Securing and Auditing the iSeries Heritage Application

Villy G Madsen, ISP

August 30, 2002

GSEC Version 1.4b Option 1

Abstract

The IBM iSeries computers traces its heritage back to the System 38 (S/38) that was first introduced to the world in the late 1970s. Many of the applications written for the S/38 are still actively in use today. In today's IT environment, it is vital that all applications and their associated data be properly protected[1]. This paper describes how system level security can be added to a "Heritage Application". It then goes on in more detail to show how OS/400 system commands and SQL can be used to audit the system level security of a "Heritage Application".

Background

The IBM iSeries of computers is the latest scion of a line that started with the introduction of the IBM System 38 in 1978 [2,3]. The evolution of this earliest of Relational Database based Operating systems has been managed so well, that applications that were written for the first system will run on the most modern iSeries systems today with little or no modification.

On a positive note, this has resulted in the AS/400 (as most of us still call it) having one of the largest (if not the largest) suite of commercially available business applications of any computer platform. It also means that many of these applications have not been completely re-engineered since they were initially published.

These applications were developed in an age where the great majority of systems stood alone. PC's were still a dream, and the only user interface device was a terminal. As Michael Walsh[4] points out, security for many of these applications was and is based upon controlling user activities within the

[1] Flierman
[2] Sloan
[3] Lansa
[4] Walsh

application. Usually the user would be automatically connected to the application upon logon, and since his access to the data would be controlled by the application, there was no need for any additional security.

Today's world is very different. Network connectivity is now common place, and the standard user interface device is a PC. Commonly available decision support tools built upon such facilities as ODBC, DRDA, FTP, SQL etc[5] provide ready access to data on the AS/400

In order to maintain the Confidentiality and Integrity of the application data measures must be undertaken to control access to the data. At the same time, users must be able to take advantage of the advanced data access facilities that are available.

A number of means have been used for controlling access to critical data resources including:

- Use of exit routines[5].
- Use of database Triggers[6]
- Controlling access to tools and commands.
- Controlling access to data and program objects.

Exit routines and controlled access to tools and commands have their place in certain circumstances, but in my opinion should only be used as an adjunct to more data centric approaches.

Database Triggers are a relatively recent[*] addition to the AS/400 toolbox. They have a great potential for moving business rules from the application to the database. This has the advantage of enforcing business rules for all access to the application data. This approach has great potential for new applications, but its use in improving the security on existing applications is questionable. Retrofitting data triggers into an existing application requires an effort on par with a major re-write of the application.

Controlling access to data and program objects at the system level can greatly improve the security of your application data, while allowing authorized users to use modern decision support tools against the application data. Successful implementation is typically not a lengthy undertaking. It does require a thorough knowledge of OS/400 object level security, and a good understanding of how the application is organized.

[5] Susani

[6] IBM

[*] Recent by comparison with to the age of the iSeries / AS400 / S38 family.

The First Challenge - Securing the Application

Our goal is to provide increased security to an existing application. The application was initially installed with security that would allow any user full access to all application objects using system commands.

The security within the application itself enforces all business rules, controlling what fields and records an authorized user can access and/or change.

The challenge is to:

- Ensure that the only way to change the application database is through the application.
- Allow authorized users to use decision support tools to gain read access to the application database.
- Ensure that only authorized users will be able to access the application program objects.

The Tools

There are a number of security features available on the AS/400 that can be used to implement a solution including:

- Authorization Lists
- Object Security
- Adoption of Privilege

Authorization Lists

Authorization Lists are a means setting up security for a number of objects that all have the same security requirements. A number of objects are secured by an Authorization List. A number of users will be granted rights against the list. The rights granted to any user will apply to each of the objects that are members of the list. Other users may be granted the same, or different rights against the list.

This feature dramatically reduces our security administration effort, as the following example will illustrate.

We have 200 objects that need to be secured, and 300 users that require read access to these objects. There are two approaches that we can take.

