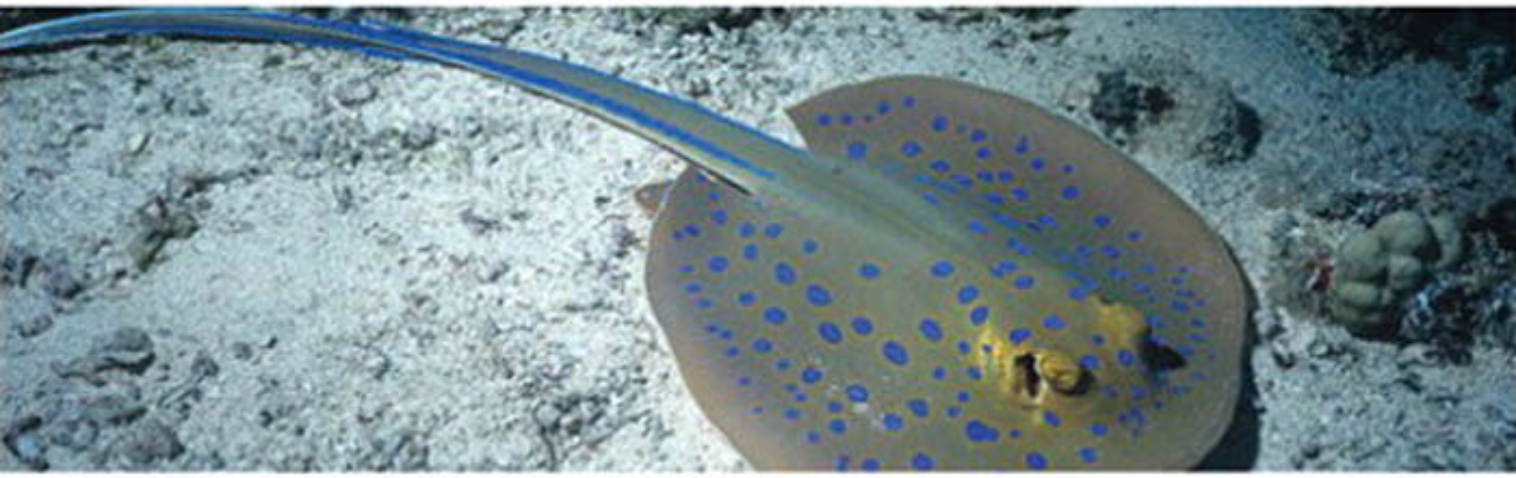
"What Kevvie Fowler has done here is truly amazing: He has defined, established, and documented SQL Server forensic methods and techniques, exposing readers to an entirely new area of forensics along the way. This fantastic book is a much needed and incredible contribution to the incident response and forensic communities."

—**Curtis W. Rose,** founder of Curtis W. Rose and Associates and coauthor of *Real Digital Forensics*

SQL SERVER FORENSIC ANALYSIS



KEVVIE FOWLER

Library of Congress Cataloging-in-Publication Data

Fowler, Kevvie.
SQL server forensic analysis / Kevvie Fowler.
p. cm.
Includes index.
ISBN 978-0-321-54436-0 (pbk. : alk. paper)
1. SQL server. 2. Computer crimes—Investigation. I. Title.

QA76.9.C55F685 2008
364.16'8—dc22

2008042778

ISBN-13: 978-0-321-54436-0
ISBN-10: 0-321-54436-6
Text printed in the United States on recycled paper at Courier in Stoughton, Massachusetts.
First printing, December 2008

Preface

During a forensic investigation, a digital investigator tracks an intruder's actions on a system until "it" happens—the investigator identifies that the intruder has, indeed, accessed the database. The database server stores sensitive financial information, but it is configured with default database logging and no third-party logging solution is in place. Even though the investigator identified that the database was accessed, he is now left to wonder: What actions did the intruder perform within the database server? Was credit card data accessed? Was anything modified?

This scenario is an all-too-familiar one, which usually leaves investigators staring into a black hole, desperately needing a way to determine which actions an intruder performed within a database server. Given that large data security breaches are occurring at an alarming rate today, investigators who are unable to properly qualify and assess the scope of a data security breach can be forced to report that all database data may have been exposed during an incident. This can, in turn, result in organizations disclosing that confidential database data was exposed when, in fact, the incident may not have involved this data.

This book is intended to help you avoid the preceding scenario. It provides the first in-depth view into the collection and preservation of database artifacts, explaining how they can be analyzed to confirm a database intrusion and how you can retrace the footsteps—and actions—of an intruder within the database server. The SQL Server forensic techniques profiled in this book can be used both to identify unauthorized data access and modifications and to restore the pre-incident database state so as to recover from the database intrusion.

WHY DO WE NEED THIS BOOK, AND WHY NOW?

Within the past few years, our reliance on database technology has increased exponentially. Databases have become an increasingly essential component of some of the world's largest corporations. Indeed, in today's business world, almost all applications use a database to manage data.

As our reliance on databases has increased, so, too, have the number of attacks targeting the data those databases store and process. According to Gartner Group, 75% of cyber-attacks are application based; these assaults often involve the theft of personal or financial data stored within a database.

With digital attacks targeting databases on the rise, large data security breaches are occurring at an alarming rate. In response, regulations at several levels have been put in place to hold those who manage and store personal information accountable if and when the confidentiality of this information is compromised. More specifically, many regulations demand that any organization that collects, uses, or stores its clients' information must notify affected clients in the event that their personal information is disclosed. Because of the need to comply with this requirement, it is becoming increasingly important for digital investigators not only to be able to confirm the occurrence of unauthorized database access but also to specifically determine what, if any, sensitive information was accessed.

WHO WILL BENEFIT FROM READING THIS BOOK

This book will appeal to a wide audience—digital forensic practitioners, information security analysts, information security managers, information security auditors, database administrators, systems administrators, and law enforcement officials interested in digital forensics, security, or relational databases. Readers will benefit from reading this book if they are interested in an in-depth view of any of the following topics:

- How SQL Server forensics can be used to pick up where traditional forensic investigations end to confirm a database intrusion and retrace an intruder's actions within the database server
- How real-world SQL Server forensic techniques can be applied on default SQL Server installations without dependency on third-party logging applications
- How to identify, extract, and analyze database evidence from published and unpublished areas of SQL Server
- How to build a SQL Server incident response toolkit

- How to detect and circumvent the threat posed by SQL Server rootkits during a database investigation
- How to identify and recover previously deleted database data using native SQL Server commands

Readers of this book should have a basic understanding of digital forensics and relational databases. Although the first two chapters provide an introduction to databases and SQL Server fundamentals, to get maximum benefit from this book, readers who feel they need more details on databases in general are directed to *An Introduction to Database Systems* by C. J. Date (Addison-Wesley, 2004) and *Inside Microsoft SQL Server™ 2005: The Storage Engine* by Kalen Delaney (Microsoft Press, 2006). Both books are excellent references that provide a broader and deeper look at relational databases and SQL Server fundamentals than is possible to cover within this book.

INFORMATION SOURCES

The content of this book is based on SQL Server documentation, industry-recognized forensically sound principles, and independent research. The primary goal of this book is to provide "real-world" database forensic techniques that can be used to investigate intrusions on SQL Server 2000, 2005, and 2008 installations with default configurations.

That being said, SQL Server is a robust relational database management system (RDBMS), and it boasts a multitude of options and configuration settings. The information within this book has been tested and will support default SQL Server configurations as well as most nonstandard configurations. It is possible, however, that some customized SQL Server configurations may produce slightly different results.

SQL Server forensics is an emerging and specialized area in the field of computer forensic science. As such, it will most certainly evolve over time. As an example, look how far we've come from the old "pull the plug" and then obtain a forensic image philosophy—even some of the most fundamental aspects of information systems management evolve and change over time. As the practice of SQL Server forensics evolves as a result of security research and feedback from those in the field, updates will be posted to the `http://www.applicationforensics.com` Web site.

11 SQL Server Forensic Investigation Scenario

Scenario Overview

In previous chapters, we've taken an end-to-end walk-through of SQL Server forensics. We began by defining what SQL Server Forensics is, examining how it can be used to augment traditional forensic investigations, and exploring each stage of the database forensics methodology. During this review, we've looked at a variety of SQL Server artifacts, the data within them, and the ways this data can be leveraged to benefit an investigation.

The goal of this chapter is to bring the technical content we've covered to life in an investigation scenario that you can walk through. Performing this walk-through will allow you to appreciate the logical progression of events during an investigation and gain a deeper understanding of how findings within artifacts can be confirmed and further analyzed. This chapter also contains some advanced activity reconstruction analysis methods that should serve as an extension to the content in Chapters 8 and 9.

This chapter will not regurgitate the information covered in previous chapters, nor will it cover all SQL Server artifacts. Instead, it aims to provide a real-world scenario of how key artifacts within an investigation can be analyzed to piece together an attack and build an attack timeline that details the actions the attacker took within the system.

