THE EXPERT'S VOICE® IN ORACLE

Expert Oracle and Java Security

Programming Secure Oracle Database Applications With Java

PRACTICE A HOLISTIC VIEW TO SECURITY THAT PROTECTS PRECIOUS DATA ON THE CLIENT, ON THE SERVER, AND WHILE MOVING IN BETWEEN

David Coffin

Apress®

Petitioner has added letters to page (a) and romanettes to pages (i)-(iii). Otherwise, it leaves the original page

MICROSOFT CORP.
EXHIBIT 1005

Expert Oracle and Java Security

Programming Secure Oracle Database Applications with Java

David Coffin

Apress®

Expert Oracle and Java Security: Programming Secure Oracle Database Applications with Java

Copyright © 2011 by David Coffin

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and the publisher.

ISBN-13 (pbk): 978-1-4302-3831-7

ISBN-13 (electronic): 978-1-4302-3832-4

Trademarked names, logos, and images may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, logo, or image we use the names, logos, and images only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

President and Publisher: Paul Manning
Lead Editor: Jonathan Gennick
Technical Reviewer: Josh Juneau
Editorial Board: Steve Anglin, Mark Beckner, Ewan Buckingham, Gary Cornell, Jonathan Gennick, Jonathan Hassell, Michelle Lowman, Matthew Moodie, Jeff Olson, Jeffrey Pepper, Frank Pohlmann, Douglas Pundick, Ben Renow-Clarke, Dominic Shakeshaft, Matt Wade, Tom Welsh
Coordinating Editor: Adam Heath
Copy Editor: Tracy Brown
Compositor: Bytheway Publishing Services
Indexer: BIM Indexing & Publishing Services
Artist: SPI Global
Cover Designer: Anna Ishchenko

Distributed to the book trade worldwide by Springer Science+Business Media, LLC., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com.

For information on translations, please e-mail rights@apress.com, or visit www.apress.com.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales–eBook Licensing web page at www.apress.com/bulk-sales.

The information in this book is distributed on an “as is” basis, without warranty. Although every precaution has been taken in the preparation of this work, neither the author(s) nor Apress shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in this work.

The source code for this book is available to readers at www.apress.com. You will need to answer questions pertaining to this book in order to successfully download the code.

In memory of my dad, Colonel Monty D. Coffin,
the best teacher I ever had
--David Coffin

Contents at a Glance

- About the Author xxii
- About the Technical Reviewer xxiii
- Acknowledgments xxiv
- Introduction xxv
- Chapter 1: Introduction 1
- Chapter 2: Oracle Database Security 5
- Chapter 3: Secure Java Development Concepts 27
- Chapter 4: Java Stored Procedures 51
- Chapter 5: Public Key Encryption 63
- Chapter 6: Secret Password Encryption 85
- Chapter 7: Data Encryption in Transit 111
- Chapter 9: Two-Factor Authentication 177
- Chapter 10: Application Authorization 209
- Chapter 11: Enhancing Security 267
- Chapter 12: Administration of Security 329
- Appendix A: List of Methods from OracleJavaSecure Class 403
- Appendix B: Oracle Procedures, Functions and Triggers for Oracle and Java Security 407
- Index 413

Contents

- **About the Author** xxii
- **About the Technical Reviewer** xxiii
- **Acknowledgments** xxiv
- **Introduction** xxv
- **Chapter 1: Introduction** 1
  - Requirements 1
  - For Windows and UNIX/Linux Users 1
  - Background 2
  - How to Use This Book 2
  - Organization of This Book 3
  - Java Objects and Oracle Database Structures 3
  - Chapter Review 3
- **Chapter 2: Oracle Database Security** 5
  - Finding a Test Oracle Database 5
  - Working from an Existing Oracle Database 5
  - Oracle Users and Schemas 6
  - SQL*Plus, SQL Developer, JDeveloper, or TOAD 7
  - Organization of the Next Few Sections 7
  - Working as the SYS User 7
    - System Privileges 8

Roles ... 8
Security Administrator User ... 11
Security Administrator Role ... 12
The Audit Trail ... 14
The Data Dictionary ... 15
Working as the Security Administrator ... 15
Acquire secadm_role from a SQL*Plus Local Connection ... 16
Toggle Between Roles ... 16
Create an Application Security User ... 17
Create an Application User ... 18
Create the HR View Role ... 18
Audit Changes to Security Administrator Procedures ... 20
Audit Failed Attempts to Access HR Data ... 20
Working as the HR Schema User ... 21
Sensitive Data in the HR Sample Schema ... 21
Public View of Employees ... 22
Sensitive View of EMPLOYEES ... 22
Test Application User Access ... 23
Audit Trail Logs for the Sensitive View ... 24
Regarding Synonyms ... 24
Chapter Review ... 25
Chapter 3: Secure Java Development Concepts ... 27
Java Development Kit ... 27
Oracle Java Database Connectivity ... 27
JAR File Directory Separator ... 28
Java Packages ... 28
Development at Command Prompt ... 28

Environment 28
Beginning Java Syntax 30
Byte Code Compilation and the Java Virtual Machine 31
Java Code and Syntax Concepts 33
Methods 34
Values 34
Members 35
Objects 36
Classes and Null 36
Garbage Collection 37
Primitives 37
Strings 37
Static Modifier and the main() Method 38
Public and Private Modifiers 39
Exceptions 40
Exception Handling Syntax 40
Exception Handling Approaches 47
Java Virtual Machine Sandbox 49
Chapter Review 50
Chapter 4: Java Stored Procedures 51
Java Stored Procedure Example 51
Acquiring the Privilege to Load a Java Stored Procedure 53
Loading Java in the Oracle Database 53
Handling Exceptions in a Java Stored Procedure 53
Calling Oracle Database from Java 55
Method Syntax in Java Stored Procedures 57
Calling Java from Oracle Database 57
Installing and Testing the Example Code 58

Review The Roster of Participants 59
Cleaning Up 60
The Oracle Java Virtual Machine 60
Oracle JVM Based on Java SE 1.5 60
A Separate JVM for Each Oracle Session 61
Oracle JVM Sandbox 61
Auto-Commit Disabled in the Oracle JVM 61
Chapter Review 62
Chapter 5: Public Key Encryption 63
Generate Keys on the Client 63
RSA Public Key Cryptography 64
Java Code to Generate and Use RSA Keys 64
Creating a Set of Keys 64
Hand the Public Key Across the Network 65
Serialize Objects 66
Building the Public Key from Artifacts 66
Generating the RSA Cipher 67
Using the RSA Cipher 68
Getting RSA Public Key Artifacts 70
Using Static Methods and Private Constructor 71
Instantiating a Connection Member from a Static Initializer 71
Using One Code for Both Client and Server 72
Testing on the Client 73
Writing the main() Method 73
Running the Code 74
Key Exchange 75
Creating a Function to Encrypt Data with Public Key 75
Creating a Procedure to get SYSDATE in Encrypted Form 76

Loading OracleJavaSecure Java into Oracle Database 78
Encrypting Data with Public Key 78
Use Stacked Calls 79
Decrypting Data with Private Key 79
Testing on Client and Server 80
Using IN and OUT Parameters in an OracleCallableStatement 80
Handle Errors Reported by Oracle Database 81
Decrypting at the Client 82
Running Our Code Again 82
Observing the Results 82
Removing the Demonstration Oracle Structures 83
Chapter Review 84
Chapter 6: Secret Password Encryption 85
Approach 85
Java Code for Secret Password Encryption 86
Sharing the Artifacts of a Secret Password Key 86
Initializing Static Class Members 87
Evaluating the Java 1.5 Password-Based Encryption Bug 88
Coding an Automatic Upgrade: Negotiated Algorithm 88
Generating the Password Key 88
Encrypting with the Public RSA Key 89
Returning Secret Password Key Artifacts to the Client 90
Encrypting Data with Our Secret Password 92
Oracle Structures for Secret Password Encryption 93
Package to Get Secret Password Artifacts and Encrypted Data 93
Application Security Package Specification 93
Application Security Package Body: Functions 95
Application Security Package Body: Procedures 95

Java Methods for Secret Password Decryption....97
Decrypting Data Using the Secret Password Key....97
Decrypting the DES Passphrase using RSA Private Key....98
Ancillary Methods for Array Conversion....99
Method Used to Show Actual Algorithm....100
Testing DES Encryption on the Client Only....100
Running the Code....102
Observing the Results....102
Coding to Test Client/Server Secret Password Encryption....102
Setting the Code to Test Server as well as Client....102
Consider the Server Portion of the main() Method....103
Getting the DES Secret Password from Oracle....103
Seeing the Negotiated Algorithm for Password-Based Encryption....104
Calling Oracle Database to get Encrypted Data....105
Testing Oracle Database Encrypt and Local Decrypt Data....106
Sending Encrypted Data to Oracle....107
Testing Our Secure Client/Server Data Transmission....107
Chapter Review....108
Chapter 7: Data Encryption in Transit....111
Security Administrator Activities....111
Granting More System Privileges to the Application Security User....112
Permitting Users to Execute Packages in Other Schemas....112
Application Security User Activities....112
Creating a Table for Error Logging....113
Creating a Table for Managing Our Error Log Table....114
Creating an Error Log Management Procedure....115
Creating a Trigger to Maintain the Error Log Table....116

