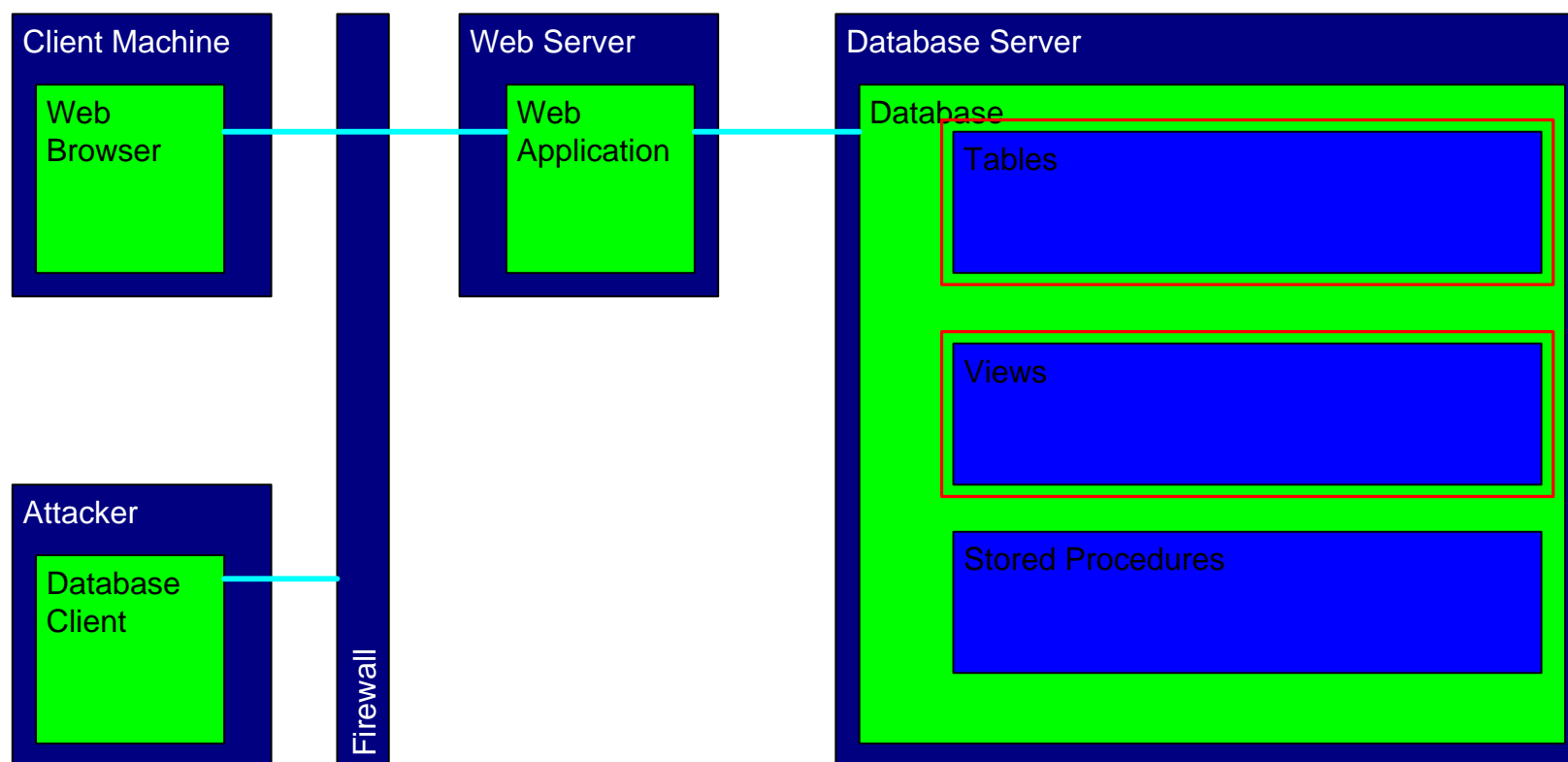
Countermeasures: Unauthorized Writes to Data