The artifacts covered in this chapter are provided within the `Chapter 11\artifacts` folder on this book's companion DVD unless explicitly noted otherwise. As in Chapter 8, prepared versions of all artifacts are provided in addition to an automation script that will simplify importing the prepared artifacts into a new analysis database that will be set up on your SQL Server instance.

IMPORTING SAMPLE ARTIFACTS

The following steps will walk you through the process of creating the `INV_308_Scenario` database on your local system and loading the supplied scenario artifacts:

1. Set the `path_to_file` and `path_to_log_file` arguments and run the following syntax to create the `Inv_308_Scenario` database that we'll use throughout this chapter:

```
CREATE DATABASE [Inv_308_Scenario] ON PRIMARY
( NAME = N'Scenario', FILENAME = N'path_to_data_file\Scenario.mdf' , SIZE = 5120OKB ,
MAXSIZE = UNLIMITED, FILEGROWTH = 1024KB )
 LOG ON
( NAME = N'Scenario_log', FILENAME = N'path_to_log_file\Scenario_ldf' , SIZE =
5120OKB , MAXSIZE = 2048GB , FILEGROWTH = 10%)
COLLATE Latin1_General_CI_AS
GO
```

2. Execute the `SSFA_CreateScenariotables.sql` script located within the `Chapter 11\Scripts` folder of the companion DVD within the scenario database you created in step 1.
3. Open the `SSFA_ScenarioImport.sql` script, which can be found within the `Chapter 11\Scripts` folder on the companion DVD, and set the `@FILEPATH` variable to the `Chapter 11\Artifacts` folder on your DVD drive or to the local file location to which you have copied the sample artifacts. Once you have set the variables, execute the script.

Both raw and prepared artifacts are located within the `Chapter 11\Artifacts` folder of the companion DVD. In addition, WFT execution results retrieved from the victim system can be found within the `Chapter 11\Artifacts\WFTSQL\PROD-SQL05\2008_08_31\15_47_29` folder of the companion DVD. You'll need to import the supplied artifacts and reference the WFT results to follow along with the investigation scenario in this chapter.

INVESTIGATION SYNOPSIS

On August 31, 2008, you receive a call from a client, who states that her company may have been a victim of a security incident some time during the past 24 hours. The client explains that an employee, Mike Smith, came into work earlier that day and noticed he could not log in to the company's production SQL Server, which is called PROD-SQL05.

Another employee logged into the server and noticed that an unauthorized account, `EASYACCESS`, had been created. Mike is convinced that someone gained access to his account and insists he has not used it in the past 24 hours. The PROD-SQL05 server in question stores and processes sensitive credit card data, so the client is concerned about the possibility of a security breach. However, the client is also concerned about a delivery deadline that will prevent the server from being taken offline as a result of the mere suspicion of a security incident.

The client advises you that the PROD-SQL05 server uses native SQL Server encryption and was in the process of being changed from an old encryption key to a new key. The production data was still using the `CCProtect_Key`, so the client is unclear as to which users have had access to the encryption keys. She does know that Mike Smith was leading the key migration project. The client also informs you that production credit card information was stored within the `Orders` table. Two other tables, `OrderHistory` and `BackupOrders`, were created with test data. The focus of the investigation should be on just the `Orders` table, as it alone contains sensitive information.

The goal of your investigation will be to determine if a database intrusion has occurred and, if so, to identify the actions the intruder performed within the system. Perhaps most importantly, you need to determine what, if any, sensitive data was disclosed.

When you arrive on scene, you are briefed by the client. She provides you with the necessary user credentials and the instance name of the SQL Server at the center of the investigation. The client also advises you that a major application release is planned later in the week, so the server cannot be taken offline unless you can provide sufficient proof that an incident has occurred.

INCIDENT VERIFICATION

Because of the constraints you face when examining the victim infrastructure, you decide to use an interactive connection for the investigation. During a live interactive analysis, volatile and nonvolatile artifacts are identified and collected using minimal assistance from the victim system.

To automate the preservation of key database artifacts for analysis, you use a SQL Server incident response (IR) CD containing trusted tools—namely, your SQL Server IR scripts and the SQL Server extended version of Windows Forensic Toolchest (WFT). Inserting the IR CD into the victim system, you launch a trusted command shell using the full file path in addition to the binary name, thereby ensuring that the binary is loaded from the trusted CD.

On Sunday, August 31, at approximately 8:43 P.M., you execute the SQL Server extended version of WFT version 3.0.03 from the trusted command shell. This application executes

predeveloped scripts and gathers SQL Server artifacts from various Distributed Management Views (DMV), and Database Console Commands (DBCC). You save the outputs from the tools run during the investigation to a sanitized USB drive that you've connected to the victim system as drive letter Z.

The WFT results are stored within the `Artifacts\WFTSQL` folder on the mapped Z drive. Once execution of WFT is complete, you image the default SQL Server error named `Errorlog` with no extension using dcfldd. This file is available within the `Chapter 11\Artifacts` folder of this book's companion DVD.

PRELIMINARY REVIEW OF THE SQL SERVER ERROR LOG

You manually review the `Errorlog` file and identify several failed login attempts as seen in the following error log snippet:

```
2008-08-31 15:28:44.64 Logon       Login failed for user 'SA'. [CLIENT: 192.168.1.20]
2008-08-31 15:28:45.17 Logon       Error: 18456, Severity: 14, State: 8.
2008-08-31 15:28:45.17 Logon       Login failed for user 'SA'. [CLIENT: 192.168.1.20]
2008-08-31 15:28:45.72 Logon       Error: 18456, Severity: 14, State: 8.
2008-08-31 15:28:45.72 Logon       Login failed for user 'SA'. [CLIENT: 192.168.1.20]
2008-08-31 15:28:58.64 Logon       Error: 18456, Severity: 14, State: 5.
2008-08-31 15:28:58.64 Logon       Login failed for user 'Administrator'. [CLIENT:
192.168.1.20]
```

The error log contains several failed login attempts originating from IP address 192.168.1.20. It's clear that a SQL Server client on this host tried to gain access to the PROD-SQL05 server using the SA, Administrator, Admin, DBAdmin, ASPNET, and MSmith login accounts. These unauthorized access attempts appear to have started at 2008-08-31 15:27:09.50. There were 256 failed login attempts using 6 different accounts over 4 minutes and 20 seconds, which indicates the intruder was using some type of automated brute-force attack tool.

Error messages resulting from failed login attempts using the Administrator, Admin, DBAdmin, and ASPNET accounts have a state value of 5, which signifies the logins did not exist on the server. By contrast, error messages for the SA and MSmith logins have a state value of 8, which signifies they were present on the system and at risk of compromise. A little later in the log, you confirm that the MSmith account—one of the targets of the brute-force attack—was used to gain access to the PROD-SQL05 server from IP address 192.168.20, which is the same source of the brute-force attack.

At this point in the investigation, you've confirmed that unauthorized access was gained to the PROD-SQL05 server via a successful brute-force attack on the MSmith login and that unauthorized access was gained by the attacker at 2008-08-31 15:31:35.24.

This point will be the beginning of your investigation timeline. You also note that the attacker then used the login to gain access to the SQL Server four more times by using the compromised credentials, as seen within the following error log snippet:

```
…
2008-08-31 15:31:28.02 Logon       Login failed for user 'MSmith'. [CLIENT:
192.168.1.20]
2008-08-31 15:31:28.56 Logon       Error: 18456, Severity: 14, State: 8.
2008-08-31 15:31:28.56 Logon       Login failed for user 'MSmith'. [CLIENT:
192.168.1.20]
2008-08-31 15:31:29.10 Logon       Error: 18456, Severity: 14, State: 8.
2008-08-31 15:31:29.10 Logon       Login failed for user 'MSmith'. [CLIENT:
192.168.1.20]
2008-08-31 15:31:29.66 Logon       Error: 18456, Severity: 14, State: 8.
2008-08-31 15:31:29.66 Logon       Login failed for user 'MSmith'. [CLIENT:
192.168.1.20]
2008-08-31 15:31:35.24 Logon       Login succeeded for user 'MSmith'. Connection: non-
trusted. [CLIENT: 192.168.1.20]
2008-08-31 15:36:34.42 Logon       Login succeeded for user 'MSmith'. Connection: non-
trusted. [CLIENT: 192.168.1.20]
2008-08-31 15:38:15.31 Logon       Login succeeded for user 'MSmith'. Connection: non-
trusted. [CLIENT: 192.168.1.20]
2008-08-31 15:42:33.61 Logon       Login succeeded for user 'OSApp'. Connection: non-
trusted. [CLIENT: <local machine>]
2008-08-31 15:43:23.74 Logon       Login succeeded for user 'MSmith'. Connection: non-
trusted. [CLIENT: 192.168.1.20]
```

The OSApp account, which successfully logged on to the PROD-SQL05 instance at 2008-08-31 15:42:33.61, was used by an interactive user locally logged on to the server as indicated by the `CLIENT: <local machine>` information within the preceding log snippet. You also note that the OSApp account was not targeted during the brute-force attack, so it was not deemed malicious as part of your initial investigation.