Updating the Application Security Package 117
Creating an Error Logging Procedure 118
Executing Package Specification and Body 118
Methods for Using and Testing Encryption in Transit 119
Method to Build the Secret Password Key 119
Temporary Method to Reset All Keys 120
Loading Updated OracleJavaSecure Class into Oracle 121
Security Structures for the HR User 121
Exploring Privileges That Enable HR Tasks 121
Creating the HR Security Package 122
Selecting Sensitive Data Columns from EMPLOYEES 122
Selecting All Data as a Single Sensitive String 124
Selecting Sensitive Data for an Employee ID 125
Revising Procedure to Get Shared Passphrase 125
Updating Sensitive Data Columns in EMPLOYEES 125
Avoiding SQL Injection 129
Demonstrating Failure to SQL Inject in Stored Procedure 130
Executing the HR Package Specification and Body 131
Inserting an EMPLOYEES Record: Update a Sequence 131
Demonstrations and Tests of Encrypted Data Exchange 133
Some Preliminary Steps 133
Selecting Encrypted Data from EMPLOYEES 135
Selecting All Columns in Encrypted String 137
Sending Encrypted Data to Oracle Database for Insert/Update 138
Selecting a Single Row from EMPLOYEES 139
Selecting EMPLOYEES Data by Last Name: Try SQL Injection 139
Selecting EMPLOYEES Data by RAW: Try SQL Injection 140
Testing Encryption Failure with New Client Keys 140

Testing Failure with New Oracle Connection 141
Some Closing Remarks 141
Executing the Demonstrations and Tests 142
Observing the Results 142
Demonstrating Scenarios 143
Querying Employees to See Updates 144
Packaging Template to Implement Encryption 144
Template for Oracle Application Security Structures 144
Template for Java Calls to Application Security 145
Java Archive for Use by Applications 145
Don't Stop Now 145
Chapter Review 146
Chapter 8: Single Sign-On 149
Another Layer of Authentication? 149
Who Is Logged-In on the Client? 150
Find a Better Source of OS User Identity 150
Use NTSystem or UnixSystem to Get Identity 150
Do Cross-Platform-Specific Coding with Reflection 151
Assure More Stringent OS Identity 152
Access Oracle Database as Our Identified User 154
Examine the Oracle SSO Options for Programmers 154
Set a Client Identifier 155
Prepare to Access HR Data 155
Update p_check_hrview_access Procedure, Non-Proxy Sessions 156
Assure Client Identifier and OS_USER 157
Audit Activity with Client Identifier Set 157

Proxy Sessions....158
Create Individual Person Users in Oracle....158
Proxy from Users IDENTIFIED EXTERNALLY....159
Establish a Proxy Session....159
Update p_check_hrview_access Procedure, Proxy Sessions....161
Audit Proxy Sessions....161
Using Connection Pools....162
Proxy Connections from an OCI Connection Pool....162
Proxy Sessions from a Thin Client Connection Pool....166
Universal Connection Pool....169
Application Use of Oracle SSO....171
Our Example Application Oracle SSO....172
Updates to OracleJavaSecure....173
A Code Template to Give Developers....175
Chapter Review....175
Chapter 9: Two-Factor Authentication....177
Get Oracle Database to Send E-Mail....178
Installing UTL_MAIL....178
Granting Access to UTL_MAIL....179
Testing Sending E-Mail....180
Getting Oracle Database to Browse Web Pages....181
Delegating Java Policy to Security Administrator....181
Permitting Application Security User to Read Web Pages....182
The Two-Factor Authentication Process....183
Security Considerations for Two-Factor Distribution Avenues....183
Security Issues with Two-Factor Delivery to E-Mail....183
Security Issues with Two-Factor Delivery to Pagers....184

Security Issues with Two-Factor Delivery to Cell Phones 184
Preferred Two-Factor Delivery 185
Oracle Structures Supporting Two-Factor Authentication 185
Creating the SMS Carrier Host Table 185
Creating a Table of Employee Mobile Numbers 187
Accessing HR Tables from Application Security Procedures 190
Create the Two-Factor Codes Cache Table 190
Testing Cache Aging 191
Verifying Current Cached Two-Factor Pass Code 192
Sending Two-Factor Pass Codes 193
Updating the Secure Application Role, HRVIEW_ROLE Procedure 193
Update OracleJavaSecurity.java for Two-Factor Authentication 194
Setting Some Company-Specific Addresses 194
Compile Two-Factor Delivery Route Codes: Binary Math 195
Exploring a Method to Distribute the Two-Factor Codes 195
Distributing the Code to SMS 201
Distributing the Code to Pager URL 202
Distributing the Code to E-Mail 203
Testing Two-Factor Authentication 203
Updating OracleJavaSecure Java in Oracle 204
Editing the Test Code 204
Planning to Pass the Two-Factor Code as an Argument to Main 205
Planning to Acquire the Secure Application Role 205
Running the Tests and Observing the Results 206
Chapter Review 207
Chapter 10: Application Authorization 209
Secure Application Role Procedure for Multiple Applications 210
Rebuild Two-Factor Cache Table for Multiple Applications 211

Update Two-Factor Code Functions to Use Application ID 211
Move Test for SSO to Separate Function 212
Add an Oracle Package for Use Only by Application Security 212
Add Helper Function to Get APP_ROLE 213
Replace Procedure for hrview_role Access with Dynamic Procedure 213
Rewrite and Refactor Method to Distribute Two-Factor Code 216
Procedure to get Employee Addresses for Two-Factor Code Delivery 216
Stored Procedure to Update Two-Factor Code Cache 217
Changes to the Method to Distribute Two-Factor Codes 218
Update to Two-Factor Distribution Formats 219
Application Authorization Overview 220
User for Application Authorization 221
A New Profile with Limits and Unlimited 221
Application Verification User 222
The Application Verification Logon Trigger 222
Application Verification Logon Procedure 223
Get Off Function 223
Function to Find Database User 224
Proxy Through Application Verification and Other Proxies 225
Auditing Application Verification 226
Structures for Application Authorization 226
More Space for Application Security 226
Application Connection Registry Table 227
A Set of Connection Strings for an Application 227
An Inner Class to Represent the Application 228
Implement an Inner Class in OracleJavaSecure 229
Deserialization and Version UID 230
Set Application Context 231

Format the User-Input Two-Factor Code 232
Save Connection Strings from the Client Perspective 233
Method to Put Connection Strings in the List for an Application 233
Client Call to Store List of Connection Strings on Oracle 234
Save Connection Strings from the Server Perspective 236
Function to Call Java to Decrypt the List of Connection Strings 236
Method to Store List of Connection Strings for Application 237
Oracle Procedures to Get Entries from the Application Registry 243
Get an Application Connection String: The Java Client Side 244
Get an Oracle Connection from the List for an Application 245
Get List of Connection Strings from Oracle Database to Client App 246
Establish a Connection for Application Verification Processes 249
Get a List of Application Connection Strings: The Server Side 250
Test Application Authentication, Phase 1 251
Get New Structures into Oracle 252
Review Steps of Testing 252
Set the Application Context 253
Call to Get Application Connections 254
Send List of Connection Strings to Oracle Database for Storage 254
Get a Unique Connection for Use in This Application 254
Use or Lose Initial Application Verification Connection 255
Get an Application Connection and the Associated Secure Application Role 255
Get Encrypted Data with the Application Connection 256
Add More Application Connection Strings 256
Testing a Second Application 257
Objects We Have Never Seen 257
Place Stub Class on Oracle 257
Get Application Authentication Connection and Role 259

Test Application Authentication, Phase 2 260
Set the Application Context 261
Store the Connection Strings in Oracle 261
Get an Application Connection with Role 261
See the Proxy Connection 262
Get Encrypted Data from Oracle 262
Chapter Review 262
Chapter 11: Enhancing Security 267
Hide the APPVER Connection String 267
Get It from a Second Source/Server 268
Get It from a Native Call: JNI 269
Get It from an Encrypted Java Class 269
Get It from an Encrypted String 270
Get It from an Encoded String 270
Create an Oracle Client Wallet 277
Install the Oracle Client 278
Create the Wallet 278
Use the Wallet from SQL*Plus 280
Use the Wallet from Java 280
Administer Wallet Security 281
Trace Oracle Client Code 282
Logging Oracle Thin Client Trace Data 283
Encrypt Data Stored on Oracle Database 285
DBMS_CRYPTO Package 285
Passwords and Keys 285
Encryption at Rest Key Store 286
Functions to Encrypt/Decrypt Data at Rest 287
Wrap Utility 289

Changes to setDecryptConns()/getCryptConns() 290
Manage Connection Strings for Applications 291
Create an Application Administrative User 291
Create an Administrative Role for Application Verification 292
Delete Connection Strings 293
Copy Connection Strings from Previous Version of Application 293
Add Other Authentication Credentials 301
Update Application Security Structures 301
Authenticate on a Separate Oracle Instance 302
Create a New Oracle Database Instance 303
Create a New Oracle Service 304
Write the Create Database Command 305
Create and Configure the Database 307
Create a Database Link to the ORCL Instance 313
Revoke PUBLIC Grant on Sensitive Data Dictionary Views 316
Create the Remaining Structures for Application Authorization 317
Create Java Structures 319
Remove Application Verification from the ORCL Instance 319
Test Enhanced Security 320
Encode the APPVER User Password for APVER Instance 320
Edit the Application Passwords to Be Used 321
Run Main to Test 322
Run Main to Copy Connection Strings to New Version 323
Test from a Different Application, TestOracleJavaSecure 324
Compile and Run as Administrative User, OSADMIN 326
Run as Non-Administrative User, OSUSER 327
Chapter Review 328

