Fine-Grained Access to Network Services in Oracle Database 11g Release 1

Oracle allows access to external network services using several PL/SQL APIs (`UTL_TCP`, `UTL_SMTP`, `UTL_MAIL`, `UTL_HTTP` and `UTL_INADDR`), all of which are implemented using the TCP protocol. In previous versions of the database, access to external services was effectively an on/off switch based on whether a user was granted execute permissions on a specific package or not. Oracle 11g introduces fine grained access to network services using access control lists (ACL) in the XML DB repository, allowing control over which users access which network resources, regardless of package grants.

Access control lists can be created, amended and deleted in the XML DB repository directly using FTP or WebDav. In addition, Oracle provide the `DBMS_NETWORK_ACL_ADMIN` and `DBMS_NETWORK_ACL_UTILITY` packages to allow ACL management from PL/SQL. These APIs are the subject of this article.

Related articles.

- Fine-Grained Access to Network Services Enhancements in Oracle Database 12c Release 1

Create an Access Control List (ACL)

Access control lists are manipulated using the `DBMS_NETWORK_ACL_ADMIN` package. The `CREATE_ACL` procedure uses the following parameters to create a new ACL:

- acl - The name of the access control list XML file, generated relative to the "/sys/acls" directory in the XML DB Repository.
- description - A description of the ACL.
- principal - The first user account or role being granted or denied permissions. The text is case sensitive.
- is_grant - TRUE to grant, FALSE to deny the privilege.
- privilege - Use 'connect' for `UTL_TCP`, `UTL_SMTP`, `UTL_MAIL` and `UTL_HTTP` access. Use 'resolve' for `UTL_INADDR` name/IP resolution. The text is case sensitive.
- start_date - Default value NULL. When specified, the ACL will only be active on or after the specified date.
- end_date - An optional end date for the ACL.

The following code creates two test users to act as principals, then creates a new ACL.

```
CONN sys/password@db11g AS SYSDBA

CREATE USER test1 IDENTIFIED BY test1;
GRANT CONNECT TO test1;

CREATE USER test2 IDENTIFIED BY test2;
GRANT CONNECT TO test2;

BEGIN
  DBMS_NETWORK_ACL_ADMIN.create_acl (
    acl          => 'test_acl_file.xml',
    description  => 'A test of the ACL functionality',
    principal    => 'TEST1',
    is_grant     => TRUE,
    privilege    => 'connect',
    start_date   => SYSTIMESTAMP,
    end_date     => NULL);

  COMMIT;
END;
/
```

Once created, the ACL is visible in the "http://host:port/sys/acls/" directory.

Additional users or roles are added to the ACL using the `ADD_PRIVILEGE` procedure. Its parameter list is similar to the `CREATE_ACL` procedure, with the omission of the `DESCRIPTION` parameter and the addition of a `POSITION` parameter, which sets the order of precedence.

```
BEGIN
  DBMS_NETWORK_ACL_ADMIN.add_privilege (
    acl         => 'test_acl_file.xml',
    principal   => 'TEST2',
    is_grant    => FALSE,
    privilege   => 'connect',
    position    => NULL,
    start_date  => NULL,
    end_date    => NULL);

  COMMIT;
END;
/
```

Each principal is defined as a separate access control element (ACE), within the ACL. When multiple principles are defined, they are evaluated in order from top to bottom, with the last relevant reference used to define the privilege. This means a role that denies access to a resource can be granted to a user, but if the user is defined as a principal further down the file, that definition will override the role definition for that user. Use the `POSITION` parameter to ensure privileges are evaluated in order.

Privileges are removed using the `DELETE_PRIVILEGE` procedure. If the `IS_GRANT` or `PRIVILEGE` parameters are NULL, all grants or privileges for the ACL and principal are removed.

```
BEGIN
  DBMS_NETWORK_ACL_ADMIN.delete_privilege (
    acl         => 'test_acl_file.xml',
    principal   => 'TEST2',
    is_grant    => FALSE,
    privilege   => 'connect');

  COMMIT;
END;
/
```

ACLs are deleted using the `DROP_ACL` procedure.

```
BEGIN
  DBMS_NETWORK_ACL_ADMIN.drop_acl (
    acl         => 'test_acl_file.xml');

  COMMIT;
END;
/
```

Assign an ACL to a Network

Access control lists are assigned to networks using the `ASSIGN_ACL` procedure, whose parameters are listed below:

- acl - The name of the access control list XML file.
- host - The hostname, domain, IP address or subnet to be assigned. Hostnames are case sensitive, and wildcards are allowed for IP addresses and domains.
- lower_port - Defaults to NULL. Specifies the lower port range for the 'connect' privilege.
- upper_port - Defaults to NULL. If the lower_port is specified, and the upper_port is NULL, it is assumed the upper_port matches the lower_port.