Now that you've confirmed that a successful database intrusion occurred, you are ready to perform a preliminary review of artifacts collected by WFT in an effort to determine the actions the unauthorized user performed within the database server.

PRELIMINARY REVIEW OF WFT EXECUTION RESULTS

You complete a preliminary review of the results of the SQL Server incident response scripts executed via WFT. These results are available within the `Chapter 11\Artifacts\WFTSQL` folder of the companion DVD. You open the `Artifacts\WFTSQL\PROD-SQL05\2008_08_31\15_47_29\index.htm` file to open the main page of the WFT interface.

From there, you select the **DB Objects & Users | Logins** link to view a list of SQL Server logins and their associated creation and last update dates. While searching this page for recent activity, you identify that the EASYACCESS account was created at 15:46:03:340 on the day of the incident. In addition, you note that the name of this account is suspect and nonstandard—that is, it does not follow the naming convention in place for the client. Figure 11.1 is a snippet of the details on the EASYACESS account creation.

File Logins.txt (md5=05BFEEE52416E79B0FE9127ADCD00C91)

name	createdate	updatedate
sa	2003-04-08 09:10:35.460	2008-08-31 15:47:34.387
EASYACESS	2008-08-31 15:46:03.340	2008-08-31 15:46:03.353
MSmith	2008-08-31 08:36:02.433	2008-08-31 15:45:43.800
OSApp	2008-08-31 08:49:01.630	2008-08-31 10:00:07.103
JEmanuel	2008-08-31 08:50:52.127	2008-08-31 09:12:22.420
PROD-SQL05\Administrator	2008-08-31 09:03:00.847	2008-08-31 09:03:00.857
KSadique	2008-08-31 08:57:33.447	2008-08-31 08:57:33.463
RSanford	2008-08-31 08:55:16.900	2008-08-31 08:55:16.913
SDwyer	2008-08-31 08:54:03.613	2008-08-31 08:54:03.627
##MS_AgentSigningCertificate##	2008-08-31 08:13:22.137	2008-08-31 08:13:22.143
BUILTIN\Users	2008-08-31 08:13:16.760	2008-08-31 08:13:16.767
PROD-SQL05\SQLServer2005MSFTEUser$KEVVIE-PC$ONLINESALES	2008-08-31 08:13:16.690	2008-08-31 08:13:16.697
NT AUTHORITY\SYSTEM	2008-08-31 08:13:16.673	2008-08-31 08:13:16.683
PROD-SQL05\SQLServer2005MSSQLUser$KEVVIE-PC$ONLINESALES	2008-08-31 08:13:16.673	2008-08-31 08:13:16.683
BUILTIN\Administrators	2008-08-31 08:13:16.627	2008-08-31 08:13:16.673
##MS_SQLResourceSigningCertificate##	2008-08-31 08:13:11.363	2008-08-31 08:13:11.363
##MS_SQLReplicationSigningCertificate##	2008-08-31 08:13:11.363	2008-08-31 08:13:11.363
##MS_SQLAuthenticatorCertificate##	2008-08-31 08:13:11.363	2008-08-31 08:13:11.363

Figure 11.1 A record of the EASYACCESS login creation

You also select the **DB Objects & Users | Database Objects** link to view a list of database objects, including information on object creation and modification activity within each database on the server. Your search identifies that the `IllB3back` table was created within the `Master` database at 15:41:55 on the day of the incident. This activity came after the MSmith account was compromised and used to gain access to the SQL Server; thus it is within the scope of your investigation. You also note that the table was created in close proximity to the EASYACCESS account previously identified. Figure 11.2 shows the record of the `IllB3back` table created within the `Master` database on the victim system.

(md5=62819017B189CA5ADC50FD716AA38AC8)

name	object_id	create_date
IllB3back	1259151531	2008-08-31 15:41:55.060
sp_MScleanupmergepublisher	1179151246	2005-10-14 02:00:59.193
sp_MSrepl_startup	1163151189	2005-10-14 02:00:58.757
MSreplication_options	1147151132	2005-10-14 02:00:58.427
spt_values	1115151018	2005-10-14 01:53:52.867

Figure 11.2 A record of the IllB3back table creation

You note that #09DE7BCC, a temporary table, was also created within Tempdb at 15:32:41.367. This timing is also within the scope of the investigation and in close proximity to the other detected activity.

To further investigate the creation of the IllB3back table, you select the **Recent Activity | MRE Transactions** link within WFT and load a snapshot of recent SQL Server transaction activity for all databases within the database instance. Using the Internet Explorer find utility (Control-F), you search for the 'IllB3back' string on the page. The first hit returned is transaction 0000:00002725. The next hit immediately follows the first; this transaction, 0000:00002724, also affected the IllB3back table as seen in Figure 11.3.

```
LOP_COMMIT_XACT              |0000:00002723 |NULL
LOP_BEGIN_XACT               |0000:00002724 |NULL
LOP_LOCK_XACT                |0000:00002724 |NULL
LOP_BEGIN_XACT               |0000:00002725 |NULL
LOP_MODIFY_ROW               |0000:00002725 |dbo.IllB3back
LOP_MODIFY_ROW               |0000:00002724 |Unknown Alloc Unit
LOP_FORMAT_PAGE              |0000:00002724 |dbo.IllB3back
LOP_HOBT_DELTA               |0000:00002724 |NULL
LOP_MODIFY_ROW               |0000:00002725 |dbo.IllB3back
LOP_CREATE_ALLOCCHAIN        |0000:00002724 |NULL
LOP_COMMIT_XACT              |0000:00002725 |NULL
LOP_HOBT_DELTA               |0000:00002724 |NULL
LOP_ROOT_CHANGE              |0000:00002724 |sys.sysallocunits.clust
LOP_ROOT_CHANGE              |0000:00002724 |sys.sysallocunits.clust
LOP_SET_BITS                 |0000:00002724 |dbo.IllB3back
LOP_MODIFY_ROW               |0000:00002724 |dbo.IllB3back
LOP_SET_BITS                 |0000:00002724 |dbo.IllB3back
LOP_SET_BITS                 |0000:00002724 |dbo.IllB3back
LOP_MODIFY_ROW               |0000:00002724 |dbo.IllB3back
```

Figure 11.3 A record of transaction activity affecting the IllB3back table

By checking the transaction begin time field, you discover that both transactions were executed within the same second. You also match the creation date and time of the IllB3back table to the second transaction. The Transaction Sid field shows that both transactions were executed by SID 0x501AEC871FD432488B4A487B06C61505, which was connected to the database server using SPID 55.

You copy the unique SID value 0x501AEC871FD432488B4A487B06C61505 to the clipboard and search for it on the **DB Objects & Users | Logins** page. You find that this SID leads to the MSmith login—the same login that was compromised during the brute-force attack.

At this point, you document your findings and present them to the client. She immediately authorizes a full investigation to be conducted with the objective of identifying whether sensitive credit card information had been compromised. You then immediately initiate artifact collection on the victim SQL Server instance.

ARTIFACT COLLECTION

You obtain the version and service pack of the victim instance by examining the **DB Configuration | SQL Server Info** page within WFT. You collect both volatile and nonvolatile SQL Server artifacts as explained in Chapters 6 and 7 of this book, and then take those artifacts off-site to a forensic laboratory for analysis.

ARTIFACT ANALYSIS

Once back at the forensic lab, you prepare collected artifacts and imported them into an analysis database named `Inv_308_Scenario` for further investigation. The first analysis step after confirming unauthorized database access is to determine the level of access the account (or related accounts) has within the database server.

AUTHORIZATION

Earlier in your investigation, you identified that the MSmith account was the attacker's point of entry and was used to create the `EASYACCESS` login. Determining the level of access that these accounts have within the database server is a critical factor in meeting your investigation objective of determining whether the attacker was able to access the credit card data.