Chapter 12: Administration of Security 329

A Security Administration Interface 329

Application Login Screen 330

The Application Inner Class 331

Center Method 333

Login Screen Constructors 334

The "Wait While Processing" Modal Dialog 335

Background Processing Thread 336

The Continue Button 337

The Login Screen Closes 338

Security Administration Menu 339

Add/Modify User Functional Screen 341

Instantiate the AddUser Screen 342

Initialize the Data Selection Components 345

Select an Existing Employee 347

Create a New Employee 352

Save Data for the Employee 352

User Administration Screen 354

Create the OJSAAdm User 355

Enable the OJSAAdm User Across a Database Link 359

Select an Existing User 361

Save Updates to the Administrative Privileges 362

Revoke User Access to Run Applications 364

Application Assignment Screen 365

Initializing the Data Selection Components 366

Selecting an Available Proxy in the Table 368

Selecting a User from the List 369

Adding a Proxy to the User's List 369

Removing a Proxy from the User's List 370
Saving Updates to the User's Proxies. 370
Application Registration Screen 372
The Application Verification Administrator Role. 374
The Create App Class Button. 375
Tables of Specific Application Administrators and Application to Class Registry 377
Security Table Access Analysis. 379
The Register Application Button. 382
Application Selection Screen 385
Initializing the List of Applications. 386
Selecting the Manage Selected Application Button. 386
Connection String Editor 388
Initializing the List of Connection Strings. 389
Selecting an Existing Connection String. 390
Updating a Connection String in the List. 390
Saving the List of Connection Strings to the Database 390
Connection String Copy Screen 391
Limiting Certain Administrators to Certain Applications 391
Virtual Private Database 392
Adding a Dynamic Where Clause to Procedures 396
Adding a Dynamic Where Clause to a View. 397
Scripts Execution and Code Compilation 398
Final Updates to OracleJavaSecure 399
Single Oracle Instance Code 399
Bootstrap OJSAdmin 399

Appendix A: List of Methods from OracleJavaSecure Class 403

Appendix B: Oracle Procedures, Functions and Triggers for Oracle and Java Security 407

Index 413

About the Author



David Coffin has over 30 years' experience in multi-platform network integration and systems programming. He has worked for large government contractors throughout his career. As a perpetual student, he has earned one Master's degree and has started a couple others. As a family man, he has raised eight children. David is a tri-athlete and distance swimmer who competes in the middle of the pack. He is also a classical guitar player, but he's not quitting his day job.

About the Technical Reviewer



Josh Juneau has been developing software since the mid-1990s. Database application programming has been the focus of his career since the beginning. He is an Oracle database administrator and has adopted the PL/SQL language for performing administrative tasks and developing applications for Oracle database. As his skills evolved, he began to incorporate Java into his PL/SQL applications, and later began to develop stand-alone applications in Java. During his tenure as a developer, he has combined his knowledge of PL/SQL and Java to develop robust Oracle database applications that harness the great features offered by both technologies. He has extended his knowledge of the JVM by learning and developing applications with other JVM languages, such as Jython and Groovy. Since 2006, Josh has been the editor and publisher for the *Jython Monthly* newsletter. In late 2008, he began a podcast dedicated to the Jython programming language. Josh was the lead author for *The Definitive Guide to Jython*, and *Oracle PL/SQL Recipes*, both published by Apress. He is the lead for the Django-Jython project (`http://code.google.com/p/django-jython/`). He loves to spend time with his four wonderful kids and excellent wife. To hear more from Josh, follow his blog at `http://jj-blogger.blogspot.com`. You can also follow him on Twitter via `@javajuneau`.

Acknowledgments

Writing this book was as much about expressing who I am as it is about the work I do. It was on New Year's Eve, 2009 that I decided I would write 100 pages of my own creation during 2010. I started writing an autobiography, but that turned out to be too complex. I have read Benjamin Franklin's autobiography, and maybe one day I will be that concise.

I decided to write about something that I think about day in and day out: computer security. Thankfully, I have one family member who gets this stuff—my son Matthew. Thank you, Matthew, for taking an interest and giving me encouragement. Also, thanks again for getting me through my refresher course on differential equations, 30 years after finishing college. You tutor me and I mentor you, all right big guy?

I wouldn't have gotten very far without the encouragement of my wife, Linda. I guess she sees the ideas bouncing around in my head, because she has always encouraged me to write. She claims I'll be like Emily Dickinson and have my writings published only after I die. Linda, I hope this assuages your concerns about sorting through my notebooks when I'm gone. Thanks for loving me through this process!

Kids, thanks for putting up with your dad's time on the computer. I love you!

I have had many technical mentors on the job. Many folks were willing to share their ideas and understanding with me, and also let me run with my own thoughts. Among my best mentors were my peers in the Intranet Development Group. I'm glad I have this opportunity to tell you guys how much you expanded my horizons during our time as a development team. I'm very glad we spent time in the Skunk Works together. A special thanks to our Fearless Leader—your ideas for application authentication came together into Chapter 10 of this book.

I also want to thank God. He has given me natural abilities in computer administration and security, and he has been my source of peace and joy for as long as I've known Him, and even before.

Introduction

Every day I read e-mails and articles dealing with computer security attacks and breaches, and the repercussions that follow. I have been getting onboard with computer security for decades, and there is no end in sight for this effort. Each and every professional in information technology has a commitment to keep with regard to computer security. We have, most of us, signed computing codes of conduct, and further we have aligned ourselves with those who protect the computing resources of our respective companies.

Of course, computer security professionals are on the front lines, and they draw the ranks of system and network engineers and administrators along into the fray. Hopefully these soldiers are joined by the software system administrators—database administrators and web services administrators. How often is that the limit of personnel dedicated to securing corporate resources on a daily basis?

Application programmers need to join this battle. Programmers cannot depend blindly on the work of those front-line combatants to protect their work and their careers. In the ranks of IT professionals, there are no civilians—we are all in the fight.

I speak like this is a war. Do I think we are dealing with so-called cyber warfare? No, not really. People are not dying. But the battle has become economic, and has threatened the wealth and economic security of each of us. This is more like theft, and we are more like guards and policemen.

You are a guard for the computer resources and data of your company, and you need to fulfill that responsibility. It is my intention to provide you tools and knowledge that you can put to immediate use. But even more than that, I want to get you thinking about how you can write bulletproof applications of your own. Your requirements will differ from those presented in this book, so you will be coming up with defenses of your own. It's my hope that, after working through this book and honing your skills, your arsenal will be greatly strengthened.

In Part 1, Chapters 2 through 4, we will build a foundation for secure programming in Java and Oracle, and their common ground, Java Stored Procedures (JSP). This is not an encyclopedic coverage of these expansive domains. Rather, I focus on the specific topics that touch on that special discipline of "secure" programming.

In Part II, Chapters 5 through 7, we explore a couple varieties of encryption, using only a couple sets of protocols. Once again, this is not encyclopedic. But after reading these chapters, you will be prepared to encrypt data in your applications, and over the network in a client-server application. You will find that, with Java running in the Oracle database as well as on the client computer, our tightly coupled client-server application can transmit sensitive data with bulletproof, session-specific encryption.

In Part III, Chapters 8 through 10, we will expand our toolset to include single sign-on, two-factor authentication and application verification. These are practices that will keep intruders from getting access to our applications, and will keep unauthorized applications and users from getting access to our Oracle credentials and data. Single sign-on has the added benefit of reducing the number of times a user needs to type his user ID and password. Two-factor authentication, on the other hand, requires that the user have a specific mobile phone or registered device to receive a code that authorizes

her access to the application. Application verification helps assure that an application is authorized access, and allows us to assign applications to users.

In Part VI, Chapters 11 and 12, we enhance the security we have been building and then build an administrative interface to get it running and keep it running. The security enhancements we will add include application-specific encryption on disk and use of a hardened Oracle database instance for application verification. In the administrative interface, we will discuss how the security data is maintained, and we will also look at some good practices for GUI application development in Java.

At several points in the book, we will prepare a template that we can provide to other application programmers so that they can implement the same security structures. The simplest and most complete template for implementing this security in a GUI application is presented in Chapter 12—the `Login` class. When you get to the end of that chapter, you will be the expert. You can do your part for computer security, and you can assist your associates in doing theirs. Thank you!

CHAPTER 1

Introduction

This book is a walk through Oracle and Java technology. I will weave the story of Oracle Database and Java security on the loom of these pages. The particular thread we will weave is code. This is a story for programmers.

This story will take you through several large tasks to help you start securing your Oracle applications. We will not be building any specific application, but will focus on the security aspects in building an application. In order to make this learning effort feel like a practical application, we will apply our efforts to the HR sample schema that is available with the Oracle Database 11g installation.

I hope to maintain a conversational tone, because I want to teach the concepts of secure programming. We are going to have "the talk" about secure programming. When you have read this book, you will be well equipped for the most difficult application programmer assignment of our day: defending our applications and computers.

Requirements

In order to work through the examples in this book, you will need an installation of Oracle Database 11g, Enterprise Edition, Release 11.2 or later. You will also need to have the Java Development Kit, 1.5 or later, installed on your workstation. That is it.

I will refer to some other products as we go along, but the intent of this book is to cover topics and approaches that will be useful to Oracle and Java programmers, rather than committing to any additional products. However, you should note that much of what we will do here can be acquired through commercial products from Oracle corporation and elsewhere.