We can grant each of the users individual access to each of the objects. This requires that we execute a GRTOBJAUT (grant object authority) command 200 times for each of the 300 users, for a total of 60,000 commands. Every time that a user requires access to a object, the system will have to search through the 300 security entries associated with that object to determine what level of access the user is allowed, resulting in significant processing overhead.

Using Authorization Lists, we grant each of the 300 users read access to the same Authorization List. Each of the 200 objects is then secured by the authorization list. We execute the CRTAUTL command (Create Authorization List) once and then ADDAUTLE (add authorization list entry) once for each user, and the GRTOBJAUT once for each object. Five hundred operations is still a daunting number, but we can easily automate this - and it can be implemented must faster than the 60,000 changes that would otherwise be needed. Each secured object will only have one security entry associated with it, rather than the 300 entries with the previous approach. When the system needs to verify that a user is allowed access to an object, it still needs to verify that the user is authorize against the list, not against the individual objects. The users' rights to access the members of the list is then cached for the term of the session, dramatically reducing processing overhead for access to additional objects.

Object Security

Object security allows us to control who has what kind access to an object. Access can be managed on a user by user basis, on a group (of users) by group basis, or through an Authorization List.

Adoption of Privilege

It is possible to configure a program object so that when a user executes it, he does so using the profile of the object's owner. This can be a powerful tool if used in conjunction with an application that has solid well thought-out application level security imbedded within it. The user needs to be authorized to access the program, but once the program is executing the rights of the program owner determine which data or additional program objects can be accessed.

Our Approach

This example assumes that the application has a single point of entry, a "Main menu", and that all other programs are called from this menu.

The following criteria must be met in order to meet our security requirements.

- The application main menu is sole point of access to the application.
- Data can only be created/changed/deleted using the application.
- Authorized users can use any tool available to access data.

- Only authorized users can change or delete the programs, menus etc associated with the application.

The Implementation Details

The approach that we have adopted to meet our requirements will involve making the following changes:

- Create a unique "application owner" User Id for this application - APPOWNER.
- "Sanitize" all libraries associated with the application so that:
  - Only objects associated with the application are present in each library.
  - All data objects are consolidated in one or more libraries.
  - Application program libraries do not contain data objects.
  - All objects belong to the application owner.
- Create an authorization list "APPRUN" that will be used to specify which users are allowed to run the application.
- Create an authorization list "APPREAD" that will be used to specify which users are allowed to read the application data files using decision support tools.
- Remove all access to application objects by other than the application owner.
- Add all authorized users to the APPRUN list with *USE (Read / Execute) privilege.
- Add only those users who require Decision Support access to the data to the APPREAD list with *USE privilege.
- Grant the APPREAD list access to all application data objects.
- Grant the APPRUN list access to the MAINMENU program.
- Change all the application programs so they run in the owner's profile.

How we wish that it was that simple! It likely won't be, but the list does represent the bulk of the changes required for many heritage applications. As any experienced iSeries developer will realize, additional changes will be required. In particular, any application specific Job Description, Printer and Display Definitions, and some File objects may need to be either secured against the APPRUN authorization list, have public granted use access against them. If the application supports batch jobs, then additional changes will be required. Then there are the applications that have functions that can only be run from QSECOFR… These cases are all manageable, but out of scope for this exercise.

Implementing security after the fact can be a bit of a trial. If one remembers to ensure that Job Description Objects are changed to cause job logs to be retained, then the trial can be a lot easier. A close perusal of the job logs can be a great help in determining why a application is not generating access violations.

In the words of the university professor, the rest of the proof is left as an exercise for the student.

The Second Challenge - The Audit

AS/400s are really a joy to audit. With few exceptions, system commands that generate output can create a database file that can be manipulated on the AS/400 using SQL, Query/400 or a custom written application. The file could also be exported to Excel or Access for further data reduction.

A simple example:

```
DSPUSRPRF USRPRF(*ALL) OUTPUT(*OUTFILE) OUTFILE(WORKLIB/USRPRFLST)
```

Will create a file that contains user profile information for all users on the system.