You execute the following syntax, which will return a list of notable permissions assigned to the MSmith login:

```
USE INV_308_Scenario
DECLARE @LOGIN VARCHAR (100)
DECLARE @DBNAME VARCHAR (100)
SET @DBNAME = 'ONLINESALES'
SET @LOGIN = 'MSmith'
-- ENDPOINTS
SELECT '1) Server | Endpoint' as 'category', ssr.name as 'grantee', ssp.state_desc as
'assignment', ssp.permission_name, CASE WHEN ssp.class = 105 THEN RTRIM(ssp.class_desc)
+ '.' + CONVERT (VARCHAR, ssp.major_id) ELSE ssp.class_desc END  as 'object',
ssp.class_desc as 'object_type', ssi.name as 'grantor' FROM SERV_Perm ssp, SERV_Prin
ssr, SERV_Prin ssi WHERE ssp.grantee_principal_id = ssr.principal_id and
ssp.grantor_principal_id = ssi.principal_id and ssp.class = 105 and rtrim(ssr.name) IN
(@LOGIN, 'PUBLIC')
-- LOGINS
UNION ALL SELECT '2) Server | Login', name, 'N/A', CASE WHEN logn_sql.status = 1 THEN
'ENABLED' ELSE 'DISABLED' END, 'N/A', 'N/A', 'N/A' from logn_sql WHERE
rtrim(logn_sql.name) IN (@LOGIN, 'PUBLIC')
--Fixed Server Role Membership
```

```
UNION ALL SELECT '3) Server | Fixed-server role', syu.name, 'N/A', sys.name , 'N/A',
'N/A', 'N/A' from SERV_Prin syu, SERV_Prin sys, SERV_Rmem sym where
sym.member_principal_id  = syu.[principal_id] and sym.role_principal_id =
sys.[principal_id] and syu.name IN (@LOGIN, 'PUBLIC')
-- Database
UNION ALL SELECT '4) Database | User', sdr.name, CONVERT(VARCHAR, sde.state_desc),
CONVERT(VARCHAR, sde.permission_name), sde.class_desc, sde.class_desc, sdi.name from
DBSE_Prin  sdr, DBSE_Prin  sdi, DBSE_Perm sde where sde.grantee_principal_id =
sdr.principal_id and sde.grantor_principal_id = sdi.principal_id and class = 0 and
sdr.name IN (@LOGIN, 'PUBLIC')
--Database Role Membership
UNION ALL SELECT '5) Database | Fixed/User-defined role', syu.name, 'N/A', sys.name ,
'N/A', 'N/A', 'N/A' from DBSE_Prin syu, DBSE_Prin sys, DBSE_Rmem sym where
sym.member_principal_id  = syu.[principal_id] and sym.role_principal_id =
sys.[principal_id] and syu.sid IN ((select sid from LOGN_SQL where name = @LOGIN),
(select sid from LOGN_SQL where name = 'PUBLIC'))
-- Schema
UNION ALL SELECT '6) Schema | Schema', sdi.name as 'grantee', sde.state_desc,
sde.permission_name, sao.name, sde.class_desc, sdr.name as 'grantor' from DBSE_Perm
sde,  DBSE_Prin sdi, DBSE_Prin sdr, SCHM_Data sao where sdi.principal_id =
sde.grantee_principal_id and sdr.principal_id = sde.grantor_principal_id  and
sao.schema_id = sde.major_id and sde.class = 3 and sao.[database] = @DBNAME  and
sdi.sid = (select sid from LOGN_SQL where name = @LOGIN or name = 'PUBLIC')
--Database Object
UNION ALL SELECT '7) Schema | Object', dbr.name,  dbe.state_desc, dbe.permission_name,
RTRIM(scd.name) + '.' + CASE WHEN dbe.minor_id >0 THEN RTRIM(syo.name) + '.' + (select
name from CLDT_Data where ID = syo.object_id and syo.[database] = @DBNAME and colid =
dbe.minor_id)  ELSE syo.name END as 'Object', CASE WHEN dbe.minor_id >0 THEN
RTRIM(syo.type_desc) + '_COLUMN' ELSE RTRIM(syo.type_desc) END as 'type', dbi.name from
DOBJ_Data syo,SCHM_Data scd, DBSE_Prin dbr, DBSE_Prin dbi, DBSE_Perm dbe where
dbr.principal_id = dbe.grantee_principal_id and dbi.principal_id =
dbe.grantor_principal_id and syo.[database] = @DBNAME and scd.[database] = @DBNAME and
scd.schema_id = syo.schema_id and dbr.sid = (select sid from LOGN_SQL where name =
@LOGIN or name = 'PUBLIC') and dbe.major_id = syo.object_id order by category, grantee,
object
```

When you run the preceding syntax within your `INV_308_Scenario` database, you receive the results captured in Figure 11.4. This data reveals that the MSmith login has been granted permissions through server and database fixed-role membership.

Briefly reviewing the permission hierarchy, you make the following observations:

Category 1: The MSmith account has implied permission to connect to the SQL Server endpoint through membership in the public server role.

category	grantee	assignment	permission_name	object	object_type	grantor
1) Server | Endpoint	public	GRANT	CONNECT	ENDPOINT.2	ENDPOINT	sa
1) Server | Endpoint	public	GRANT	CONNECT	ENDPOINT.3	ENDPOINT	sa
1) Server | Endpoint	public	GRANT	CONNECT	ENDPOINT.4	ENDPOINT	sa
1) Server | Endpoint	public	GRANT	CONNECT	ENDPOINT.5	ENDPOINT	sa
2) Server | Login	MSmith	N/A	ENABLED	N/A	N/A	N/A
3) Server | Fixed-server role	MSmith	N/A	securityadmin	N/A	N/A	N/A
3) Server | Fixed-server role	MSmith	N/A	serveradmin	N/A	N/A	N/A
3) Server | Fixed-server role	MSmith	N/A	setupadmin	N/A	N/A	N/A
4) Database | User	MSmith	GRANT	CONNECT	DATABASE	DATABASE	dbo
5) Database | Fixed/User-defined role	MSmith	N/A	db_datareader	N/A	N/A	N/A
5) Database | Fixed/User-defined role	MSmith	N/A	db_datawriter	N/A	N/A	N/A

Figure 11.4 Permissions granted to the MSmith login

Category 2: The second stage in the hierarchy shows that MSmith has an enabled login account, which will allow this account access to the SQL Server instance.

Category 3: Membership in the securityadmin, serveradmin, and setupadmin roles grants the MSmith account the ability to assign server-wide permissions, alter the server and/or endpoints, change permissions on the server, and alter linked servers. (Refer to Chapters 2 and 8 for additional details about permissions.)

Category 4: The CONNECT permission gives the MSmith login access to the `OnlineSales` database.

Category 5: Fixed-database role membership in the `db_datareader` and `db_datawriter` roles grants the MSmith account read and write access within the `OnlineSales` database, including the `Orders` table—which contains sensitive information.

Next, you rerun your earlier executed syntax, this time using the EASYACCESS as the login, to determine the level of permission the EASYACCESS account had within the database. The results you receive, which are captured in Figure 11.5, show that no explicit or implied permissions were assigned to the EASYACCESS login.

category	grantee	assignment	permission_name	object	object_type	grantor
1) Server | Endpoint	public	GRANT	CONNECT	ENDPOINT.2	ENDPOINT	sa
1) Server | Endpoint	public	GRANT	CONNECT	ENDPOINT.3	ENDPOINT	sa
1) Server | Endpoint	public	GRANT	CONNECT	ENDPOINT.4	ENDPOINT	sa
1) Server | Endpoint	public	GRANT	CONNECT	ENDPOINT.5	ENDPOINT	sa

Figure 11.5 Permissions granted to the EASYACCESS login

Because no permission has been explicitly denied to the MSmith login, and because the highest level of access is the permission assigned within Category 3 (fixed-server role membership), you check the permissions placed on encryption keys and certificates. These permissions will be the deciding factor in confirming whether the MSmith account has sufficient access within the database to access the credit card data.

NATIVE SQL SERVER ENCRYPTION

As you discussed with the client at the beginning of the investigation, encryption is in use on the database server. The MSmith account, which you verified was compromised during the brute-force attack, is believed to have access to the encryption keys.

To observe a listing of keys gathered from the server during artifact collection, you execute the following syntax against the `Inv_308_Scenario` database. This syntax returns a list of all symmetric keys on the victim system:

```
Select name, create_date, modify_date, key_guid from dbse_semk
```

The results of this query, which are captured in Figure 11.6, show that in addition to the database master key, there are two encryption keys on the system. This finding is a good indicator that encryption is in use.

name	create_date	modify_date	key_guid
##MS_DatabaseMasterKey##	2008-08-31 08:58:57.453	2008-08-31 08:58:57.470	D7E9E400-5692-4434-BF30-0D9C16D14B8E
CCProtect_Key	2008-08-31 08:59:00.790	2008-08-31 08:59:00.790	DBCF3B00-92C7-4C6C-B2D6-29D5896E282C
CCProtect_NewKey	2008-08-31 08:59:32.637	2008-08-31 08:59:32.650	5AD11A00-A95F-421E-9009-6AE0BB345A02

Figure 11.6 Identified symmetric keys on the victim system

Because native SQL Server encryption supports only the varbinary data type, to further confirm the use of encryption, you perform a search on all columns in use within the database using the varbinary data type. To do so, you execute the following syntax within the `INV_308_Scenario` database:

```
select dbj.name as 'object', dbj.type_desc as 'object_type', cdt.name as 'column_name',
st.name as 'datatype' FROM CLDT_Data cdt, systypes st, DOBJ_Data dbj where st.xusertype
= cdt.xusertype and dbj.object_id = cdt.id and st.name = 'Varbinary' and
dbj.is_ms_shipped = 0
```

Once this query is run, you receive the results captured in Figure 11.7. These results indicate that three tables use the varbinary data type.

object	object_type	column_name	datatype
sysdiagrams	USER_TABLE	definition	varbinary
sp_creatediagram	SQL_STORED_PROCEDURE	@definition	varbinary
sp_alterdiagram	SQL_STORED_PROCEDURE	@definition	varbinary
Orders	USER_TABLE	CCNumber	varbinary
ORDERSBKU	USER_TABLE	CCNumber	varbinary

Figure 11.7 Identified tables using the varbinary data type

The `sysdiagrams` table, although set with an `is_ms_shipped` value of 0, is not a user object; thus the client would not use it to store application data. With this caveat in mind, you determine that the only two tables using the varbinary data type are the `Orders` and `ORDERSBKU` tables. Earlier in the investigation, the client mentioned to you that the `ORDERSBKU` table did not contain production credit card information and that your investigation should focus on the `Orders` table. Therefore you exclude the `ORDERSBKU` table from the remainder of the investigation.

