

COMP-7: Securing Your Swiss Cheese Environment

PAUL KOUFALIS

PRESIDENT
PROGRESSWIZ CONSULTING

Progresswiz Consulting

- Based in Montréal, Québec, Canada
- Providing technical consulting in Progress[®], Oracle, UNIX, Windows, MFG/PRO and more
- Specialized in performance tuning, system availability and business continuity planning
- ...and security of Progress-based systems

Who Are You and Why Are You Here?

What you should expect to take away from this session:

- Executive:
 - A better understanding of how vulnerable your systems might be
- Manager:
 - Simple suggestions for controlling your environments
- DBA/Sysadmin:
 - A few holes you might have missed

Who am I and Why am I here?

- More and more companies are starting to take security seriously
 - Sarbanes-Oxley/C198 Compliance
 - It's about time!
- Direct result: More and more security-related consulting
- The information in this ppt was extracted directly from my experiences

Before we start...

- This is a NINETY minute presentation
- Interruptions and discussions **REQUIRED**
 - Don't fall asleep please!
- I encourage you to disagree with me!
 - Argue, fight, call me names...

Agenda

- Security Overview
- Layer by layer from the inside out
 - DB, ABL, Operating System, Network, etc...
- Q&A

Security Overview

- Think of security as a castle with walls, moats and gates



Security Overview

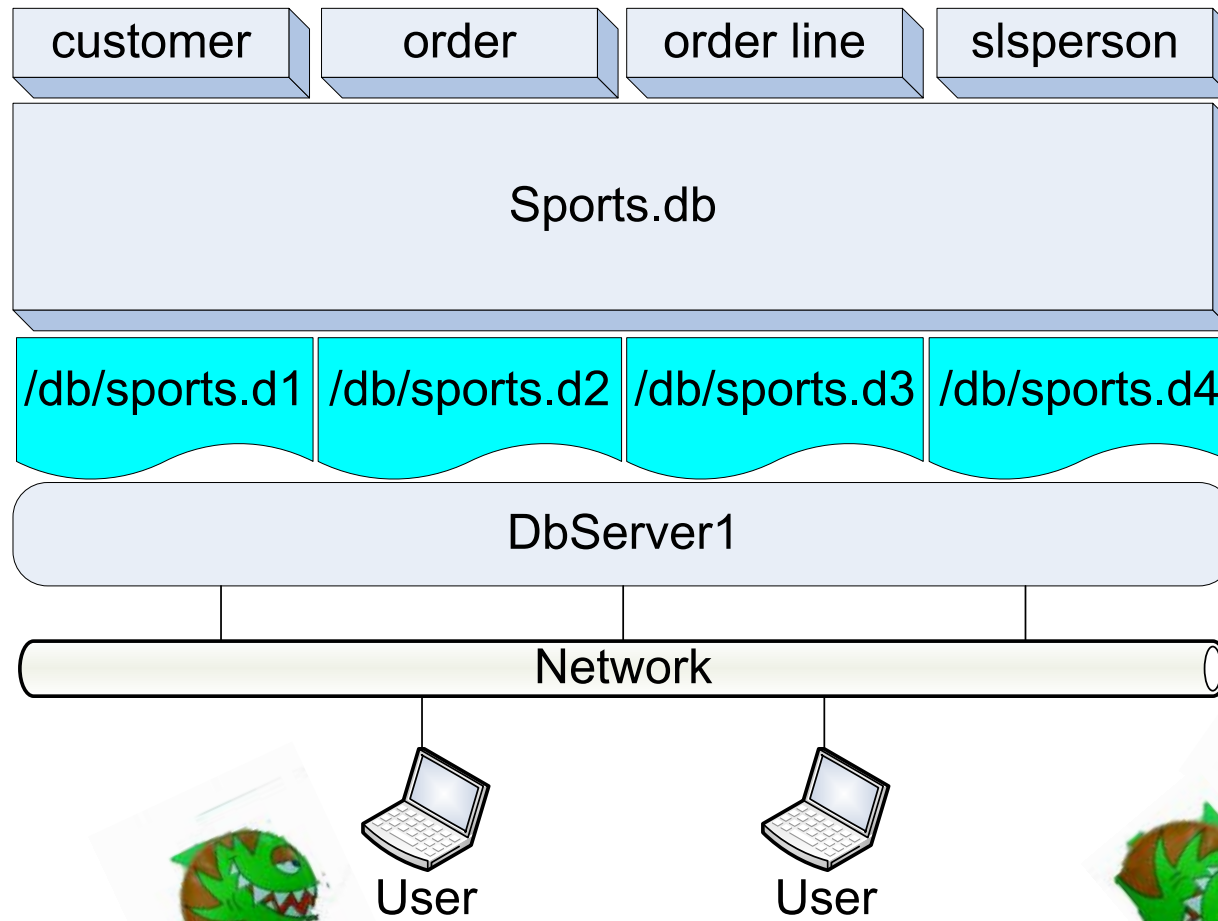
- Each wall is a layer of security
- BUT...the wall has to have gates
 - each gate is a potential security hole
- A successful security implementation requires a solid understanding of both the walls *and* the gates