Notably, two of the features we will be building in this book—data encryption over the network and data encryption on disk—are available in a product from Oracle called Oracle Advanced Security. That product is relatively easy to configure and use, although it is expensive. But adding encryption onto a badly secured database or application will just hide the things you, as a programmer, should be addressing. So even if you use Oracle Advanced Security, you would still do well to learn about programming secure Oracle Database applications with Java.

For Windows and UNIX/Linux Users

It is not a requirement that you be a Windows user to accomplish the tasks in this book. All of the code is in PL/SQL and Java, and will run cross-platform. However, I've had to be a bit focused in the development of this material. All of the descriptions of filesystem directories, command prompt, environment settings, services, scripts, and processes are written using Microsoft Windows as the model.

To avoid the appearance of a strong Windows bias, let me remind you that we are talking about Java and Oracle here. I've been using UNIX for longer than Windows has been around. I didn't start using

Windows until Windows 3.1, when the trumpet winsock tcp/ip stack became available. I started out in UNIX with SunOS 4.1 as a system administrator, along with Netware servers. Then for a few years I was a business UNIX system administrator. I ran dozens of machines and over a half-dozen flavors of UNIX. These I managed centrally with a lot of willpower and a thorough the application of scripting and Perl over NFS with scheduled *cron* updates, and by knowing the unique attributes of BSD, System 5, and AIX flavors. I combined that with standardization—for example, the Korn shell (or mksh) and lprNG.

So why not write this book for UNIX/Linux users? In the first place, you guys likely already know how to do what I'm saying in Windows-speak on your Linux systems. Secondly, I can talk about Windows and only rarely have to mention the differences between Windows 7 and Windows XP. There is no way I could talk about UNIX and Linux with that much clarity. UNIX is not an operating system; it is a foundation for many operating systems. That is also true of OSX.

Additionally, there is not just one command prompt, but many shells in UNIX. There is not just one UNIX batch/command language, but innumerable scripting options.

Background

So, why write or study code that can be purchased off the shelf? As a programmer, I believe in the importance of knowing how programs on my computers and servers work, even if I didn't write the programs. I think all programmers by nature enjoy seeing new functions in code; it is educational and empowering. My goal with this book is reader understanding, perhaps with a measure of career development built in.

And if it is your goal to achieve application security using the basic Oracle Database and Java services, without going out to spend your tens of thousands of dollars, this book will give you a foundation for doing just that. You can write secure Oracle applications with Java!

I will keep things clear and concise, so I don't clutter the message. This book is also intended as a primer (I like that word, no matter which way you pronounce it) for both Oracle Database and Java security. I am not going to assume too much foreknowledge, but if some terms are unfamiliar, I suggest a quick Internet search for clarification. Also, I am deeply indebted to the Oracle Documentation Library, and to Sun's (Oracle's) Java Documentation, and I recommend you keep links to those resources close for frequent reference. They can be found at:

`http://www.oracle.com/technetwork/indexes/documentation/index.html`
`http://download.oracle.com/javase/6/docs/`

How to Use This Book

You will need to refer to the source code. You can read this book from cover to cover and understand everything and become the expert you are meant to be. But you have other responsibilities and distractions. So you will need to leave and come back and get re-oriented from time to time. This book will proceed with the assumption that you are reading from cover to cover and have complete retention, but I've never really met anyone who does.

I much prefer assuming you are mortal, like me. You'll be asking yourself the questions, "Now, where was I, and what are we doing?" Also, even if you are willing to take it on faith that we are proceeding in a logical order and are actually making forward progress, reading on even though you might be a bit disoriented, you will still have the question, "What does this have to do with anything?"

These questions can be answered pretty easily with a quick reference to the code. You will want to be able to search the code for the keywords that you are reading about, find the code in question, and find references to it. You'll want to see the code in context, and you'll want to see how the code starts and runs. Now, I've incorporated large and small chunks of code throughout this text, and I don't talk about code without showing it to you, but you will want to see it in context.

I organized the text and code to easily walk hand in hand together. For the most part, you will only need to have one source code file open at a time as you read through the text. In Chapter 2, which is coming right up, you will complete Oracle Database security tasks as several different users—for each user, there is a corresponding source code file. The chapter is organized around those files. In fact, it would be best if you also open a SQL client and execute each portion of the code as it is discussed.

Organization of This Book

There are four main sections in this book. In Part I (Chapters 2–4), you will learn the technical ins and outs of secure programming in Oracle Database, secure programming in Java, and Java Stored Procedures (Java running in the Oracle Database engine). From novice Oracle PL/SQL programmers and novice Java programmers to experienced Oracle application programmers using Java, these chapters will fill in the missing concepts that you need to master for security' sake.

In Part II (Chapters 5–7), we build on the Java and Oracle Database secure programming foundation. We develop a strong layer of encryption—specifically, encryption in transit. You will build this for yourself using Java Cryptography Extension (JCE), rather than a purchased package or SSL service. Because encryption plays such a key (pun intended) role in secure programming today, you need to have a solid grasp of these data encryption concepts, and you also need something more: you need to know when to apply encryption and how to evaluate your encryption strategy to determine if you have successfully protected your data. We will cover all of that.

In Part III (Chapters 8–10), we get into some fun topics that will permit you to provide the solutions your company is seeking: single sign-on, two-factor authentication, and something you may not have realized you need: application identification and authorization. In application authorization, we will be providing a secure data store for our application passwords—one that not only avoids embedded passwords, but also provides enhanced security along with ease of management and central distribution.

In the last section, Part IV (Chapters 11–12), we will harden the security with additional data encryption in the Oracle Database store, with a further hardened Oracle Database and with some additional programming efforts, such as obfuscation. We also establish a simple user interface for administering the tables and data we've built to accomplish security, and do that securely.

Java Objects and Oracle Database Structures

In this book, I will be using certain terms non-interchangeably that are used interchangeably elsewhere. Often schemas, tables, indexes, and other items in Oracle Database are referred to as both *objects* and *structures*. Because *object* is a technical term in Java and not just another word for *thing*, I will reserve the word for our discussion of Java objects. In Oracle Database, I'm calling things by their primary type name and collectively or generically as *structures*, so tables will be called both *tables* and *structures*. I will also have a need to refer to collections of Oracle tables and views and indexes and things, which I will call collectively *structures*. We will define all these terms later, so don't be alarmed if they are new to you.

Chapter Review

It is my sincere hope that you will enjoy working through this book. I believe the best way to learn and use this material is to take your time with each chapter, executing the code as you come to it. You will be building a secure Oracle application environment with Java. Definitely test all the places I've provided you with test cases, scenarios, and test code. If you find other situations, test those too, and please send me your comments and results when I need to address an issue—my aim is to serve you.

CHAPTER 2

Oracle Database Security

As you might imagine, the subject of this chapter could fill a whole book, or even several, if I tried to cover the entire scope of Oracle security. Instead, I will cover the essentials, and also some particular aspects of Oracle security to which a programmer can relate. An example of essential Oracle security is using roles and granting privileges to those roles. An example of a programmer's extension of Oracle security is secure application roles.

The basic goals of this chapter are as follows:

- Create two users: a security administrator user and an application security user. These users will be granted privileges and delegated responsibilities for achieving application security.
- Use Oracle roles to control access and enhance application security, and learn about secure application roles identified by Oracle stored procedures.
- Distinguish between sensitive and non-sensitive data in the EMPLOYEES table.

Prepare to delve into many aspects of database administration and design. This chapter will get you started with Oracle security, and we will build on the concepts covered here throughout the remainder of the book. Some aspects, such as proxy connections, will only be really clear when presented in context. I want to be sure you really understand what is covered in the book, rather than just be exposed to it.

Finding a Test Oracle Database

We are going to hit the ground running. I hope you already have access to the SYS account on your database. If not, I hope you are good friends with the database administrator (DBA) for the server, and that you are recognized as a database security administrator. If so, you can ask your DBA friend to do the small portion of this work that must be done by SYS or a DBA.

If you'd like to learn the concepts in this chapter, but you don't want to use a managed server—and if you have sufficient computer power in your personal workstation—then you can download and install the Oracle Database 11g Enterprise Edition server and do the tasks that I describe in this chapter locally. I highly recommend that approach in any case, because you should definitely demonstrate and test the security measures that I describe in this chapter for your own peace of mind, before you place any of them in production.

Working from an Existing Oracle Database

If you are working from an existing database installation, you may have some issues to address. If you have a default Oracle Database 11.2 installation, then you need only consider the steps you have taken since installation to see if you have undone any built-in/default security. At minimum, you need to assure password complexity and secrecy. However, if you have a database that has been around for a while and has been upgraded from previous releases of Oracle, then you may have to spend some time and effort correcting the security issues.

I recommend you adopt a resource that Arup Nanda of the Oracle Technology Network has produced, called Project Lockdown. Project Lockdown is a series of checklists, tasks, and projects that will effectively enable and enforce Oracle database security. You can find this resource on the Oracle Technology Network web site at www.oracle.com/technetwork/articles/project-lockdown-133916.pdf.

Project Lockdown may take as much as several months to accomplish, depending on how lax your current security stance is. However, the first couple phases, which are the most critical, may be accomplished in a week.

Oracle Users and Schemas

Once Oracle Database 11g is up and running, you will want to consider users and user security, even before you think about the data, because users create data. Even application databases that don't belong to any particular person are associated with a user in an Oracle database. Each user on a local database has an associated schema, which is basically an organized storage allocation for Oracle structures (or objects) such as tables and indexes that belong to the user. See Table 2-1 for a list of users we will discuss.