You've now confirmed that encrypted data is likely to be found within the `CCNumber` column of the `Orders` table. You execute the following statement to determine if the compromised database account has access to the encryption keys created on the victim server:

```
SELECT syk.name as 'key_name', class_desc, spr.name as 'db_user', permission_name,
state_desc, major_id from dbse_perm sdp, dbse_semk syk, dbse_prin spr where class IN
(24, 25, 26) and syk.symmetric_key_id = sdp.major_id and spr.principal_id =
sdp.grantee_principal_id and class = 24
ORDER BY key_name, spr.name, permission_name, state_desc
```

When this code is run, you receive the results captured in Figure 11.8. As seen in the figure, the MSmith login was denied VIEW DEFINITION permission on the `CCProtect_Key`. This login does, however, have VIEW DEFINITION permission on the `CCProtect_NewKey` along with another SQL Server user.

key_name	class_desc	db_user	permission_name	state_desc	major_id
CCProtect_Key	SYMMETRIC_KEYS	MSmith	VIEW DEFINITION	DENY	256
CCProtect_NewKey	SYMMETRIC_KEYS	JEmanuel	VIEW DEFINITION	GRANT	257
CCProtect_NewKey	SYMMETRIC_KEYS	MSmith	VIEW DEFINITION	GRANT	257

Figure 11.8 Symmetric key permissions

For a user to have adequate permissions to encrypt and decrypt data using an encryption key within SQL Server, the user would need to have any level of permission on the

key granted to that user. As your findings in Figure 11.8 reveal, the VIEW DEFINITION permission on the `CCProtect_NewKey` would have allowed the attacker to decrypt or encrypt data encrypted by it. However, the `CCProtect_Key` is another symmetric key on the system to which the MSmith account has been explicitly denied access. Earlier the client stated that `CCProtect_Key` is the current key being used for encryption. You have now proved the compromised account did not have access to this key to decrypt the sensitive data.

The final objective of your investigation is to determine the actions performed by the unauthorized user within the database. This objective can be satisfied through activity reconstruction.

ACTIVITY RECONSTRUCTION

The incident timeline developed thus far in your investigation will allow you to limit the scope of the activity reconstruction. Table 11.1 summarizes the incident timeline you have created.

A key method of identifying past activity is reviewing command execution. Thus the next step in your investigation focuses on identifying the commands executed by the attacker on the SQL Server during his or her period of unauthorized access. The first artifact you analyze is server state information in an effort to determine details about the intruder's actions.

Table 11.1 Incident Timeline

Time	Event	Source
2008-08-31 15:27:09.50	Brute-force attack initiated	192.168.1.20
2008-08-31 15:31:35.24	Attacker gained unauthorized access to the PROD-SQL05 server using the MSmith login account	MSmith login
2008-08-31 15:32:41.367	Temporary object `#09DE7BCC` was created within `Tempdb` database	MSmith login
2008-08-31 15:36:34.42	Attacker reconnects to PROD-SQL05	MSmith login
2008-08-31 15:38:15.31	Attacker reconnects to PROD-SQL05	MSmith login
2008-08-31 15:41:55.060	`IllB3back` table was created within `Master` database	MSmith login
2008-08-31 15:43:23.74	Attacker reconnects to PROD-SQL05	MSmith login
2008-08-31 15:46:03.340	`EASYACCESS` login created	MSmith login

SERVER STATE

The server state artifact captures the current state of active connections and processes on a server. It includes a wealth of information, ranging from active user connections to database processes executing in the background to the last command executed by each connected user. Within WFT, you select the **Active Connections | Connections and Sessions** links to view active sessions on the victim system at the time of automated artifact collection. After reviewing the data on these pages, you conclude that the attacker has disconnected from the system: The MSmith login account does not appear in the list of actively connected users.

The next artifact you analyze is the plan cache, which may have cached the attacker's previously executed database statements.

PLAN CACHE

Reviewing the plan cache enables you to pinpoint anomalous entries that may be associated with the attacker. Although plan cache entries cannot be associated with a specific user, the process of associating plan cache entries to other database activity can map actions to a specific SQL Server login or database user. As your next move, you run the following syntax within the `INV_308_Scenario` database to return a list of plan cache entries that were cached during the scope of the incident:

```
SELECT * from plch_data order by convert(datetime, creation_time) desc
```

A snippet of the results returned appears in Figure 11.9.

Your first search of the plan cache was done in an effort to identify which statement forced the creation of the `#09DE7BCC` temporary table within the `Tempdb` database. Because the creation of this object has already been mapped back to the MSmith login, you can place the attacker at a specific cache entry. This information serves as a starting point for identifying past activity as well as future activity not yet discovered.

As shown earlier in Table 11.1, the `#09DE7BCC` table was created at 2008-08-31 15:32:41.367. Using this time to the second, you perform a search of the plan cache entries in the hopes of finding a cached statement that was the source of the temporary table creation:

```
SELECT * FROM PLCH_Data WHERE CAST ([Creation_time] AS DATETIME) >= cast ('2008-08-31
15:32:41.000' AS DATETIME) AND CAST ([Creation_time] AS DATETIME) <= CAST ('2008-08-31
15:32:41.999' AS DATETIME) or CAST ([Last_execution_time] AS DATETIME) >= cast ('2008-
08-31 15:32:41.000' AS DATETIME) AND CAST ([Last_execution_time] AS DATETIME) <= CAST
('2008-08-31 15:32:41.999' AS DATETIME) order by last_execution_time desc
```

Figure 11.10 shows a snippet of the results produced by this query.

Creation_time	Last_execution_time	statement
2008-08-31 15:45:15.420	2008-08-31 15:45:43.780	create procedure sys.sp_password @old sysname ...
2008-08-31 15:41:17.973	2008-08-31 15:41:17.990	select * from sys.asymmetric_keys
2008-08-31 15:41:05.123	2008-08-31 15:41:05.140	select * from sys.symmetric_keys
2008-08-31 15:40:28.390	2008-08-31 15:40:28.407	select * from sysobjects where name like '%password%'
2008-08-31 15:40:22.490	2008-08-31 15:41:55.060	select * from sysobjects where name like '%key%'
2008-08-31 15:40:10.610	2008-08-31 15:40:10.623	select * from sysobjects where name like '%passwords%'
2008-08-31 15:38:57.957	2008-08-31 15:38:57.957	select * from orderhistory
2008-08-31 15:33:58.333	2008-08-31 15:38:47.980	select * from sysobjects where type = 'U'
2008-08-31 15:33:30.773	2008-08-31 15:34:38.330	select * from sys.sysusers
2008-08-31 15:33:29.047	2008-08-31 15:33:29.047	(@1 varchar(8000),@2 tinyint)UPDATE [ORDERS] se...
2008-08-31 15:33:14.053	2008-08-31 15:33:14.053	(@1 varchar(8000),@2 smallint)UPDATE [ORDERS] s...
2008-08-31 15:32:41.617	2008-08-31 15:32:41.617	create procedure sys.sp_helpdb -- 1995/12/20 15:34 ...
2008-08-31 15:32:41.600	2008-08-31 15:32:41.600	create procedure sys.sp_helpdb -- 1995/12/20 15:34 ...
2008-08-31 15:32:41.600	2008-08-31 15:32:41.617	create procedure sys.sp_helpdb -- 1995/12/20 15:34 ...
2008-08-31 15:32:41.587	2008-08-31 15:32:41.600	update #spdbdesc set dbsize = (select str(conve...
2008-08-31 15:32:41.570	2008-08-31 15:32:41.570	update #spdbdesc set dbsize = (select str(conve...
2008-08-31 15:32:41.477	2008-08-31 15:32:41.477	create procedure sys.sp_helpdb -- 1995/12/20 15:34 ...
2008-08-31 15:32:41.447	2008-08-31 15:32:41.447	update #spdbdesc set dbsize = (select str(conve...
2008-08-31 15:32:41.400	2008-08-31 15:32:41.413	update #spdbdesc set dbsize = (select str(conve...
2008-08-31 15:32:41.400	2008-08-31 15:32:41.400	create procedure sys.sp_helpdb -- 1995/12/20 15:34 ...
2008-08-31 15:32:41.383	2008-08-31 15:32:41.400	create procedure sys.sp_helpdb -- 1995/12/20 15:34 ...
2008-08-31 15:32:41.367	2008-08-31 15:32:41.400	create procedure sys.sp_helpdb -- 1995/12/20 15:34 ...
2008-08-31 15:32:41.367	2008-08-31 15:32:41.600	create procedure sys.sp_helpdb -- 1995/12/20 15:34 ...
2008-08-31 15:32:41.350	2008-08-31 15:32:41.367	create procedure sys.sp_helpdb -- 1995/12/20 15:34 ...
2008-08-31 15:32:41.320	2008-08-31 15:32:41.367	create procedure sys.sp_helpdb -- 1995/12/20 15:34 ...
2008-08-31 15:32:16.740	2008-08-31 15:32:16.773	select * from sys.syslogins