Security Overview

- In an OpenEdge® database and application environment, the basic layers are:
 - The data itself
 - The database containing the data
 - The filesystem containing the DB's data files
 - The server hosting the filesystem
 - The network surrounding the server

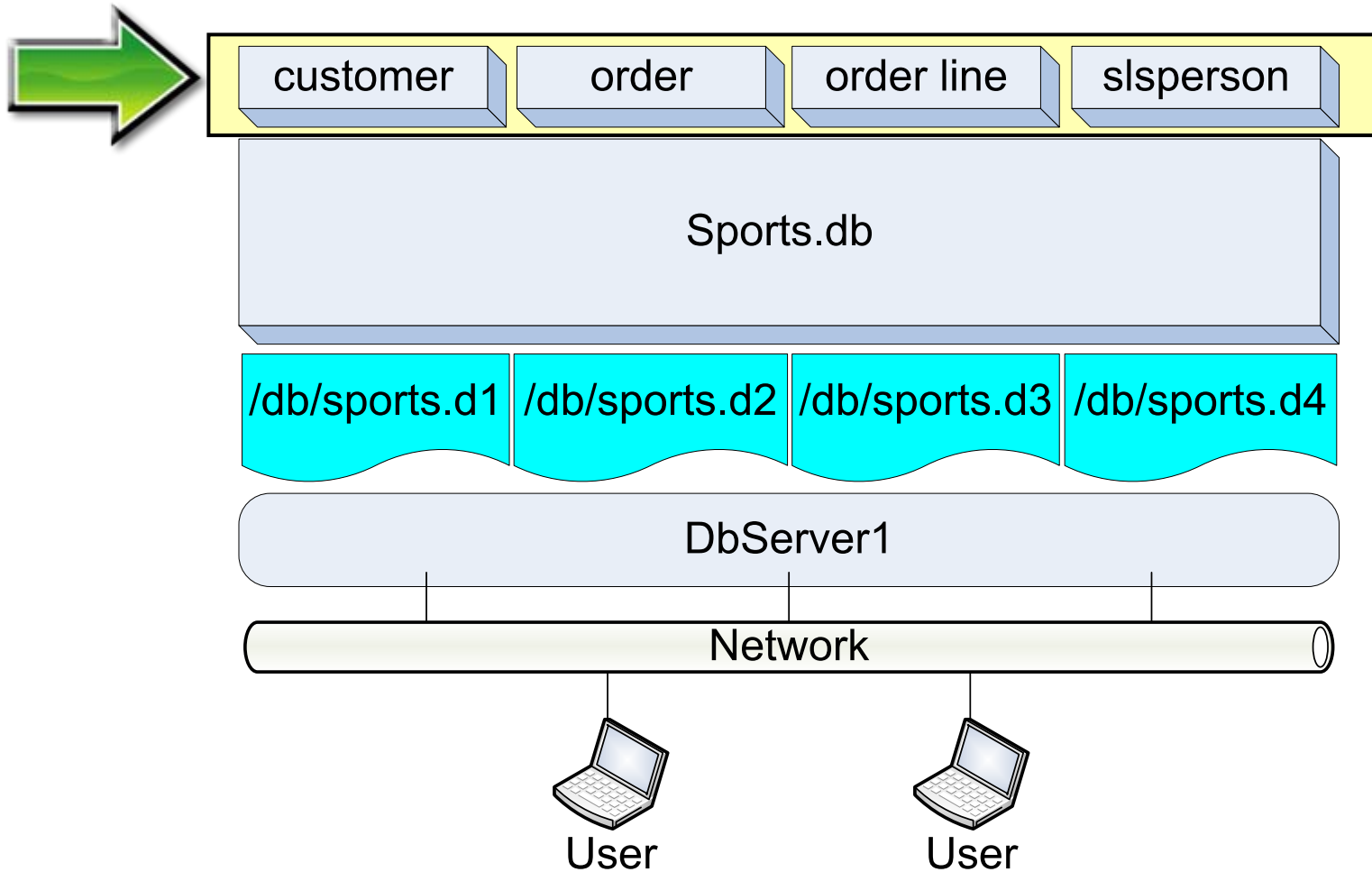
Security Overview



Security Overview – the Gates

- As we said earlier, gates let users through the security walls
- We'll start at the very heart of the castle and work our way out!