Table 2-1. *Oracle Users We Will Use or Create*

Username	Description
`SYS`	Default Oracle system administrator.
`HR`	Human resources user/schema; installs with sample Oracle structures.
`secadm`	Our security administrator; we will create this account and give it privileges to implement all the security measures we need.
`appsec`	Our application security user; we will create this user and load code and other structures into her schema that we can use for app security.
`appusr`	Our first application user account; this account will only have the privileges needed to use the client application.
`osuser`	This is a pseudonym for any operating system username on your system; e.g., the username you use to log into Windows.
`appver`	User for application verification, covered in Chapter 10.

`osadmin`	This is another pseudonym for any operating system username on your system, in particular one who will be responsible for managing Oracle connection strings for applications.

SQL*Plus, SQL Developer, JDeveloper, or TOAD

I am going to assume that, if you are reading this, you are already doing some work as an Oracle developer. I'm going to further assume that you have some tool for submitting commands on an Oracle database, such as the prompt in SQL*Plus or the editor in TOAD, SQL Developer, or JDeveloper. SQL*Plus comes with Oracle, so if you don't have one of the other tools, you still have SQL*Plus. That is all you need to do the tasks outlined in this book.

■ **Note** Some of these tools are more sensitive than others to multi-line commands. They may require a slash character (/) on the line following a multi-line command.

Many of the SQL script files included in the source code for this book will need specific values edited, to be unique to your computer and corporate or home environment. Those values are listed at the top of the SQL files. After editing as appropriate, most of the SQL files can be executed as scripts; just be sure you are connected as the appropriate Oracle user.

Organization of the Next Few Sections

I want to make the process of working through this material as easy as possible for you. Toward that end, I have separated into sections the tasks and concerns of each of three users: `SYS` (the database administrator), the security administrator that we will be creating here, and the `HR` (human resource) schema owner.

For the most part, `SYS` as the database administrator could do all these tasks, but it is my intent to demonstrate security with delegation. We will have each of these users accomplish the tasks that are specific to their delegated responsibility. The commands to be executed by each of these users are consolidated into a single script file per user, and it is possible for us to focus on tasks and concerns of each of those users in turn, one at a time.

Each of these users must address a number of diverse topics, all related to security. These concerns are addressed in the order in which they are needed as we work toward the goals of this text. We will continue to build on all the topics I introduce here throughout this book.

You should have access to the code as we progress through this material. If you are reading this away from your computer, I hope to have provided enough of the code right in the text for you to completely understand the discussion. However, it is also my intention that you implement this code, and that this text will guide your understanding of what is occurring at each juncture.

When you work through the code, you will find that the best way to execute the commands in the various script files is to copy the file contents to a SQL command editor like TOAD or JDeveloper, and to execute one command at a time. That is preferred over executing the entire code as a script, because you will see each command as it gets executed. Remember, the goal is to understand secure programming, not just to implement security software.

Working as the SYS User

SYS has a schema (by the same name) and has lots of structures belonging to his account. You can connect (log in) to a database as SYS, but a unique requirement of that user is that you have to specify that you are using the SYSDBA privilege. (Note that in the text I will be using all uppercase for reserved words in Oracle commands and for default Oracle users, schemas, and structures.)

```
CONNECT sys AS sysdba;
```

At that point, you will have to enter the password associated with the SYS user.

■ **Note** You can find a script of the following commands in the file named *Chapter2/Sys.sql*.

System Privileges

SYSDBA is a unique system privilege; let's call it a "super" system privilege. It provides practically unlimited administrative ability. It can be granted to any user, but is best left granted only to SYS. As SYS you can check who has SYSDBA:

```
SELECT * FROM sys.v$pwfile users;
```

With this query, you request to SELECT (see) all the columns of data (*) from the V$PWFILE USERS view. The sys. prefix on that view name indicates the schema (user) that owns the view. You will see a couple of other super system privileges listed in this query: SYSOPER and SYSASM. These should also only be granted to SYS.

Besides the super system privileges, there are several dozen other system privileges that can be granted to provide such administrative abilities as creating other users and granting them privileges. We will be granting privileges as we explore Oracle security.

In addition to system privileges, there are schema object privileges. These allow, for example, another user to read your data. We will grant these privileges also. They have a much more tailored impact on Oracle application security.

Roles

Roles are collections of granted privileges that are much more granular and diverse and limited than the super system privileges. Instead of granting SYSDBA, the preferred method for giving users access to the various privileges is to grant the privileges to a role, then grant the role to the user. If you do this, you can always substitute a different user in a job, or replicate the privileges required for a job to another user by simply granting the existing role to the new user. (Privileges can be granted directly to users, but we will mostly avoid that.)

For example, I can create a role named appaccess and grant the ability to read the application tables to that role. Then I can grant that role to a user. When I have another user who needs access to the same application, I can permit that by granting the appaccess role to the new user as well. And when I delete the first user, the permissions required for other users to access the application do not disappear in the process.

I need to grant access to the application data as described previously, because the data tables will be in a schema belonging to the application, and each user will have her own schema. Users cannot read

data in other schemas unless that access is specifically granted to them or to a special user named PUBLIC. Anything granted to PUBLIC is granted to all users. See Table 2-2 for a list of Oracle roles we will discuss.

Table 2-2. *Oracle Roles We Will Use or Create*

Role Name	Description
DBA	This predefined role comes with Oracle upon installation and provides system administrative privileges for the database; we will not grant this role to any user.
CONNECT	This predefined role allows users to connect to Oracle; we are encouraged not to use predefined roles, so we won't.
create session role	Our role to allow users to connect to Oracle.
secadm role	Our security administrator role; we will grant the privileges required to perform security administration to this role, and we will grant this role to the SECADM user.
hrview role	Our secure application role for granting access to data in the human resources (HR) sample schema.
appsec role	A non-default role used by APPSEC when configuring application security.
appver role	Secure application role for running application verification, which I cover in Chapter 10.

The DBA Role

The DBA role is named for the job it enables: database administrator. It is similar to the SYSDBA super system privilege. Traditionally, the DBA role was granted to those users who needed to manage the database. The DBA role is almost as powerful as the SYSDBA privilege, but it can be modified to have some of its privileges removed.

In recent releases of Oracle, database administrators have been discouraged from using the DBA role. Rather, they are encouraged to create their own roles and to grant just those administrative privileges that are required. Therefore, we will not grant DBA role to any user.

We will be creating a security administrator role, secadm role. We will use this role for most of our administrative actions. It will have a variety of privileges, but only those required for the scope of this book. This approach adheres to the concept of "least privilege," which means providing only the privileges required for the task at hand.

Oracle Database Vault is a product that allows you to use DBA and other privileged roles while restricting their access. This is geared toward Department of Defense and national security users, where a database administrator does not necessarily have access to the data.

The Create-Session Role

We will also create a simple role, `create session role`, which will only have one privilege: `CREATE SESSION`. The `CREATE SESSION` privilege is required for a user to connect to an Oracle database. Traditionally, this has been accomplished through a predefined role (it exists when you install Oracle) named `CONNECT`. In current releases of Oracle database, `CONNECT` only has the one privilege. But in this case, as with the `DBA` role, Oracle recommends that administrators create their own roles and not rely on the predefined roles (like `CONNECT`). Do this as `SYS`:

```
CREATE ROLE create session role NOT IDENTIFIED;
GRANT CREATE SESSION TO create session role;
```

We will be granting this role to all users in the database. It would take the exact amount of effort to grant the `CREATE SESSION` privilege to each user as it takes to grant `create session role`. However, there is a benefit to roles as a centralization of privileges. For example, if we wanted to quickly give another privilege to all users, we could grant it to the `create session role`; although, I don't recommend that. More likely there might be a time when you need to keep the database running, but want to keep all users from connecting to it. That can be done by a single command, revoking `CREATE SESSION` from `create session role`.

NOT IDENTIFIED

I hope the `CREATE ROLE` syntax is fairly obvious. The only odd feature is the `NOT IDENTIFIED` keyword. That merely indicates that when a user is acquiring this role, we do not have any password or encoded process to verify his access to it. A role of this type has to be granted to a user (or to another role) by an administrative command. Generally, roles that are designated as `NOT IDENTIFIED` are also default roles, which a user acquires automatically when he connects to the Oracle database. This is the most usual configuration of a role.

Using Roles

Most roles get created as default roles. When a user connects to an Oracle database, he acquires all his default roles, and all the privileges associated with those roles. Any regular `NOT IDENTIFIED` role can be set to be default or non-default.

■ **Note** In Oracle Database 11g, default roles that are identified by a password are no longer acquired when a user first connects—the user must enter the password. More on that later.

At any time, the Oracle user can acquire a role that has been granted by executing a SET ROLE command. This is true for both default and non-default roles.

When we SET ROLE to a new role, it becomes the only role in use, even if other roles were in use before. Fortunately, the session has already been created (we are already connected when the new role is set), so we can stand to lose the `create session role` (which will happen). I can imagine a scenario

wherein we would require every role to be granted `CREATE SESSION` privilege so that when you `SET ROLE`, you still have `CREATE SESSION` in the new role.

There is a way to add to existing roles, keeping those that exist. By issuing the command:

```
SET ROLE ALL;
```

This command cannot be used to set a role that is `IDENTIFIED` by a password or a procedure. However, after you acquire a password-protected role or secure application role (we will discuss those in a minute), you can `SET ROLE ALL` and restore your default roles as well.