Figure 11.9 A snippet of the plan cache entries cached during the scope of the incident

Creation_time	Last_execution_time	statement
2008-08-31 15:32:41.600	2008-08-31 15:32:41.617	create procedure sys.sp_helpdb -- 1995/12/20 15...
2008-08-31 15:32:41.617	2008-08-31 15:32:41.617	create procedure sys.sp_helpdb -- 1995/12/20 15...
2008-08-31 15:32:41.600	2008-08-31 15:32:41.600	create procedure sys.sp_helpdb -- 1995/12/20 15...
2008-08-31 15:32:41.367	2008-08-31 15:32:41.600	create procedure sys.sp_helpdb -- 1995/12/20 15...
2008-08-31 15:32:41.587	2008-08-31 15:32:41.600	update #spdbdesc set dbsize = (select str(co...
2008-08-31 15:32:41.570	2008-08-31 15:32:41.570	update #spdbdesc set dbsize = (select str(co...
2008-08-31 15:32:41.477	2008-08-31 15:32:41.477	create procedure sys.sp_helpdb -- 1995/12/20 15...
2008-08-31 15:32:41.447	2008-08-31 15:32:41.447	update #spdbdesc set dbsize = (select str(co...
2008-08-31 15:32:41.400	2008-08-31 15:32:41.413	update #spdbdesc set dbsize = (select str(co...
2008-08-31 15:32:41.367	2008-08-31 15:32:41.400	create procedure sys.sp_helpdb -- 1995/12/20 15...
2008-08-31 15:32:41.400	2008-08-31 15:32:41.400	create procedure sys.sp_helpdb -- 1995/12/20 15...
2008-08-31 15:32:41.383	2008-08-31 15:32:41.400	create procedure sys.sp_helpdb -- 1995/12/20 15...
2008-08-31 15:32:41.350	2008-08-31 15:32:41.367	create procedure sys.sp_helpdb -- 1995/12/20 15...
2008-08-31 15:32:41.320	2008-08-31 15:32:41.367	create procedure sys.sp_helpdb -- 1995/12/20 15...

Figure 11.10 Plan cache entries created or executed at 2008-08-31 15:32:41

At first glance, you notice that the `sys.sp_helpdb` procedure was created multiple times—when a stored procedure is cached, its definition is displayed within the plan cache after execution. In close proximity to the `sp_helpdb` execution are four statements that update the temporary table `#spdbdesc`. The naming convention of this table seems to be related to `sp_helpdb`, so you select the object definition of `sp_helpdb` for further analysis. You gather the definition for `sp_helpdb` from server SP2, which runs SQL Server 2005. This version of SQL Server was the same major and minor release used on the victim, based on the server information obtained from the **DB Configuration | SQL Server Info** link within WFT.

On the trusted instance, you execute the following syntax:

```
SELECT OBJECT_DEFINITION (OBJECT_ID ('sys.sp_helpdb'))
```

Within the returned definition, you identify the statement that creates the `#spdbdesc` table. You note the following snippet of code, which shows the syntax within the `sp_helpdb` procedure that creates the temporary table logged under the transaction executed by the MSmith account:

```
create table #spdbdesc  (   dbname sysname,   owner sysname null,   created
nvarchar(11),   dbid smallint,   dbdesc nvarchar(600) null,   dbsize  nvarchar(13)
null,   cmptlevel tinyint  )      /*  **  If no database name given, get 'em all.  */
```

Although plan cache entries cannot be directly mapped back to a user or SID as operations within the transaction log can be, you map the executed statement within the plan cache to the resulting table creation data logged within the transaction log; this mapping again leads back to the MSmith SID. This discovery proves that the MSmith account executed the `sp_helpdb` procedure once the user gained access to the database server. Because `sp_helpdb` provides information about the databases on a SQL Server instance, you suspect that the attacker executed this procedure to learn the structure of the database server to which he or she just gained access. With an idea of the attacker's objective, you execute the following query, which allows you to view all plan cache entries in descending order by `last_execution_time`:

```
select * from plch_data order by convert(datetime, last_execution_time )desc
```

Once this query is executed, database reconnaissance-related entries are noted in the plan cache. These entries are highlighted in Figure 11.11.

The transaction execution history suggests that database reconnaissance activity has, indeed, taken place. Database reconnaissance typically involves the execution of vague statements that return a manageable amount of data and allow the user to sort out the

	Creation_time	Last_execution_time	statement	dbid
1	2008-08-31 15:31:54.050	2008-08-31 15:47:35.507	(@1 int,@2 smallint)UPDATE [ORDERS] set [SHIPSTAT...	5
2	2008-08-31 15:47:34.433	2008-08-31 15:47:34.433	-- -- Source: SQL Server Forensic Analysis -- Author: Ke...	1
3	2008-08-31 15:47:02.653	2008-08-31 15:47:13.630	select * from sys.sysmessages	5
4	2008-08-31 15:45:15.420	2008-08-31 15:45:43.780	create procedure sys.sp_password @old sysname = ...	32767
5	2008-08-31 15:40:22.490	2008-08-31 15:41:55.060	select * from sysobjects where name like '%key%'	5
6	2008-08-31 15:41:17.973	2008-08-31 15:41:17.990	select * from sys.asymmetric_keys	5
7	2008-08-31 15:41:05.123	2008-08-31 15:41:05.140	select * from sys.symmetric_keys	5
8	2008-08-31 15:40:28.390	2008-08-31 15:40:28.407	select * from sysobjects where name like '%password%'	5
9	2008-08-31 15:40:10.610	2008-08-31 15:40:10.623	select * from sysobjects where name like '%passwords%'	5
10	2008-08-31 15:38:57.957	2008-08-31 15:38:57.957	select * from orderhistory	5
11	2008-08-31 15:33:58.333	2008-08-31 15:38:47.980	select * from sysobjects where type = 'U'	5
12	2008-08-31 15:29:00.070	2008-08-31 15:34:44.047	(@1 tinyint)SELECT [FirstName],[LastName],[Address] F...	5
13	2008-08-31 15:33:30.773	2008-08-31 15:34:38.330	select * from sys.sysusers	5
14	2008-08-31 15:33:29.047	2008-08-31 15:33:29.047	(@1 varchar(8000),@2 tinyint)UPDATE [ORDERS] set [...	5
15	2008-08-31 15:33:14.053	2008-08-31 15:33:14.053	(@1 varchar(8000),@2 smallint)UPDATE [ORDERS] set ...	5
16	2008-08-31 15:32:41.600	2008-08-31 15:32:41.617	create procedure sys.sp_helpdb -- 1995/12/20 15:34 #...	32767
17	2008-08-31 15:32:41.617	2008-08-31 15:32:41.617	create procedure sys.sp_helpdb -- 1995/12/20 15:34 #...	32767
18	2008-08-31 15:32:41.600	2008-08-31 15:32:41.600	create procedure sys.sp_helpdb -- 1995/12/20 15:34 #...	32767
19	2008-08-31 15:32:41.367	2008-08-31 15:32:41.600	create procedure sys.sp_helpdb -- 1995/12/20 15:34 #...	32767
20	2008-08-31 15:32:41.587	2008-08-31 15:32:41.600	update #spdbdesc set dbsize = (select str(convert(...	1
21	2008-08-31 15:32:41.570	2008-08-31 15:32:41.570	update #spdbdesc set dbsize = (select str(convert(...	1
22	2008-08-31 15:32:41.477	2008-08-31 15:32:41.477	create procedure sys.sp_helpdb -- 1995/12/20 15:34 #...	32767
23	2008-08-31 15:32:41.447	2008-08-31 15:32:41.447	update #spdbdesc set dbsize = (select str(convert(...	1
24	2008-08-31 15:32:41.400	2008-08-31 15:32:41.413	update #spdbdesc set dbsize = (select str(convert(...	1
25	2008-08-31 15:32:41.367	2008-08-31 15:32:41.400	create procedure sys.sp_helpdb -- 1995/12/20 15:34 #...	32767
26	2008-08-31 15:32:41.400	2008-08-31 15:32:41.400	create procedure sys.sp_helpdb -- 1995/12/20 15:34 #...	32767
27	2008-08-31 15:32:41.383	2008-08-31 15:32:41.400	create procedure sys.sp_helpdb -- 1995/12/20 15:34 #...	32767
28	2008-08-31 15:32:41.350	2008-08-31 15:32:41.367	create procedure sys.sp_helpdb -- 1995/12/20 15:34 #...	32767
29	2008-08-31 15:32:41.320	2008-08-31 15:32:41.367	create procedure sys.sp_helpdb -- 1995/12/20 15:34 #...	32767
30	2008-08-31 15:32:16.740	2008-08-31 15:32:16.773	select * from sys.syslogins	1

Figure 11.11 Database reconnaissance-related plan cache entries

information he or she needs. Attackers are typically unfamiliar with the structure of a database, but they must be careful when attempting to learn it because that effort may attract attention or simply take time away from their primary objective.