The code below shows the ACL created previously being assigned to a specific IP address and a subnet.

```
BEGIN
  DBMS_NETWORK_ACL_ADMIN.assign_acl (
    acl         => 'test_acl_file.xml',
    host        => '192.168.2.3',
    lower_port  => 80,
    upper_port  => NULL);

  DBMS_NETWORK_ACL_ADMIN.assign_acl (
    acl         => 'test_acl_file.xml',
    host        => '10.1.10.*',
    lower_port  => NULL,
    upper_port  => NULL);

  COMMIT;
END;
/
```

Only one ACL can be assigned to a specific host and port-range combination. Assigning a new ACL to a specific host and port-range results in the deletion of the previous assignment. You must take care when making a new assignment that you are not opening ports that were closed by a previous ACL assignment, or you could be opening yourself to attack. When wildcard usage causes overlapping assignments, the most specific assignment will take precedence, so an ACL assigned to 192.168.2.3:80 takes precedence over once assigned to 192.168.2.* etc.

The `UNASSIGN_ACL` procedure allows you to manually drop ACL assignments. It uses the same parameter list as the `ASSIGN_ACL` procedure, with any NULL parameters acting as wildcards.

```
BEGIN
  DBMS_NETWORK_ACL_ADMIN.unassign_acl (
    acl         => 'test_acl_file.xml',
    host        => '192.168.2.3',
    lower_port  => 80,
    upper_port  => NULL);

  COMMIT;
END;
/
```

ACL Views

The `DBA_NETWORK_ACLS`, `DBA_NETWORK_ACL_PRIVILEGES` and `USER_NETWORK_ACL_PRIVILEGES` views display the current ACL settings. The expected output below assumes none of the delete/drop/unassign operations have been performed.

The `DBA_NETWORK_ACLS` view displays information about network and ACL assignments.

```
COLUMN host FORMAT A30
COLUMN acl FORMAT A30

SELECT host, lower_port, upper_port, acl
FROM   dba_network_acls;

HOST                           LOWER_PORT UPPER_PORT ACL
------------------------------ ---------- ---------- ------------------------------
10.1.10.*                                            /sys/acls/test_acl_file.xml
192.168.2.3                            80         80 /sys/acls/test_acl_file.xml

2 rows selected.

SQL>
```

The `DBA_NETWORK_ACL_PRIVILEGES` view displays information about privileges associated with the ACL.

```
COLUMN acl FORMAT A30
COLUMN principal FORMAT A30

SELECT acl,
       principal,
       privilege,
       is_grant,
       TO_CHAR(start_date, 'DD-MON-YYYY') AS start_date,
       TO_CHAR(end_date, 'DD-MON-YYYY') AS end_date
FROM   dba_network_acl_privileges;

ACL                            PRINCIPAL                      PRIVILE IS_GR START_DATE  END_DATE
------------------------------ ------------------------------ ------- ----- ----------- -----------
/sys/acls/test_acl_file.xml    TEST1                          connect true  02-APR-2008
/sys/acls/test_acl_file.xml    TEST2                          connect false

2 rows selected.

SQL>
```

The `USER_NETWORK_ACL_PRIVILEGES` view displays the current users network ACL settings.

```
CONN test1/test1@db11g

COLUMN host FORMAT A30

SELECT host, lower_port, upper_port, privilege, status
FROM   user_network_acl_privileges;

HOST                           LOWER_PORT UPPER_PORT PRIVILE STATUS
------------------------------ ---------- ---------- ------- -------
10.1.10.*                                            connect GRANTED
192.168.2.3                            80         80 connect GRANTED

2 rows selected.

SQL>

CONN test2/test2@db11g

COLUMN host FORMAT A30

SELECT host, lower_port, upper_port, privilege, status
FROM   user_network_acl_privileges;

HOST                           LOWER_PORT UPPER_PORT PRIVILE STATUS
------------------------------ ---------- ---------- ------- -------
10.1.10.*                                            connect DENIED
192.168.2.3                            80         80 connect DENIED

2 rows selected.

SQL>
```