I should mention that roles can be granted to roles, which we will do. Setting your role replaces whatever you had before, but will include not only the role you set, but all the roles granted to that role, and so on, recursively. There is a constraint to these cascading grants so that no endless loops can be formed.

Password-Protected Roles

Let me preface this discussion with a note regarding Oracle 11g: starting with this version of Oracle, in order to set password-protected roles from an active connection, you have to supply the password, even if it's a default role. This may catch you off guard. Previously, if a role was protected by a password but it was a default role, the user got that role by default when she connected.

I agree with this change to Oracle; and for the same reasons, we will not have any password-protected roles in this book. The only reason to have a password-protected role is in a case where the role is granted to a user, but you don't want the user to use the role. You protect the role by setting a password, and don't let the user know. But, we have to ask, "Why did we grant him the role in the first place?"

A better approach is to only grant the role to the users who need it. Perhaps this is a bit muddied by the possibility for many people to connect as a specific Oracle user, where some of those people need access to the role and some don't. In that case, I recommend you either make the people who need the role enter the role password, or, better yet, revisit your users and roles and sort them out.

One possibility is to have a secure application role, validated by a procedure. That role can be granted based on user id or some group membership. Then no role password is needed. You just need to maintain a list of acceptable users or a group membership list.

I have observed a corporate policy of securing every application role with a password. The idea is that people sitting at a general SQL client, like SQL*Plus, will not have access to the role, because they don't know the password. But from the application where we want them to have the role, a procedure is called to set the role by looking up the role password in a database table. It may be that this configuration provides a small impediment to casual Oracle users getting the application roles; however, the bare facts to observe here are that the person who has been granted the role will have access to the role either by that same procedure that the application calls, or by the same code that's in that procedure. From SQL*Plus, the user may simply call the procedure, get the password, and set the role.

Perhaps the role could be protected from bald access by assuring that only a certain client application is being used; again, this is a job for a secure application role. But the real problem here is that a role has been granted to an Oracle user, and the administrator does not want the user to access the role. Password-protected roles provide only the illusion of enhanced security, unless you make specific users memorize and manually enter the password to set the role. That is the thinking behind the new policy in Oracle 11g.

Security Administrator User

Let's go ahead and start defining our security administrator. The security administrator is going to be a separate non-person user; that is, an account and password that can be delegated to various people who rotate into and out of the job responsibility. The security administrator will be doing tasks that would typically be done by a DBA or even by SYS, but we are going to limit the privileges we give to the security administrator to just those aspects related to application security.

First, as SYS we will create the user `secadm` and grant him the `create session role`, in one step. Substitute a real password for "password" in this command:

```
GRANT create session role TO secadm IDENTIFIED BY password;
```

■ **Caution** Be sure to give a very complex password to this user; it will be powerful enough to be dangerous in the wrong hands. This warning goes for the SYS and SYSTEM accounts as well.

Security Administrator Role

Next, we will create the security administrator role, `secadm role`. It will be the role that is granted all the privileges needed to do security administration. We can then grant this role to any user, but we will limit it to just one user, `secadm`.

At the outset, I want to set some requirements regarding how and when this role can be used, so I will not use the `NOT IDENTIFIED` keywords that we saw with `create session role`. Rather, I will have it be identified (verified) by a procedure. You can see the name of the verification procedure, `sys.p check secadm access` in this command:

```
CREATE ROLE secadm role IDENTIFIED USING sys.p check secadm access;
```

A role that is identified by a procedure is called a secure application role.

Security Administration Role Verification

As I mentioned, all the privileges that the security administrator needs will be granted to `secadm role`, so we are going out of our way to protect it. The keywords `IDENTIFIED USING sys.p check secadm access` indicate that when a user attempts to acquire `secadm role`, he will have to get it from the procedure named `p check secadm access`, which exists in the SYS schema.

A stored procedure (procedure) is a named block of Procedural Language/Structured Query Language (PL/SQL) code that is stored and run (executed) on the Oracle database. Generally, a procedure takes parameters and does work. It can also return information through its parameters. In Oracle there are also stored functions (functions), which are very similar to procedures except that they usually take values; do research or calculations; and then return a single value as the result. We will be using both procedures and functions.

The specific procedure used for verifying `secadm role`, `p check secadm access` does not take any parameters (arguments or values passed to the procedure for evaluation), and does not return any results. Most procedures do take parameters, but they don't have to.

In the Listing 2-1, we are creating the procedure to be used for acquiring the security administrator role. Note that the simple goal I have for this procedure is to require that the security administrator be

running on the same computer as the Oracle database server (the IP address 127.0.0.1 is also known as the localhost or the loopback address). This requirement may not be appropriate for your system; if not, you can still create the procedure, but comment out the three lines that start with IF, THEN, and END IF by placing two dashes (minus signs) in front of it. You can execute this command as SYS, and the procedure will be created.

Listing 2-1. p check secadm access Procedure for Secure App Role

```
CREATE OR REPLACE PROCEDURE sys.p check secadm access
AUTHID CURRENT USER
AS
BEGIN
    -- This is a comment
    IF( SYS CONTEXT( 'USERENV', 'IP ADDRESS' ) = '127.0.0.1' )
    THEN
        EXECUTE IMMEDIATE 'SET ROLE secadm role';
    END IF;
END;
/
```

We will want to come back and visit (replace) this procedure later, adding whatever additional security constraints we deem appropriate. For now, our Security Administrator must be connected directly to the Oracle Database (running SQL*Plus on the server.)

As a programmer, you will already understand the IF/THEN/END IF syntax and realize there are other PL/SQL grammar requirements like the BEGIN/END and the semicolons.

Look at the code between the BEGIN flag and the last END flag. Between there it says, in English, "if the user's environment has an *Internet Protocol* (IP) address of the local host, then immediately set his role to secadm role." We are using the SYS CONTEXT *context* to get the IP address from the *user environment*, and we are determining if the address is equal to 127.0.0.1 (localhost). If that test is true, then we immediately set the role for the current *session* to secadm role.

INVOKER'S RIGHTS VERSUS DEFINER'S RIGHTS

In the procedure we defined above, we say AUTHID CURRENT USER so that this procedure is executed as the current user, not as SYS (the one who defined the procedure.) So when executed, this procedure will use what is called "Invoker's Rights" (CURRENT USER) as opposed to "Definer's Rights" (the default or owner, which in this case is SYS). If we didn't do this, the environment and identity would appear to be SYS when we execute the procedure, but we want to be able to determine the acceptability of (authorize) the specific, current user. Another reason we need to execute with invoker's rights is that otherwise we are not allowed to set a role from a procedure–we can only set the role for the *authenticated* (current) user.

Security Administrator Role Acquisition

We are not nearly done defining our Security Administrator yet. We need to permit the Security Administrator to execute the procedure we created. We do that by issuing the following command:

```
GRANT EXECUTE ON sys.p check secadm access TO secadm;
```

Notice that we don't need to grant `secadm role` to `secadm`; rather, if the user meets all the requirements of the procedure (`p check secadm access`), then that `IDENTIFIED USING` procedure will set his role to `secadm role`. There is an analogous restriction here: either you grant a role or you grant the ability to execute a procedure, which will set the role. In both cases you are limiting access to the role by requiring a specific grant. However, by using the procedure to set the role, you can place further restrictions on access.

Notice that in this case, we are granting a privilege (to execute the procedure) to a user directly, instead of, as we recommended previously, granting permissions to a role and granting the role to a user. Every rule has exceptions! The reason for this specific exception is that we are protecting a role with a procedure. We do not need to further protect the procedure with a role.

The Security Administrator, `secadm` user can acquire `secadm role` by executing the procedure as shown below. In fact, every time secadm connects to Oracle Database, he will need to call that procedure to get the `secadm role` role.

```
EXEC sys.p check secadm access;
```

System Privileges Granted to the Security Administrator Role

The bulk of privileges that we want our Security Administrator to have will be granted by `SYS`:

```
GRANT
    CREATE USER
    ,ALTER USER
    ,CREATE ROLE
    ,GRANT ANY OBJECT PRIVILEGE
    ,GRANT ANY PRIVILEGE
    ,GRANT ANY ROLE
    ,CREATE ANY PROCEDURE
    ,CREATE ANY TRIGGER
    ,CREATE ANY CONTEXT
    ,CREATE PROFILE
    ,AUDIT SYSTEM
    ,AUDIT ANY
TO secadm role;
```

Most of these system privileges that we are granting to `secadm role` permit us to do some of what we have been doing already as `SYS`. We will allow our security administrator to create and modify users and to create roles and grant them to users. `secadm` will grant privileges for roles to work with Oracle objects (structures) in other schemas. He will grant certain system privileges to roles. He will also create procedures and triggers, which are like procedures, but execute when certain events occur (we'll discuss this more in a bit.) We will also create an application context when we get to Chapter 10. And the security administrator will create profiles, which we will see when we get to Chapter 8. For now, we rely on the default profile.

We will also be setting up some auditing as security administrator. We will audit a variety of system events, and we are going to audit access to tables and other structures in the `HR` schema—hence the `AUDIT ANY` privilege. Any time you see `ANY` in an object privilege grant, you can read that as "in any schema." Generally, a user already has those privileges in his own schema.

These are not all the system and schema object privileges that our security administrator will need to do his job, but they will get us started. We will come back as `SYS` and give the security administrator a bit more privileges later.