Within the results captured in Figure 11.11. some fields have been reordered due to formatting limitations. By beginning at row 30 and working backward, you identify the following actions believed to be executed by the unauthorized user:

- Row 30: Queried the `sys.syslogins` view within the `Master` database to get an understanding of the accounts on the SQL Server instance. Execution of this statement within the `Master` database was identified by the `dbid` value of 1, which maps to the `Master` database (as seen by executing the **DB Objects & Users | Database** link within WFT).

- Rows 29–16: Executed `sp_helpdb` to learn about the databases on the SQL Server instance (which you proved earlier). Because `sp_helpdb` is a system object that can be executed, it is launched from the `Resource` database (database ID 32767).
- Row 13: Switched database context to the `OnlineSales` database after the user received the results from `sp_helpdb` as identified by the `dbid` value of 5 in the next database reconnaissance-related query.
- Row 13: Queried the `sys.sysusers` view to learn about the database users within the `OnlineSales` database.
- Row 11: Queried `sysobjects` with a `where type = 'U'` clause, which returns a listing of all tables within the database.
- Row 10: Executed a `select` statement against the `orderhistory` table, which would have been listed within the previous `sysobjects` results.
- Rows 9–5: Executed multiple queries against multiple views in search of password or encryption key–related information, perhaps in an attempt to decrypt the data within the `orderhistory` table believed to be encrypted by the attacker.
- Row 4: After the reconnaissance, executed the `sp_password` procedure, as identified by its definition being recorded as a statement.

Even your artifact collection actions are logged within the plan cache. Row 2 from the results captured in Figure 11.11 shows the execution of the IR scripts used during incident response and, therefore, can be ignored.

Other entries within the plan cache involve updates to the `Orders` table. However, all entries affecting the `Orders` table within the transaction log were executed by the OSApp login, which is also interactively logged into the system. (This fact can be verified by reviewing the information within **Active Connections | Connections and Sessions** links.)

You now conclude that the `SELECT INTO` statement executed by the unauthorized user is not resident in the cache, either because the database engine did not cache it or because it was evicted prior to its preservation.

This step completes your plan cache analysis. Your final artifact to analyze is the active VLFs obtained from the OnlineSales and Master databases of the victim system.

ACTIVE VLFS

To begin analysis of active VLF data, you execute the following syntax within the database to return a summary of all transactions performed by SID 0x501AEC871FD432488B4 A487B06C61505, which belongs to the MSmith login:

```
SELECT DISTINCT [TRANSACTION NAME], [BEGIN TIME], [current lsn] FROM AVLF_TLOG where
[TRANSACTION ID] IN (SELECT distinct [TRANSACTION ID] FROM AVLF_TLOG WHERE [TRANSACTION
```

```
SID] = '0x501AEC871FD432488B4A487B06C61505') and [transaction name] <> 'NULL' order by
[current lsn]
```

When this syntax is run, you receive the results captured in Figure 11.12.

TRANSACTION NAME	BEGIN TIME	current lsn
CREATE STATISTICS	2008/08/31 10:21:57:683	00000010:00001335:000c
CREATE STATISTICS	2008/08/31 10:21:57:683	00000010:0000133a:0001
SELECT INTO	2008/08/31 15:41:55:060	000000c8:00000150:0002
SELECT INTO	2008/08/31 15:41:55:120	000000c8:00000168:0001
FirstPage Alloc	2008/08/31 15:41:55:137	000000c8:00000168:0003
ALTER LOGIN	2008/08/31 15:45:15:437	000000c8:00000168:008e
ALTER LOGIN	2008/08/31 15:45:19:880	000000c8:00000168:0091
ALTER LOGIN	2008/08/31 15:45:43:780	000000c8:00000168:0094
CREATE LOGIN	2008/08/31 15:46:03:340	000000c8:00000180:0001

Figure 11.12 Summary of all transactions performed by the suspect SID

You immediately discount the two `CREATE STATISTICS` transactions, which occurred prior to the unauthorized user gaining access to the database server. Instead, you focus on the entries highlighted in Figure 11.12. To obtain more details about these transactions, you execute the following slightly modified query, which returns 269 rows:

```
SELECT [database], [transaction ID], [transaction name], [operation], [allocunitname],
[begin time], [end time], spid, [transaction sid] FROM AVLF_TLOG where [TRANSACTION ID]
IN (SELECT DISTINCT [TRANSACTION ID] FROM AVLF_TLOG WHERE [TRANSACTION SID] =
'0x501AEC871FD432488B4A487B06C61505')
```

(*Note:* The preceding query and its results will be referred to several times throughout the rest of this chapter as "the detailed transaction summary results.")

To see the logical progression of the attacker's actions on the database, you step through the detailed transaction summary results in sequence by transaction ID, beginning with transaction 2723.

You first analyze transaction ID 2723, a `SELECT INTO` statement located within row 21 of the detailed transaction summary results. Figure 11.13 contains a snippet of this transaction.

database	transaction ID	transaction name	operation	allocunitname	begin time
master	0000:00002723	SELECT INTO	LOP_BEGIN_XACT	NULL	2008/08/31 15:41:55:060
master	0000:00002723	NULL	LOP_LOCK_XACT	NULL	NULL
master	0000:00002723	NULL	LOP_INSERT_ROWS	sys.sysschobjs.clst	NULL

Figure 11.13 Snippet of transaction ID 2723

Unfortunately, the `SELECT INTO` statement did not record the affected table within the `Master` database in the `AllocUnitName` field. Nevertheless, you examine the date and time when the transaction was committed. Matching this information to an object within the `Master` database will allow you to identify the created table. Transaction 2723 began on 2008/08/31 15:41:55:060. You develop the following syntax to compare the time the transaction was initiated against the object creation times within the `DOBJ_Data` table, which contains all objects and their associated creation and last update times on the victim server:

```
SELECT * from dobj_data where convert(datetime, create_date) = convert(datetime, '2008-
08-31 15:41:55:060')
```

When this query is run, it returns the `IllB3back` table. This result reaffirms your earlier investigation finding. A snippet of the results appears in Figure 11.14. Because the original plan cache entry could not be recovered during your investigation, the other table involved in the `SELECT INTO` statement cannot be identified.

database	name	object_id	create_date	modify_date	type	type_desc
master	lllB3back	1259151531	2008-08-31 15:41:55.060	2008-08-31 15:41:55.120	U	USER_TABLE

Figure 11.14 Database objects created at 2008/08/31 15:41:55:060

The second statement executed by the unauthorized user is transaction ID 2724, which is located on row 100 of the previous detailed transaction summary results. A snippet of transaction ID 2724 within these results is shown in Figure 11.15.

database	transaction ID	transaction name	operation	allocunitname
master	0000:00002724	SELECT INTO	LOP_BEGIN_XACT	NULL
master	0000:00002724	NULL	LOP_LOCK_XACT	NULL
master	0000:00002725	FirstPage Alloc	LOP_BEGIN_XACT	NULL
master	0000:00002725	NULL	LOP_MODIFY_ROW	dbo.lllB3back

Figure 11.15 Snippet of transaction ID 2724

Stepping through the various operations within this transaction, you focus on row 240. It reveals that the transaction was aborted at 2008/08/31 15:43:16:570, as shown in Figure 11.16. This operation would have reverted the copying of data from the source table to the destination `IllB3Back` table.

database	transaction ID	transaction name	operation	allocunitname	begin time	end time
master	0000:00002724	NULL	LOP_MODIFY_ROW	dbo.lllB3back	NULL	NULL
master	0000:00002724	NULL	LOP_HOBT_DELTA	NULL	NULL	NULL
master	0000:00002724	NULL	LOP_MODIFY_ROW	Unknown Alloc Unit	NULL	NULL
master	0000:00002724	NULL	LOP_ABORT_XACT	NULL	NULL	2008/08/31 15:43:16:570

Figure 11.16 An aborted transaction

Transaction ID 2725 was executed by the database engine in response to the `SELECT INTO` statement executed via transaction 2724. Transaction 2724 began allocating data pages for the `I11B3back` table before it was aborted. Figure 11.17 shows the `I11B3back` table referenced under the allocation unit field of transaction 2725.