Executing the following query from SQL

```
SELECT UPUPRF, UPUSCL, UPPSOD, UPSPAU FROM WORKLIB/USRPRFLST
```

Will provide us with a list of all users on the system, their class (User, Programmer, Operator etc), when they last logged on, and what (if any) special privileges they have.

As is the norm for OS/400, field names are fully documented on the system. The easiest way to retrieve this information is through the use of the F4 (Prompt) key from within SQL or Query/400.

A few different queries will quickly retrieve such useful information as stale user id's, how many users have elevated privileges, and which ids have static passwords.

We can use this same approach to verify that all of the changes that we have implemented to secure our application are still in place, and fully operational.

Verifying the Application Owner ID

First of all, we will ensure that no one inherits the privileges of the Application Owner Id using the AS/400 Group Profile feature. The following command will list on the screen all user ids that are linked to APPOWNER. We would only expect to see one or more support Ids listed.

```
DSPUSRPRF USRPRF(APPOWNER) TYPE(*GRPMBR) OUTPUT(*)
```

This command when used with the *GRPMBR specification, is one of the few instances where we cannot route the results to a file.

Verify the Application Libraries

The second step is to build a file containing information about all of the application program objects.

The following command will create a new file containing information about all of the objects in the APPPGML1 library. If the output file already exists it will be over-written.

```
DSPOBJD OBJ(APPPMGL1/*ALL) OBJTYPE(*ALL) OUTPUT(*OUTFILE)
OUTFILE(WORKLIB/PGMLST)
```

If necessary, information about additional libraries can be added to this file as follows

```
DSPOBJD OBJ(APPPMGL2/*ALL) OBJTYPE(*ALL) OUTPUT(*OUTFILE)
OUTFILE(WORKLIB/PGMLST) OUTMBR(*FIRST *ADD)
```

The following query will list all of the records in the file, showing the library, program name and owner for each object in the library. The owner in each case should be APPOWNER.

```
SELECT ODLBNM, ODOBNM, ODOBOW FROM WORKLIB/PGMLST
```

In many cases, this list will be long enough that you really don't want to manually scan through it. The following command will provide you with a count of the number of objects by owner, type and library. A sample from an application where all of the program objects are contained in a single library looks like this:

```
OBJECT       LIBRARY     OBJECT       COUNT ( * )
OWNER                    TYPE
APPOWNER     APPLIB1     *CMD                  13
APPOWNER     APPLIB1     *DTAARA                3
APPOWNER     APPLIB1     *FILE                286
APPOWNER     APPLIB1     *MSGF                  1
```

This is a nice situation, ownership of all objects is correct. We would probably check to ensure that the *FILE and *DTAARA objects are actually part of the application and not misplaced application data objects.

The following shows an example of a situation where the application program files are stored in multiple libraries:

```
OBJECT       LIBRARY     OBJECT       COUNT ( * )
OWNER                    TYPE
APPOWNER     JCPLIB      *FILE                 90
APPOWNER     JCPLIB      *PGM                 357
APPOWNER     ORPLIB      *FILE                 59
APPOWNER     ORPLIB      *PGM                 332
APPOWNER     POPLIB      *FILE                 92
```

We will now perform the same process on the data libraries - the libraries that actually contain the applications database.

```
DSPOBJD OBJ(APPDATA/*ALL) OBJTYPE(*ALL) OUTPUT(*OUTFILE)
OUTFILE(WORKLIB/DATLST)
```

If there is more than one data library associated with the application, then the command can be repeated with the addition of the outmbr option as follows:

```
DSPOBJD OBJ(APPDATA/*ALL) OBJTYPE(*ALL) OUTPUT(*OUTFILE)
OUTFILE(WORKLIB/DATLST) OUTMBR(*FIRST *ADD)
```