The Audit Trail

Finally, as SYS, we are going to set up some initial auditing on the auditing trail itself. This will deter a rogue database administrator from doing something wrong and then erasing their tracks by deleting their audit records:

```
AUDIT SELECT, UPDATE, DELETE
    ON sys.aud$
    BY ACCESS;
```

When we designate BY ACCESS for auditing, we are saying that we want detailed information. The other (possibly default) option is BY SESSION. This gives less detail, but still audits each occurrence, rather than only providing a single audit record per session, as in earlier releases of Oracle.

The Data Dictionary

We want our security administrator to be able to view all data in the Data Dictionary, which is a collection of views in the SYS schema that list structures and system data in Oracle. (A *view* is a defined way to look at a table of data.) For example, we may want to list details about all the database users:

```
SELECT * FROM sys.dba users;
```

There are many columns in the DBA USERS view that are not available in the PUBLIC data dictionary views: ALL USERS (fewer details) and USER USERS (a bit more detail, but only for the current user).

Much of the data dictionary has been granted to PUBLIC by default, and can be selected by every user. For the most part, this is needed. But we will deal with this a bit more stringently in Chapter 11, *Enhancing Our Security*. However, selecting some parts of the Data Dictionary requires the SELECT CATALOG ROLE. Grant that role to the secadm role:

```
GRANT select catalog role TO secadm role;
```

Note that this is a role granted to a role. From now on, when we set the role to secadm role, we will also have the SELECT CATALOG ROLE.

Working as the Security Administrator

Now that our security administrator has been defined with the privileges he needs to do his job, we are going to put him to work. Go ahead and connect:

```
CONNECT secadm;
```

■ **Note** You can find a script of the following commands in the file named *Chapter2/SecAdm.sql*.

As you will recall from our creation of the security administrator role, secadm role, we required that it be validated by a procedure. We have only permitted one account, secadm to execute the procedure. Execute it now to acquire secadm role:

```
EXEC sys.p check secadm access;
```

Note that when secadm connects to Oracle, he does not automatically acquire secadm role. Because it is a secure application (verified) role, and not directly granted to secadm, it cannot be a default role. Every time the secadm user connects, he will have to execute the procedure to gain his Security Administrator role and privileges.

This is in contrast to roles that are granted directly to a user, which are default roles initially. That status of default role can be unset.

Acquire secadm_role from a SQL*Plus Local Connection

There are always gotcha's, and here's one that will probably get you a few times if you use SQL*Plus as your primary client: when you are sitting at a command prompt on the Oracle database, you can connect locally to the default instance. Do that by executing SQL*Plus without any arguments, like this:

```
sqlplus
```

You can connect then as secadm user by entering the username and the password. If you then attempt to execute the procedure, sys.p check secadm access that sets secadm role, it will not succeed. Why does it not succeed? Our address should be that of localhost, which should be okay. Well, when connecting locally, SQL*Plus doesn't use the network at all—it just talks directly to the database. You can see the lack of IP address information by executing this command:

```
SELECT SYS CONTEXT( 'USERENV', 'IP ADDRESS' ) FROM DUAL;
```

This has some implications for security. When connected locally like this, the commands you enter do not traverse the network adapter and have no possibility of leaking out on the network to snooping devices.

So how, you might ask, are we supposed to connect as secadm and run sys.p check secadm access from SQL*Plus on the Oracle Database? There is a way, and it only requires that you act like you're not connecting locally by adding the arguments for user and instance name (orcl in this example) on the command line. Actually orcl in this context is a TNS alias with the same name as the instance. We will discuss TNS aliases in Chapter 11.

```
sqlplus secadm@orcl
```

At that point, you have an IP address in the session context, and you can successfully set role through the procedure:

```
SELECT SYS CONTEXT( 'USERENV', 'IP ADDRESS' ) FROM DUAL;
EXEC sys.p check secadm access;
```

Toggle Between Roles

You can see the effects of running SET ROLE by observing your current session roles as you toggle back and forth between create session role and secadm role. As secadm user, do this and observe the list of roles when you do each SELECT query:

```
SELECT * FROM sys.session roles;
SET ROLE create session role;
SELECT * FROM sys.session roles;
EXECUTE sys.p check secadm access;
SELECT * FROM sys.session roles;
```

You will see three roles when you execute the procedure and then select from SESSION ROLES. The procedure itself sets your role to secadm role. To that role, we granted SELECT CATALOG ROLE. And the

SELECT_CATALOG_ROLE has been granted HS_ADMIN_SELECT_ROLE. All of those will be listed as current roles of the session. If you have additional packages or different versions installed on your Oracle Database, you may see additional roles related to those packages.

Create an Application Security User

We will need a couple more users to develop and demonstrate our security. The first user is our Application Security user, appsec. She will own all the structures that make our application security work.

Please take note of the differentiation I am making between Oracle security, which we've delegated to security administrator, and application security. The security administrator has been granted a number of system privileges so that he can create users and do other tasks in Oracle. The application security user, on the other hand, will own a number of procedures and structures that are used for application security processes. The application security user will not connect as part of any particular application; rather, she will grant access to her logic and data to application users.

```
GRANT create session role TO appsec IDENTIFIED BY password;
```

■ **Caution** Be sure to give a very complex password to this user; she does not have significant privileges in the database, but she owns some logic and data that we do not want to expose to corruption.

Application Security Role

Our application security user needs to create procedures, functions, Java stored procedures, tables, and views. When creating those items, appsec requires the CREATE PROCEDURE, CREATE TABLE, and CREATE VIEW system privileges. We will grant those privileges to a role named appsec_role, and grant that role to the appsec user:

```
CREATE ROLE appsec_role NOT IDENTIFIED;
GRANT CREATE PROCEDURE TO appsec_role;
GRANT CREATE TABLE TO appsec_role;
GRANT CREATE VIEW TO appsec_role;
GRANT appsec_role TO appsec;
```

Non-Default Role

Our application Security user only requires these privileges (for example, CREATE PROCEDURE) on occasion. Our preference would be for her to set her role to appsec_role as needed, but currently it is a default role. In order to change the behavior for roles created as NOT IDENTIFIED from being default roles to be non-default, we need to issue the ALTER USER command. In the following command, we ask Oracle to make all the roles granted to appsec user to be DEFAULT except this one we want to exclude:

```
ALTER USER appsec DEFAULT ROLE ALL EXCEPT appsec_role;
```

Hereafter, when appsec connects in order to create new structures, she will have to issue one of the SET ROLE commands. (Do not execute these commands as secadm user—at this point, they are for explanation only.)

```
SET ROLE appsec role;
SET ROLE ALL;
```

The second command listed above will set the current session for appsec to enable ALL the roles that have been granted to her, both default and non-default.

Create an Application User

As an example user for connecting to Oracle to run an application, we will create an application user, appusr. For our purposes, appusr is not tied to a single person, but is what Oracle refers to as a "one big application user" model user. In this model, numerous people will use an application, and the application will connect all of them to Oracle as our one big application user. They do not need individual person accounts and passwords for this access.

```
GRANT create session role TO appusr IDENTIFIED BY password;
```

■ **Caution** Be sure to give a very complex password to this user—they can potentially select data and do updates to application data. We only want that activity to take place from within the application. I hope you are getting used to this warning about passwords and can even anticipate it.

INDIVIDUAL PERSON ACCOUNTS

We will have an opportunity to create Oracle user accounts that are specific to individual people; however, the administrative requirement for maintaining those users (and granting, validating, and revoking the required privileges) is a huge responsibility. I think there are better, less tedious and more secure ways to recognize and authorize individual people in the database, and we will look at that in Chapter 8.

Create the HR View Role

We are going to begin securing access to the data in the Human Resources (HR) Sample Schema. We will create a role named hrview role. Through that role we will grant access to the data needed by a variety of applications that we plan to build. At the outset, we only want folks who are on our internal network to access this data, and only during our normal office hours of 7 AM to 7 PM.

To accomplish these constraints, we will create the role and require that it be verified by a procedure. Oracle calls this a secure application role because that is its function – it is a role that gives access to application data, but it is secured by some encoded constraints. This is the same thing we did to secure secadm role.

```
CREATE ROLE hrview role IDENTIFIED USING appsec.p check hrview access;
```

Verify HR View Role by a Procedure

Just as we did previously for the `secadm role`, we will create a procedure to protect access to the `hrview role`. At the end of the procedure, if the `CURRENT USER` meets the requirements encoded in this procedure, we will `SET ROLE` to the `hrview role`.

Create the procedure by executing the code in Listing 2-2 as `secadm`. Here, we are creating a procedure in another schema. Notice the schema name, appsec. prepended on the procedure name. To do this requires `secadm` to have the `CREATE ANY PROCEDURE` system privilege.

Listing 2-2. *`p check hrview access` Procedure for Secure App Role*

```
CREATE OR REPLACE PROCEDURE appsec.p check hrview access
AUTHID CURRENT USER
AS
BEGIN
    IF( ( SYS CONTEXT( 'USERENV', 'IP ADDRESS' ) LIKE '192.168.%' OR
        SYS CONTEXT( 'USERENV', 'IP ADDRESS' ) = '127.0.0.1' )
    AND
        TO CHAR( SYSDATE, 'HH24' ) BETWEEN 7 AND 18
    )
    THEN
        EXECUTE IMMEDIATE 'SET ROLE hrview role';
    END IF;
END;
/
```

Notice that, here again, we have the statement `AUTHID CURRENT USER`. We are going to execute with invoker's rights (IR) instead of the default definer's rights (DR). In this procedure we have encoded two tests in the `IF` statement. And for those sessions that pass both tests, we set the role to `hrview role`.