Security Layers



Data

- What are you trying to control?
 - Read?
 - Write?
 - Both?
- By default, OpenEdge grants full access to all data by all users
- SQL only grants access to owner


Data – Lock the Gates

- ABL access managed by _CAN-* fields in _FILE and _FIELD records
 - By default all _CAN-* fields = “*”
 - Data Security menu available in Data Admin
- SQL access controlled via GRANT
- Very difficult to manage data to application to user security

Data – Lock the Gates

- Security - disable blank user-id access
 - Really only adds a “!” to beginning of all _CAN-* fields
 - Not automatically added to new tables/fields
- Before 10.1A only applied to compile time
 - Limited use

Data – Lock the Gates

-  Enable runtime security
 - As of 10.1A
 - `_CAN-*` access validated at runtime

Data – Lock the Gates

- Stop refreshing your TEST environment with PROD data!



- Scramble the data first
- Better yet, create a standard test data set

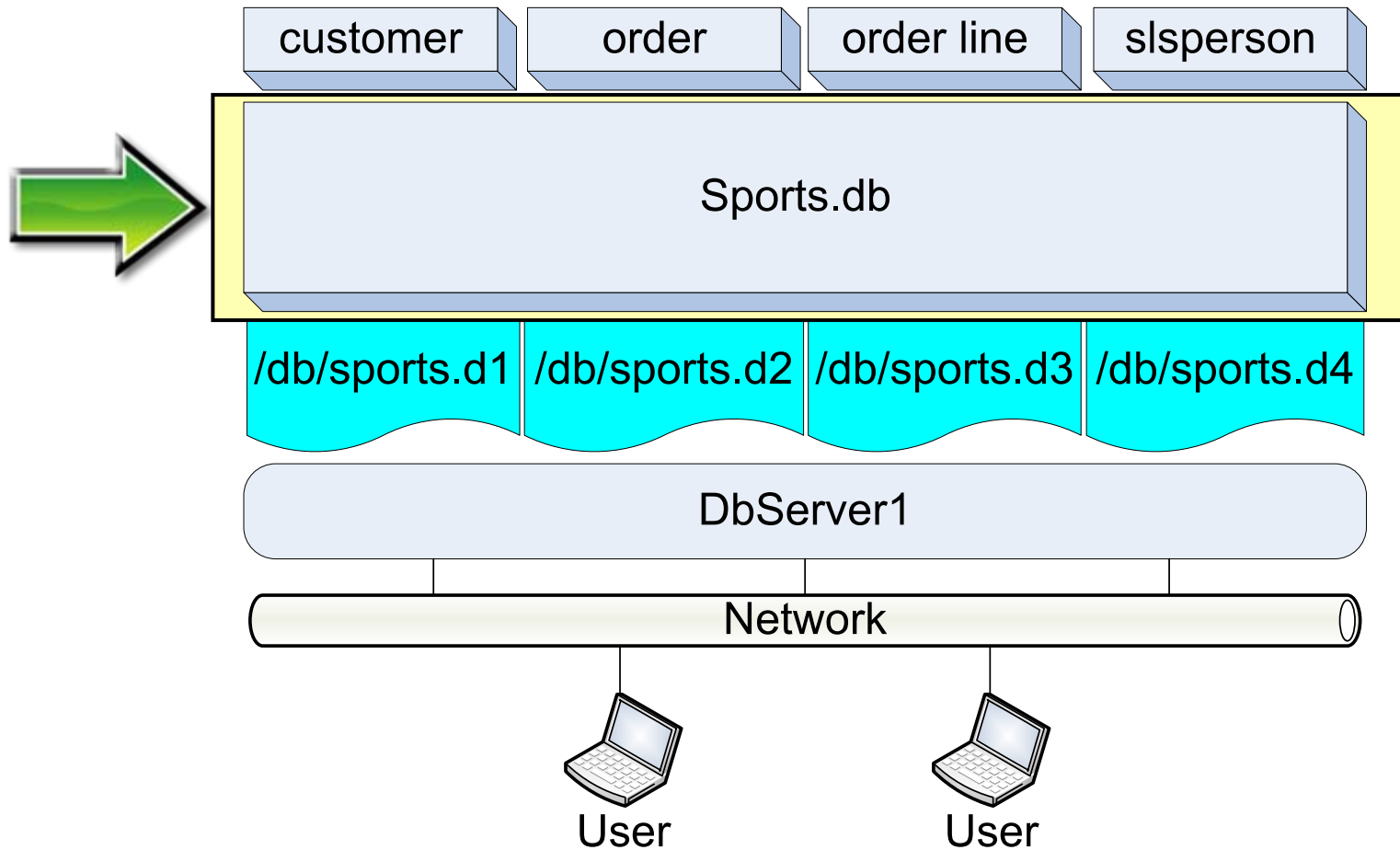
- Refresh + smart programmer + public company = easy insider trading information