- Do not allow web application users direct access to execute CREATE or UPDATE queries on tables
- Use stored procedures instead when creating and updating data. Implement fine-grained control over access to these stored procedures.
- SQL Server Management Studio allows control of these permissions via the GUI
 - *Ownership Chains allow for table/stored procedure permission splits*
- MySQL allows you to manage these explicitly with GRANT syntax
 - *GRANT EXECUTE ON PROCEDURE 'Database','procProcedure' TO Username IDENTIFIED BY 'password'*
 - *See: <http://dev.mysql.com/doc/refman/5.0/en/grant.html>*

Threat: Unauthorized Reads from Data



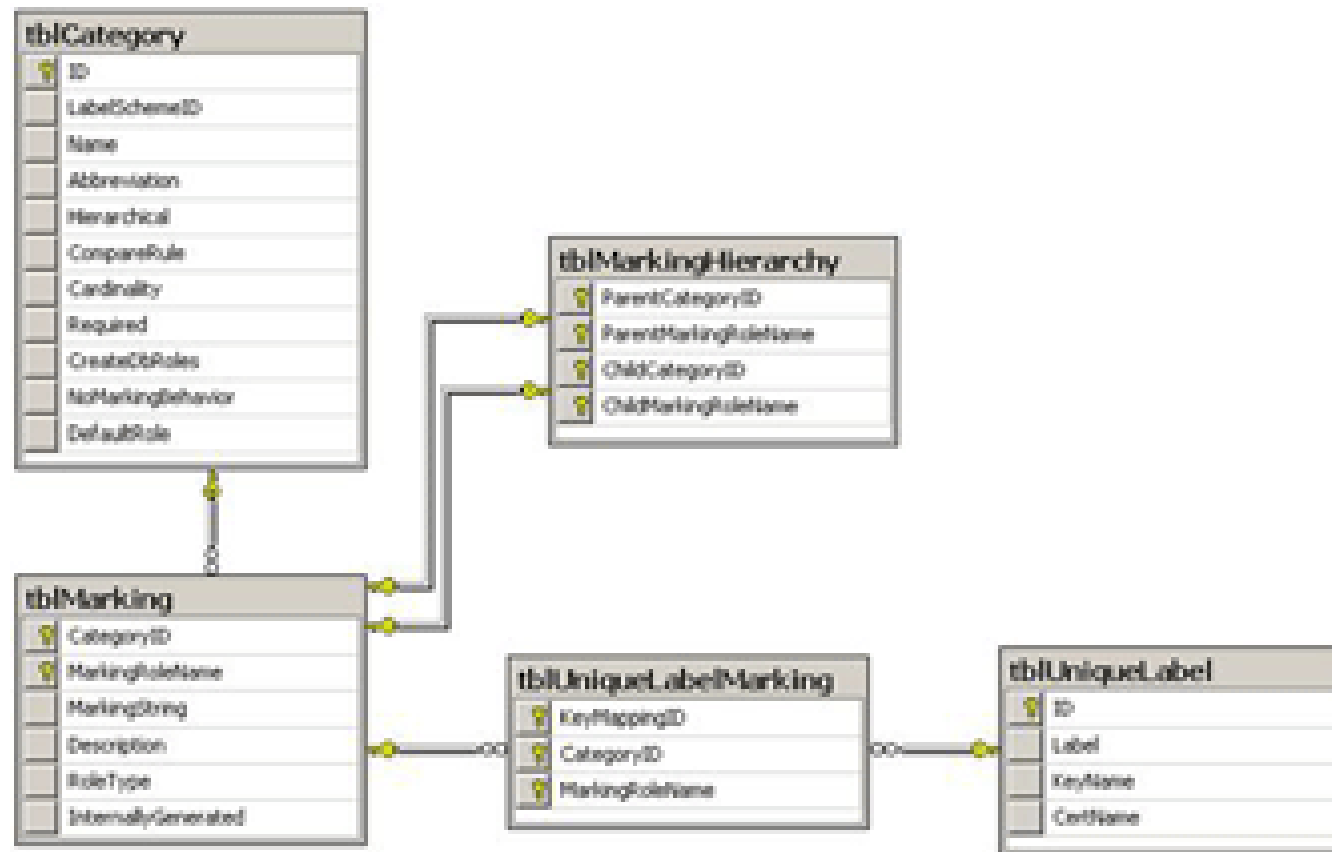
Countermeasures: Unauthorized Reads from Data