Checking Privileges

In addition to the ACL views, privileges can be checked using the `CHECK_PRIVILEGE` and `CHECK_PRIVILEGE_ACLID` functions of the `DBMS_NETWORK_ACL_ADMIN` package.

```
CONN sys/password@db11g AS SYSDBA

SELECT DECODE(
         DBMS_NETWORK_ACL_ADMIN.check_privilege('test_acl_file.xml', 'TEST1', 'connect'),
         1, 'GRANTED', 0, 'DENIED', NULL) privilege
FROM dual;

PRIVILE
-------
GRANTED

1 row selected.

SQL>

COLUMN acl FORMAT A30
COLUMN host FORMAT A30

SELECT acl,
       host,
       DECODE(
         DBMS_NETWORK_ACL_ADMIN.check_privilege_aclid(aclid, 'TEST2', 'connect'),
         1, 'GRANTED', 0, 'DENIED', NULL) privilege
FROM   dba_network_acls;

PRIVILE
-------
DENIED

1 row selected.

SQL>
```

The `DBMS_NETWORK_ACL_UTILITY` package contains functions to help determine possible matching domains. The `DOMAINS` table function returns a collection of all possible references that may affect the specified host, domain, IP address or subnet, in order of precedence.

```
SELECT *
FROM   TABLE(DBMS_NETWORK_ACL_UTILITY.domains('oel5-11g.localdomain'));

COLUMN_VALUE
------------------------------
oel5-11g.localdomain
*.localdomain
*

3 rows selected.

SQL>

SELECT *
FROM   TABLE(DBMS_NETWORK_ACL_UTILITY.domains('192.168.2.3'));

COLUMN_VALUE
------------------------------
192.168.2.3
192.168.2.*
192.168.*
192.*
*

5 rows selected.

SQL>
```

The `DOMAIN_LEVEL` function returns the level of the specified host, domain, IP address or subnet.

```
SELECT DBMS_NETWORK_ACL_UTILITY.domain_level('oel5-11g.localdomain')
FROM   dual;

DBMS_NETWORK_ACL_UTILITY.DOMAIN_LEVEL('OEL5-11G.LOCALDOMAIN')
-------------------------------------------------------------
                                                            2

1 row selected.

SQL>

SELECT DBMS_NETWORK_ACL_UTILITY.domain_level('192.168.2.3')
FROM   dual;

DBMS_NETWORK_ACL_UTILITY.DOMAIN_LEVEL('192.168.2.3')
----------------------------------------------------
                                                   4

1 row selected.

SQL>
```

These functions may be useful for when querying the ACL views for possible matches to a specific host, domain, IP address or subnet.

```
SELECT host,
       lower_port,
       upper_port,
       acl,
       DECODE(
         DBMS_NETWORK_ACL_ADMIN.check_privilege_aclid(aclid,  'TEST1', 'connect'),
         1, 'GRANTED', 0, 'DENIED', null) PRIVILEGE
FROM   dba_network_acls
WHERE  host IN (SELECT *
                FROM   TABLE(DBMS_NETWORK_ACL_UTILITY.domains('10.1.10.191')))
ORDER BY
       DBMS_NETWORK_ACL_UTILITY.domain_level(host) desc, lower_port, upper_port;

HOST                           LOWER_PORT UPPER_PORT ACL                            PRIVILE
------------------------------ ---------- ---------- ------------------------------ -------
10.1.10.*                                            /sys/acls/test_acl_file.xml    GRANTED

1 row selected.

SQL>
```