Using the following SQL command to parse the data

```
SELECT ODOBOW,ODLBNM, ODOBTP, ODOBAT, COUNT(*) FROM WORKLIB/DATLST
GROUP BY ODLBNM,ODOBTP,ODOBOW, ODOBAT
```

gives us the following output:

```
OBJECT       LIBRARY     OBJECT    OBJECT        COUNT ( * )
OWNER                    TYPE      ATTRIBUTE
APPOWNER     APPDATA     *DTAARA                         313
APPOWNER     APPDATA     *DTADCT                           1
APPOWNER     APPDATA     *FILE     ICFF                    3
APPOWNER     APPDATA     *FILE     LF                  1,629
APPOWNER     APPDATA     *FILE     PF                    682
APPOWNER     APPDATA     *FILE     PRTF                    4
```

A happy result. We see that all of the objects are owned by APPOWNER, the only objects that might perhaps be better placed in a "program" library are the *FILE objects with the PRTF attribute, and the *JOBD. These are printer definition files and job description files respectively, and it could easily be argued that they could be located either in the data libraries or in the program libraries.

Check Program Object Permissions

The next step is to check the permissions associated with every one of the application objects. This unfortunately needs a little programming.

The following CL program will create a file containing the permission information for all of the objects in our application program libraries

```
PGM
             DCLF       FILE(WORKLIB/PGMLIST)
             DLTF       FILE(WORKLIB/OBJAUT)
             MONMSG     MSGID(CPF0000)
LOOP:
             RCVF
             MONMSG     MSGID(CPF0864) EXEC(GOTO CMDLBL(EOF))
             DSPOBJAUT  OBJ(&ODLBNM/&ODOBNM) OBJTYPE(&ODOBTP) +
                          OUTPUT(*OUTFILE) +
                          OUTFILE(WORKLIB/OBJAUT) OUTMBR(*FIRST
*ADD)
             GOTO       CMDLBL(LOOP)
```

This program will run for quite a while, and should be run in batch to minimize the impact upon other system users.

When our CL program has completed, we can use the following SQL query to determine if there is anything "interesting" to look at.

```
SELECT OAOWN, OAGRPN, OAUSR, OAANAM, OAOBJA, COUNT(*) FROM
WORKLIB/OBJAUT
GROUP BY OAOWN, OAGRPN, OAUSR, OATYPE, OAOBJA, OAANAM
```

The results below tell us that of the 1539 program objects, only one is accessible by anyone that doesn't belong to the APPOWNER group. Note that the PUBLIC permission is specifically set to *EXCLUDE. There is one object that is secured using an Authorization List. Based upon our security design, this is expected.

```
OWNER       GROUP       USER        AUTH. LIST  OBJECT      COUNT (*)
                                                AUTHORITY
APPOWNER                *PUBLIC     *NONE       *EXCLUDE    1,539
APPOWNER    APPOWNER    *GROUP      *NONE       *ALL        1,539
APPOWNER    APPOWNER    *GROUP      APPUSER     *ALL            1
```

We should now extract the details associated with this item using the following query.

```
SELECT OALIB, OANAME, OATYPE, OAOWN FROM WORKLIB/OBJAUT WHERE OAANAM =
'APPUSER'
```

giving us the following result:

```
LIBRARY     OBJECT      TYPE      OWNER
APPLIB1     UINIT       *PGM      APPOWNER
```

Since UINIT is the initial menu program for the application. Things are looking good. The next step is to check the contents of the APPUSER authorization list.

Issuing the following OS/400 command

```
DSPAUTL APPUSER
```

gives us

```
OBJECT . . . . . . . . :   APPUSER      OWNER  . . . . . . . :  APPOWNER
   LIBRARY  . . . . . :      QSYS       PRIMARY GROUP  . . . :   *NONE

              OBJECT    LIST
 USER        AUTHORITY  MGT
 APPOWNER    *ALL        X
 APPUSER1    *USE
 APPUSER2    *USE
 APPUSER3    *USE
```

Since the *USE authority on the AS/400 only gives the user execute privilege on a program object, we are happy!

Check Data Object Permissions

The next step is to repeat the previous steps, but this time for the data objects. Since the changes required in the previous steps will be obvious to anyone with AS/400 experience, we will assume that the file OBJAUT now contains the authorization information for the data libraries.