Test for Subnet

The first test in `p check hrview access`, shown in Listing 2-2, gets the IP Address of the client from the user environment and tests that it begins with the string "192.168." The `LIKE` syntax indicates that the address should consist of the specified characters followed by zero or more characters (the "%" symbol is a wildcard). The test assures that the client exists on our internal corporate subnet. ("192.168" is a non-routed subnet, and can be used behind firewalls or on test subnets. It is typically the subnet used for a home network that attaches to the Internet through a DSL or cable modem.) Your corporate subnet is most likely different from this subnet, so modify the code for the procedure before you create it. At the command prompt, you can find the IP address and other networking information of your Windows workstation by issuing a command:

```
C:\>ipconfig /all
```

Probably the first two octets (sets of three or fewer digits separated by dots, or sets of eight bits in binary) of your workstation's IP address are representative of your corporate subnet. Contact your networking support personnel to determine exactly what to use.

This procedure also allows you to connect to Oracle from the Oracle database server itself. If the IP address is 127.0.0.1, the localhost address (that is, the Oracle database server itself), then the procedure also succeeds.

Test for Normal Business Hours

The second test in `p check hrview access`, shown in Listing 2-2, contains the statement `TO CHAR (SYSDATE, 'HH24') BETWEEN 7 AND 18`. Let's take a minute to break this down. `SYSDATE` is the name for the current time and date on the Oracle database. You can `SELECT` (request) `SYSDATE` from the server, and you can set date values to be equal to `SYSDATE` when you update data in Oracle. In this case, we are requesting `SYSDATE` and using the `TO CHAR` function to format just the hour (`HH24`) portion the date into a character string using the 24-hour format. In this format, when it is 7AM, the `TO CHAR` function will return 7; and when it is 7PM, 19. So we check that the hour is `BETWEEN` 7 and 18 (at 6:59PM, this test will still be true; but at 7PM, false.) We have determined that our normal business hours are between 7AM and 7PM, so these tests are valid.

■ **Caution** There is no code here to disconnect the user from Oracle when 7PM arrives. This code only prevents further connections from being established after 7PM.

Permit Application User to Acquire HR View Role

Every user that needs to get access to the `HR` data will have to execute the `p check hrview access` procedure to have the `hrview role` set. It is only the `hrview role` that has access to the data, and the only way to set that role is by executing the procedure. We will grant the `EXECUTE` privilege for this procedure to our application user, `appusr`. As `secadm`, execute this:

```
GRANT EXECUTE ON appsec.p check hrview access TO appusr;
```

We will have to grant `EXECUTE` on this procedure to all users who need access to the data. Alternatively, we could grant the `EXECUTE` privilege to `PUBLIC`, but we would only want to do that if every user of the database required access to the `HR` data. Do not grant privileges to `PUBLIC` without considering the implications.

Audit Changes to Security Administrator Procedures

We will close out this part of the chapter by establishing some additional auditing. Because we are working as `SECADM`, the default structures for which we define auditing are in the `SECADM` schema. We will audit any change to procedures in our schema, because they are security related. We need to assure that any changes have been vetted.

```
AUDIT ALTER ANY PROCEDURE BY ACCESS;
```

This is actually one of the privileges that is audited by default when the `secconf.sql` script is run as part of the database creation.

Audit Failed Attempts to Access HR Data

Our first audit on access to the `HR` data is really an audit on attempted execution of the `p check hrview access` procedure, which sets the `hrview role`. We don't want to know when this procedure succeeds, but we would like to know when invalid access is attempted, so we use the keywords `WHENEVER NOT SUCCESSFUL`.

```
AUDIT EXECUTE ON appsec.p check hrview access
    BY ACCESS
    WHENEVER NOT SUCCESSFUL;
```

Working as the HR Schema User

For several aspects of the endeavors put forth in this book, we are going to use the HR sample schema that can be installed with Oracle 11g. If you haven't got that schema installed, you will have had difficulties in the preceding step that configured auditing on HR schema structures. Please browse to the Oracle web site for instructions on how to install the sample schemas after the fact.

Now, at the installation of Oracle, you also had the opportunity to configure a password for the HR user. If the sample schemas exist but the HR user is not configured (or simply to reconfigure the user), you can issue the ALTER USER command (as SYS or secadm):

```
ALTER USER hr ACCOUNT UNLOCK IDENTIFIED BY password;
```

■ **Caution** Be sure to give a very complex password to this user; HR owns the data that we are going to attempt to protect.

Connect to Oracle as the HR user:

```
CONNECT hr;
```

HR has no system privileges outside his own schema, but has been granted the privileges required to connect to the database and create a number of structures in his own schema: tables, views, indexes, and the like.

■ **Note** You can find a script of the following commands in the file named *Chapter2/HR.sql*.

Sensitive Data in the HR Sample Schema

In the HR sample schema, which can be installed with Oracle 11g, there is a table named EMPLOYEES. That table has a couple columns that we should call "sensitive": SALARY and COMMISSION_PCT. We are going to make it our goal to grant access to that table without compromising the sensitive data.

View the contents of the EMPLOYEES table with this command, as HR:

```
SELECT * FROM hr.employees;
```

Let's pretend that it is our company's goal to publish an online personnel directory (phone book), and that this table is the primary source data. We could export just the non-sensitive portions of the data for use in the phone book, but it would be much better to grant secure access directly to the primary data in this table, rather than replicating the data (assuming our database server will support the additional load).

Public View of Employees

The most basic method for granting limited access to a data table is to create a view. A view is like a filter that we put on the data table that can show only certain data, reorganize the data, format the data and provide data from multiple tables or other views. We will create a view named `v employees public` that only includes the non-sensitive columns of data:

```
CREATE OR REPLACE VIEW hr.v employees public
AS SELECT
    employee id,
    first name,
    last name,
    email,
    phone number,
    hire date,
    job id,
    manager id,
    department id
FROM hr.employees;
```

Test the new view to assure it returns just the data we've requested. We should notice that the `SALARY` and `COMMISSION PCT` columns are missing:

```
SELECT * FROM hr.v employees public;
```

Because we have omitted the sensitive columns (`SALARY` and `COMMISSION PCT`) from this view, we can grant access to this view to the entire company without exposing sensitive data. We could grant it to `PUBLIC`; however, we still feel a bit protective of our data, and we want to be sure access is controlled, so we grant access to the view to our secure application role, `hrview role` rather than to `PUBLIC`. `HR` user has the privilege to grant access to structures in his own schema:

```
GRANT SELECT ON hr.v employees public TO hrview role;
```

With this grant we are permitting `hrview role` to `SELECT` data from the view. Other common privileges one might grant to a view are `INSERT`, `UPDATE` and `DELETE`. We are not granting any of those privileges at this time. The `hrview role` can see but not modify the non-sensitive data through the `v employees public` view.

Sensitive View of EMPLOYEES

Now that we have our non-sensitive view configured, it should be obvious how to create a sensitive view of all columns of the `EMPLOYEES` table. As `HR`, we create a view:

```
CREATE OR REPLACE VIEW hr.v employees sensitive
    AS SELECT *
    FROM hr.employees;
```

The asterisk (*) represents all columns, and this view selects all the data from all the columns. Test that this view returns all the data from the `EMPLOYEES` table:

```
SELECT * FROM hr.v employees sensitive;
```

This view might be useful at some point in the application—for instance, when we want to let certain financial analysts count how many people earn a certain salary in preparation for annual raises. We might also want to use this view for a financial application that we want to grant `UPDATE` access for

giving out those raises. It would also be useful to have this view to allow Human Resources to INSERT and DELETE folks from the database as they are hired and terminated.

However, given the sensitivity of the data (how much money you and I or our managers earn), we are not going to grant access to this view at all. Later, we are going to provide access to this data through much more heavily guarded and encrypted channels.

Our first audit statement on HR data was given previously in the section "Audit Failed Attempts to Access HR Data." Our second audit statement for HR data follows. With it, we intend to audit any direct access to the sensitive view of the EMPLOYEES table. Later, we will look at auditing the selection of just those fields that we consider to be sensitive, no matter what view or table they are selected from.

```
AUDIT SELECT ON hr.v employees sensitive BY ACCESS;
```

Test Application User Access

To test our security, we need to try things that we believe should not work (because of our security measures) as well as the things that we permitted to succeed. To do this, you will need to connect to Oracle as appusr user:

```
CONNECT appusr;
```

■ **Note** You can find a script of the following commands in the file named *Chapter2/AppUsr.sql*.

The next three commands, we believe, will fail because the appusr user has not been directly granted access to anything in the HR schema. He only has his default roles, which also do not have access. We should see an error message that those tables or views don't exist.

```
SELECT * FROM hr.employees;
SELECT * FROM hr.v employees sensitive;
SELECT * FROM hr.v employees public;
```

Now we will execute the procedure that will check our validity, and then set role to hrview role. This should succeed if we are on our corporate subnet (or on the Oracle Database server, itself) and it is between 7AM and 7PM:

```
EXEC appsec.p check hrview access;
```

Then we will test our access to the EMPLOYEES structures in the HR schema. We do not expect the first two commands to work; again we should see an error message that those structures don't exist. The hrview role does not give access to the sensitive data:

```
SELECT * FROM hr.employees;
SELECT * FROM hr.v employees sensitive;
```

On this last command, we should see the data from the EMPLOYEES table; however, we note that there are two columns missing from the data: SALARY and COMMISSION PCT.

```
SELECT * FROM hr.v employees public;
```