Test the ACL

The TEST1 and TEST2 users have the ACL allowed and denied respectively. This means we can test the ACL functionality by comparing their responses to calls to external network services. The following code grants execute permission on the `UTL_HTTP` package to both users, then attempts to access a web page from each user.

```
CONN sys/password@db11g AS SYSDBA
GRANT EXECUTE ON UTL_HTTP TO test1, test2;

CONN test1/test1@db11g

DECLARE
  l_url            VARCHAR2(50) := 'http://192.168.2.3:80';
  l_http_request   UTL_HTTP.req;
  l_http_response  UTL_HTTP.resp;
BEGIN
  -- Make a HTTP request and get the response.
  l_http_request  := UTL_HTTP.begin_request(l_url);
  l_http_response := UTL_HTTP.get_response(l_http_request);
  UTL_HTTP.end_response(l_http_response);
END;
/

PL/SQL procedure successfully completed.

SQL>

CONN test2/test2@db11g

DECLARE
  l_url            VARCHAR2(50) := 'http://192.168.2.3:80';
  l_http_request   UTL_HTTP.req;
  l_http_response  UTL_HTTP.resp;
BEGIN
  -- Make a HTTP request and get the response.
  l_http_request  := UTL_HTTP.begin_request(l_url);
  l_http_response := UTL_HTTP.get_response(l_http_request);
  UTL_HTTP.end_response(l_http_response);
END;
/
DECLARE
*
ERROR at line 1:
ORA-29273: HTTP request failed
ORA-06512: at "SYS.UTL_HTTP", line 1029
ORA-24247: network access denied by access control list (ACL)
ORA-06512: at line 7


SQL>
```

From this we can see that the TEST1 user was able to access the web page, while the TEST2 user was denied access by the ACL.

The default action of the server is to deny access to external network service, as shown by the following test on a new user.

```
CONN sys/password@db11g AS SYSDBA

CREATE USER test3 IDENTIFIED BY test3;
GRANT CONNECT TO test3;
GRANT EXECUTE ON UTL_HTTP TO test3;

CONN test3/test3@db11g

DECLARE
  l_url            VARCHAR2(50) := 'http://192.168.2.3:80';
  l_http_request   UTL_HTTP.req;
  l_http_response  UTL_HTTP.resp;
BEGIN
  -- Make a HTTP request and get the response.
  l_http_request  := UTL_HTTP.begin_request(l_url);
  l_http_response := UTL_HTTP.get_response(l_http_request);
  UTL_HTTP.end_response(l_http_response);
END;
/
DECLARE
*
ERROR at line 1:
ORA-29273: HTTP request failed
ORA-06512: at "SYS.UTL_HTTP", line 1029
ORA-24247: network access denied by access control list (ACL)
ORA-06512: at line 7


SQL>
```

This may cause some confusion when upgrading databases that access external network services from 10g to 11g. In these situations, it will be necessary to implement suitable access control lists before your original functionality is possible.

Other Security Considerations

Pete Finnigan commented on his blog and in his security presentations about the fact that the ACLs are not tied to a specific package. This means opening a port on a server with the 'connect' privilege makes it accessible by `UTL_TCP`, `UTL_SMTP`, `UTL_MAIL` and `UTL_HTTP`. With this in mind there are some things to consider:

- The use of fine-grained access to network services is not an excuse to ignore basic security measures, like revoking unnecessary privileges on network service related packages.
- Control over the services you make available is possible by limiting access to the specific ports. If you only need HTTP access to port 80, specify the port rather than opening access to all ports on the server.
- Wildcards can be dangerous as you may be granting access to more servers that you should.
- You must protect your ACLs. If people can alter them, they become useless as a protection mechanism. Prevent direct access to the ACLs in the XML DB repository and make sure users don't have access to the management APIs.

Thanks to Pete Finnigan for his input.

Open ACL

From a security standpoint, it's not a good idea to allow complete network access from the database, but for testing features I sometimes find it useful to create an open ACL for an instance.

```
CONN / AS SYSDBA
BEGIN
  DBMS_NETWORK_ACL_ADMIN.create_acl (
    acl          => 'open_acl_file.xml',
    description  => 'A test of the ACL functionality',
    principal    => 'TEST',
    is_grant     => TRUE,
    privilege    => 'connect',
    start_date   => SYSTIMESTAMP,
    end_date     => NULL);

  DBMS_NETWORK_ACL_ADMIN.assign_acl (
    acl         => 'open_acl_file.xml',
    host        => '*',
    lower_port  => 1,
    upper_port  => 9999);

  COMMIT;
END;
/
```

For more information see:

- Managing Fine-Grained Access to External Network Services
- DBMS_NETWORK_ACL_ADMIN
- DBMS_NETWORK_ACL_UTILITY
- Fine-Grained Access to Network Services Enhancements in Oracle Database 12c Release 1

Hope this helps. Regards Tim...

Back to the Top.

13 comments, read/add them...