We then run the following SQL Query

```
SELECT OAOWN, OAGRPN, OAUSR, OAANAM, OAOBJA, COUNT(*) FROM
WORKLIB/OBJAUT
GROUP BY OAOWN, OAGRPN, OAUSR, OATYPE, OAOBJA, OAANAM
```

In this particular case, we get some results that are rather more interesting than our results with the program libraries.

```
OWNER        GROUP    USER     TYPE    AUTH.  OBJECT COUNT(*)     LINE
                                       LIST  AUTHORITY              #
APPOWNER              *PUBLIC *DTAARA *NONE  *EXCLUDE    313        1
APPOWNER              *PUBLIC *FILE   *NONE  *CHANGE       1        2
APPOWNER              *PUBLIC *FILE   *NONE  *EXCLUDE  2,305        3
APPOWNER              *PUBLIC *FILE   *NONE  *USE          2        4
APPOWNER              *PUBLIC *FILE   *NONE   USER DEF    10        5
APPOWNER              *PUBLIC *JOBD   *NONE  *USE          2        6
APPOWNER     APPOWNER *GROUP  *DTAARA *NONE  *ALL        313        7
APPOWNER     APPOWNER *GROUP  *FILE   APPREAD*ALL      2,307        8
APPOWNER     APPOWNER *GROUP  *FILE   APPREAD USER DEF    11        9
```

The first line of data indicates that we have 313 Data Area objects in our data libraries, and that there is no public access allowed. We have no problems with this.

The second line raises a flag. There is a file that anyone can change. We issue another query

```
SELECT OALIB, OANAME, OATYPE FROM WORKLIB/OBJAUT WHERE (OAUSR=
'*PUBLIC') AND
(OAOBJA='*CHANGE') AND (OATYPE = '*FILE')
```

And we get

```
LIBRARY     OBJECT      TYPE
SWCRLOD     TIVPOX1     *FILE
```

Now this is something that we want to add to our list of shortcomings!!

The next line (no. 3) tells us that 2,305 files have no public access. The next line however shows us that there are 2 files with Public *USE access. For a file this generally means read. We need to explore this. We change our SQL query slightly

```
SELECT OALIB, OANAME , OATYPE FROM WORKLIB/OBJAUT WHERE (OAUSR=
'*PUBLIC')
AND (OAOBJA='*USE') AND (OATYPE = '*FILE')
```

and get the following results

```
LIBRARY     OBJECT      TYPE
SWCRLOD     CONVEND     *FILE
SWCRLOD     GIMMIT      *FILE
```

This gives us more information for our final report, and then on to line 5. The USER DEF object authority indicates that these objects have authorities that don't fall into the standard definitions. So we change our query again…

```
SELECT OANAME, OAOPR, OAOMGT, OAEXS, OAREAD, OAADD, OAUPD,
OADLT FROM WORKLIB/OBJAUT WHERE (OAUSR= '*PUBLIC') AND (OAOBJA='USER
DEF')
AND (OATYPE = '*FILE')
```

```
OBJECT    OBJECT       OBJECT      OBJECT     READ ADD UPDATE DELETE
        OPERATIONAL  MANAGEMENT  EXISTENCE
QIDCTP02                                        X
QIDCTP10                                        X
QIDCTP20                                        X
QIDCTP21                                        X
QIDCTP25                                        X
QIDCTP30                                        X
QIDCTP31                                        X
QIDCTP51                                        X
QIDCTP52                                        X
QIDCTP53                                        X
```

These files have been set up giving the world read access to them. The OS/400 standard *USE authority gives both READ access and OBJECT OPERATIONAL access. The latter allows a user to look at the object's attributes.

In the event, this is incorrect and we can add this to our list of shortcomings.

Line 6 shows that a couple of Job Definitions exist, and the world has *USE access. This is normal and in fact necessary. The next line that causes some concern is line 9. We would investigate this discrepancy in the same manner as we did for Line 5.