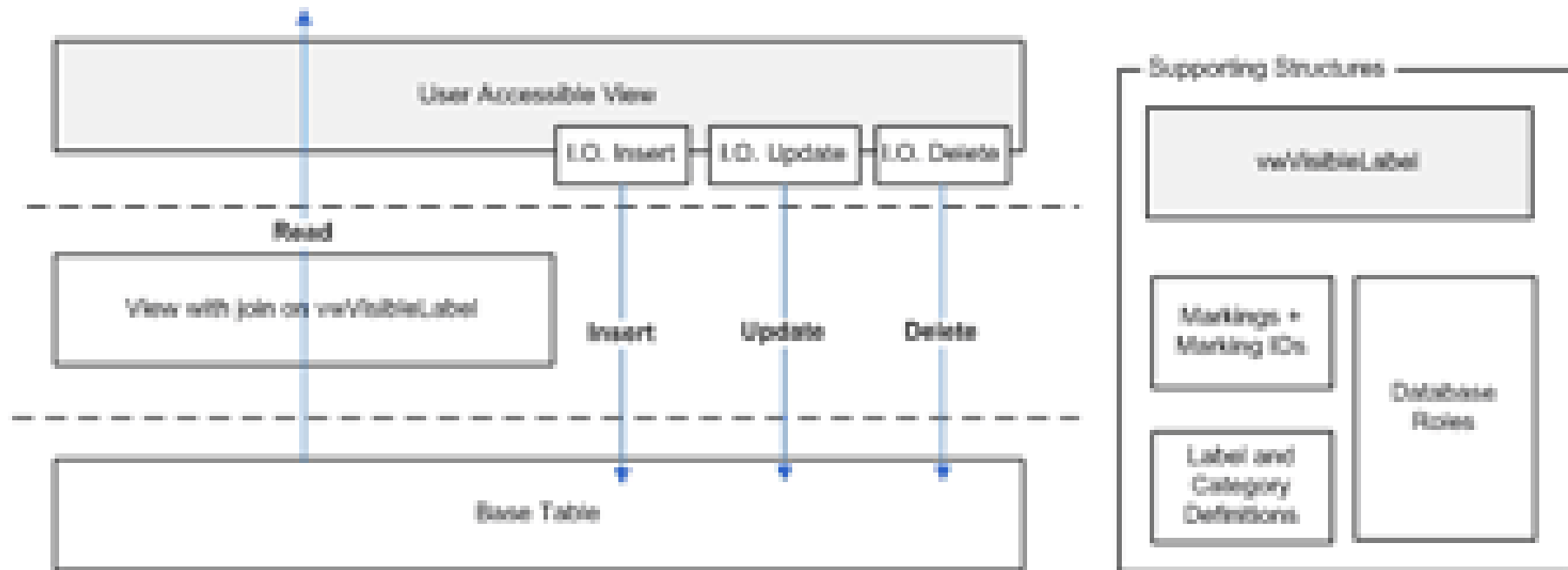
Countermeasures: Unauthorized Reads from Data

- Do not allow direct read access to tables
- Create views and allow access to the views
- Row-level security
 - <http://www.microsoft.com/technet/prodtechnol/sql/2005/multisec.mspx>
 - *Performance impact is significant due to effective loss of connection pooling*
 - *SELECTs are easy to handle, but what about INSERT, UPDATE, DELETE?*
 - “read-down, write-down” or “read-down, write-up”
 - Instead-of triggers
 - *CURRENT_USER in SQL Server*
 - *SESSION_USER() in MySQL*

Countermeasures: Unauthorized Reads from Data



Countermeasures: Unauthorized Reads from Data



Conclusions

- It is a scary world out there
- Modern database engines have a number of tools and technical features that can help
- Implement these as appropriate to accomplish specific security goals

Questions

Dan Cornell

dan@denimgroup.com

<http://www.denimgroup.com/>

<http://www.denimgroup.com/Knowledge>