transaction ID	transaction name	operation	allocunitname	begin time
0000:00002724	SELECT INTO	LOP_BEGIN_XACT	NULL	2008/08/31 15:41:55:120
0000:00002724	NULL	LOP_LOCK_XACT	NULL	NULL
0000:00002725	FirstPage Alloc	LOP_BEGIN_XACT	NULL	2008/08/31 15:41:55:137
0000:00002725	NULL	LOP_MODIFY_ROW	dbo.lllB3back	NULL
0000:00002724	NULL	LOP_MODIFY_ROW	Unknown Alloc Unit	NULL

Figure 11.17 The lllB3back table referenced under the allocation unit field of transaction 2725

As seen in Figure 11.18, transaction ID 2726 related to the alteration of a login. This transaction shows that a login was modified, but that the operations were immediately aborted. This behavior is usually associated with a validation check performed on the server, which was not meant to proceed with the login alteration.

transaction ID	transaction name	operation	allocunitname	begin time	end time
0000:00002726	ALTER LOGIN	LOP_BEGIN_XACT	NULL	2008/08/31 15:45:15:437	NULL
0000:00002726	NULL	LOP_LOCK_XACT	NULL	NULL	NULL
0000:00002726	NULL	LOP_ABORT_XACT	NULL	NULL	2008/08/31 15:45:15:437

Figure 11.18 Snippet of an aborted alter login operation

In the next part of your investigation, you develop the following syntax to query the plan cache and identify cached statements that were created or used at the same time of the transaction to the second:

```
SELECT * FROM PLCH_Data WHERE CAST ([Creation_time] AS DATETIME) >= cast ('2008-08-31
15:45:15.000' AS DATETIME) AND CAST ([Creation_time] AS DATETIME) <= CAST ('2008-08-31
15:45:15.999' AS DATETIME) or CAST ([Last_execution_time] AS DATETIME) >= cast ('2008-
08-31 15:45:15.000' AS DATETIME) AND CAST ([Last_execution_time] AS DATETIME) <= CAST
('2008-08-31 15:45:15.999' AS DATETIME) order by last_execution_time desc
```

When this code is executed, the results (captured in Figure 11.19) reveal that the cache plan entry for `sp_password` was created at the same time to the second as the `ALTER LOGIN` operation within the transaction log.

Creation_time	Last_execution_time	statement
2008-08-31 15:45:15.420	2008-08-31 15:45:43.780	create procedure sys.sp_password @old sysna...

Figure 11.19 Plan cache entries created or last executed at Database objects created at 2008-08-31 15:45:15

The creation time shown in Figure 11.19 is the date and time the entry was cached, and the `last_execution_time` is the last time the plan entry was used by a database user. Reviewing your log summary, you note that two additional `ALTER LOGIN` operations were executed under transactions 2727 and 2728; these operations are found in rows 244 and 247 within the detailed transaction summary results (see Figure 11.20).

transaction ID	transaction name	operation	allocunitname	begin time	end time
0000:00002726	ALTER LOGIN	LOP_BEGIN_XACT	NULL	2008/08/31 15:45:15:437	NULL
0000:00002726	NULL	LOP_LOCK_XACT	NULL	NULL	NULL
0000:00002726	NULL	LOP_ABORT_XACT	NULL	NULL	2008/08/31 15:45:15:437
0000:00002727	ALTER LOGIN	LOP_BEGIN_XACT	NULL	2008/08/31 15:45:19:880	NULL
0000:00002727	NULL	LOP_LOCK_XACT	NULL	NULL	NULL
0000:00002727	NULL	LOP_ABORT_XACT	NULL	NULL	2008/08/31 15:45:19:880
0000:00002728	ALTER LOGIN	LOP_BEGIN_XACT	NULL	2008/08/31 15:45:43:780	NULL
0000:00002728	NULL	LOP_LOCK_XACT	NULL	NULL	NULL
0000:00002728	NULL	LOP_DELETE_RO...	sys.sysobjvalues.clst	NULL	NULL
0000:00002728	NULL	LOP_INSERT_ROWS	sys.sysobjvalues.clst	NULL	NULL
0000:00002728	NULL	LOP_MODIFY_ROW	sys.sysxlgns.cl	NULL	NULL
0000:00002728	NULL	LOP_MODIFY_ROW	sys.sysxlgns.cl	NULL	NULL
0000:00002728	NULL	LOP_COMMIT_XACT	NULL	NULL	2008/08/31 15:45:43:797

Figure 11.20 Two additional ALTER LOGIN operations executed under transactions 2727 and 2728

In Figure 11.20, the `ALTER LOGIN` operation was executed under transaction ID 2727 (row 253) but immediately failed after initiation. This mirrors transaction 2726's result,

which indicates another password reset-related failure. Transaction 2728, however, executed successfully, as evidenced by the LOP_COMMIT_XACT operation logged at 2008/08/31 15:45:43:797.

You develop the following syntax to compare transaction 2726's commit time with logins created or updated, to the second, during the successful account update performed by transaction 2728:

```
SELECT * FROM LOGN_SQL WHERE CAST ([UPDATEDATE] AS DATETIME) >= cast ('2008/08/31
15:45:43:000' AS DATETIME) AND CAST ([UPDATEDATE] AS DATETIME) < CAST ('2008/08/31
15:45:43:999' AS DATETIME)
```

When this syntax is run, it produces the results shown in Figure 11.21. It appears that the MSmith account was updated in accordance to the transaction log entry to the second. This finding also allows you to reconfirm Mike's statement from earlier during the investigation—namely, that he could not gain access to his account earlier in the day.

name	createdate	updatedate	accdate
MSmith	2008-08-31 08:36:02.433	2008-08-31 15:45:43.800	2008-08-31 08:36:02.433

Figure 11.21 Logins updated at 2008/08/31 15:45:43

Transaction 2729 (row 254) is a CREATE statement that was analyzed earlier in your investigation. It led to your discovery that the MSmith account created the EASYACESS account within the database server.

No other notable findings were identified in the analysis of the other artifacts collected from the victim system.

INVESTIGATION SUMMARY

As a result of your SQL Server forensic investigation of the PROD-SQL05 server, you have determined that a remote user gained unauthorized database access through a brute-force attack. During this attack, several SQL Server logins names were targeted, eventually resulting in the compromise of the MSmith account. Unauthorized access was gained on 2008-08-31 15:27:09.50.

The MSmith login had a relatively high level of access within the database server through fixed-server role membership. Luckily, it did not have access to the CCProtect_Key that was used to encrypt the sensitive credit card information within the database.

After performing database reconnaissance and learning about the databases, users, and objects within the databases, the intruder utilizing the MSmith account initiated a search for passwords and encryption key–related information. No credit card information was disclosed during the incident, and no data within production tables was modified by the MSmith account. The unauthorized user's last known database access occurred at 2008/08/31 2008-08-31 15:31:35.24.

A summary of your investigation findings appears in Table 11.2.

Table 11.2 SQL Server Forensic Investigation Findings

Time	Event	Source
15:27:09.50	Brute-force attack initiated	IP: 192.168.1.20
15:31:35.24	Attacker gains unauthorized access to the database server	Login: MSmith
15:32:16.740 - 15:41:17.990	Database reconnaissance, including the viewing of data within the `orderhistory`, `sys.symmetric_keys`, and `sys.asymmetric_keys` views	Login: MSmith
15:32:41.320	The `sp_helpdb` procedure is executed	Login: MSmith
15:32:41.367	Temporary object `#09DE7BCC` is created within the `Tempdb` database associated with the `sp_helpdb` statement	Login: MSmith
15:36:34.42	Attacker reconnects to PROD-SQL05	Login: MSmith
15:38:15.31	Attacker reconnects to PROD-SQL05	Login: MSmith
15:41:55:060	`SELECT INTO` is statement executed, which initiates the copying of data from an unknown table into the `IllB3back` table within the `Master` database	Login: MSmith
15:41:55.060	`IllB3back` table is created within the `Master` database	Login: MSmith
15:43:16:570	Repeat `SELECT INTO` statement is executed using transaction ID 2724 but is aborted at 2008-08-31 15:43:16:570	Login: MSmith
15:43:23.74	Attacker reconnects to PROD-SQL05	Login: MSmith
15:45:15:437	Login password reset is attempted using `sp_password`	Login: MSmith
15:45:19:880	Login password reset is attempted using `sp_password`	Login: MSmith
15:45:43:797	Successful password reset of the MSmith password occurs using `sp_password`	Login: MSmith
15:45:43.800	MSmith account is updated	Login: MSmith
15:46:03.340	`EASYACCESS` login created	Login: MSmith

Note: All events within Table 11.2 occurred on 2008/08/31.

After reviewing the investigation findings, the client resets the password on the MSmith account compromised by the attacker and removes the EASYACCESS account created as a backdoor by the intruder. A stronger password policy is also implemented to help prevent a repeat occurrence of unauthorized access gained from a successful brute-force attack.

Index

D

E

F

G

H

I

N

O

P

Q

R

S

T

U

V

W

X