Check for Programs

You will remember that programs on the AS/400 can be setup so that they run in the owner's profile. We want to ensure that there aren't any programs that can run in the APPOWNER profile that we aren't aware of. Thankfully OS/400 has a command that will list all programs that run in a particular user's profile

```
DSPPGMADP USRPRF(APPOWNER) OBJTYPE(*PGM) OUTPUT(*OUTFILE)
OUTFILE(WORKLIB/ADPTLIST)
```

and then running a SQL query against the result

```
SELECT COUNT(*),PALIB FROM WORKLIB/ADPTLST GROUP BY PALIB
```

giving us an unexpected result, and yet more entries in our list of shortcomings.

```
COUNT ( * )   LIBRARY
         22   GLCNVRT
      1,539   APPLIB1
```

Conclusion

Applying system level security to iSeries heritage applications data is an viable, and low cost approach to securing data. It dramatically reduces the probability of data loss occurring, while still allowing the use of modern Decision Support tools for adhoc data mining and data reduction.

The database centric nature of the iSeries computers provides some very significant benefits for the developer trying to add system level security to heritage applications, and for the auditor trying to verify the veracity of such security.

Glossary

Since iSeries people have their own language, an English to iSeries dictionary is probably in order.

When we say	What we mean is
AS/400	iSeries (the hardware)
DB2/400	The database, We don't usually talk about it - it just exists. It is the file system under OS/400.
DRDA	Distributed Relation Database Access
File	An object that usually contains a table, or a collection of tables sharing a common table specification.
Library	A single level directory; contains objects of any type except Library (QSYS is a special case)
OS/400	the native operating system on the AS/400
QSYS*	The high level library - think of root. or c:\. It is the only library that can contain library objects.

References

1. Flierman, Margaret. "Data Classification." 30 April 2001. http://rr.sans.org/securitybasics/class.php (8 September 2002)

2. Sloan, Jim. "History of the TAA Tools." URL: http://www.taatool.com/history.htm

3. Lansa, "Interview with Dr Frank Soltis, IBM AS/400 Chief Architect." LANSA User & Technical Conference 2000. URL: http://www.lansa.com/conference/2000_interview.htm (30 Aug 2002)

4. Walsh, Michael. "Some of the Dangers of Connecting your AS/400 to a Network." 25 September 2001. URL http://rr.sans.org/sysadmin/connect_AS400.php (30 Aug 2002)

5. Susani, Simi. " Brief Overview of AS/4000 Security Fundamentals" 13 April 2001. URL http://rr.sans.org/sysadmin/AS400.php (18 September 2002)

IBM Corporation, "Triggering automatic events in your database." iSeries DB2 Universal Database for iSeries Database Programming Version 5." URL: http://publib.boulder.ibm.com/html/as400/v5r1/ic2924/info/dbp/rbafomst.pdf (30 Aug 2002)

Welborn, Layton. "What About My AS/400? An Auditing How-To." 10 April 2001. URL: http://rr.sans.org/audit/AS400.php (30 Aug 2002)

IBM Corporation. "AS/400 Internet Security: Protecting Your AS/400 from HARM in the Internet."
http://www.redbooks.ibm.com/pubs/pdfs/redbooks/sg244929.pdf (30 Aug 2002)

IBM Corporation. "An Implementation Guide for AS/400 Security and Auditing: Including C2, Cryptography, Communications, and PC Connectivity"
http://www.redbooks.ibm.com/pubs/pdfs/redbooks/gg244200.pdf (30 Aug 2002)

I IBM Corporation. :"IBM iSeries Information Center."
http://publib.boulder.ibm.com/html/as400/v5r1/ic2924/index.htm (30 Aug 2002)

BM Corporation. "I Series Security Reference V5."
http://publib.boulder.ibm.com/html/as400/v5r1/ic2924/books/c4153025.pdf (30 Aug 2002)

IBM Corporation. "AS/400 Online library."
http://publib.boulder.ibm.com/pubs/html/as400/online/homeeng1.htm
(30 Aug 2002)