- Or price list
- Or customer list
- Etc...

Conclusion - Data

- Best solution: 10.1A and runtime security
- Don't miss DB-14 "OpenEdge Run-Time Security Revealed"
 - Today at 15:30

Security Layers



The Database

- What accesses are we trying to control?
- What are the holes?
Don't peek ahead!

The Database

- Three big gates:
 - Local shared memory connection
 - Remote ABL client/server
 - Remote SQL client/server
- We'll talk about AppServer™ a little later...

The Database

- First level of defense: username/password
 - OpenEdge uses the _USER table
- The default implementation is a little weak
 - No complex passwords
 - No aging
 - You must implement everything in your application

The Database

- Who is using ONLY _USER security?
- Are you managing password age, complexity, etc?
- What about UNIX pwd? Windows pwd?
 - All independent?

The Database

- Reminder: OpenEdge Security is PUBLIC by default
 - Everyone has access to everything
- Two built-in controls for ABL access:
 - Admin – Security – Disable Blank User-ID Access
 - Admin – DB Options – Disable Blank UserID
 - 10.1A +
 - Someone at PSC was lacking in imagination when naming the second!



Security – Disable Blank UserID

- Changes default _CAN-* values from “*” to “!,*”
 - I.e. blank user cannot read/write any data
 - Except _User so that blank user can login
- BUT...
 - New tables/fields will get “*”
- YOU must manage security

DB Options – Disable Blank UserID

- Very simple: need –U –P at startup
- Otherwise connection is immediately refused
- No way to validate on-the-fly
- Depending on architecture could be easy or difficult to implement
 - Startup program collects username/pwd
 - ABL connect to DB

Locking the Gate



Username/pwd – Locking the Gate

- DB Options – No Blank UserID – DISABLED
- DB Options – Runtime Security - ENABLED
 - Remember: Does not exist before 10.1A
- Security – No Blank UserID – ENABLED
- Two security administrators: “Bob, Doug”
- Many “normal” users defined in _user




Security Administrator

- Metaschema security is convoluted
 - See Appendix A (not for the faint of heart)
- Remember to specify more than one security administrator!
 - One guy with 4 PC's doesn't count!

Security Administrator



Security Administrator

- Only SecAdmins can modify _CAN* fields
- Only SecAdmins can add/delete _USER records
- Normal users can modify their own _USER data
-  Interesting note: for SecAdmin to change a user's password he must delete and re-create the underlying _USER record

Managing Usernames and Passwords

Complex passwords gotchas:

- Spaces at end of password are ignored
- First character cannot be “?”
 - ABL interprets it as unknown
- Check aging at each login and by batch
- Cannot “disable” an account
 - Change password to some random key



Managing Usernames and Passwords

- No generic usernames
 - DBA, SecAdmin, operations, report
- Who has the password? Why? For what is the account used?
- Raise your hand if guilty:
 - ONE SQL username/pwd used by all users' ODBC DSNs for reporting

One Username/Pwd = Wide Open Door



DBRSTRCT

“Defines the access that all users, including you, have to your application database.”

- Hmm...interesting
- Look for it in “Managing ABL Applications”
 - Not in DB Admin Guide
- Haven’t played with this yet

DBRSTRCT

Database Restriction Utility

This utility lets you make IRREVOCABLE changes to EVERYONE's ability to modify or query database files. Before doing this, you should:


- make a copy of the database
- precompile all procedures

Deny (except for precompiled procedures):

1. Update access and/or query access to all current files.
2. Update access and/or query access to individual files.
3. Addition of new files to the database.

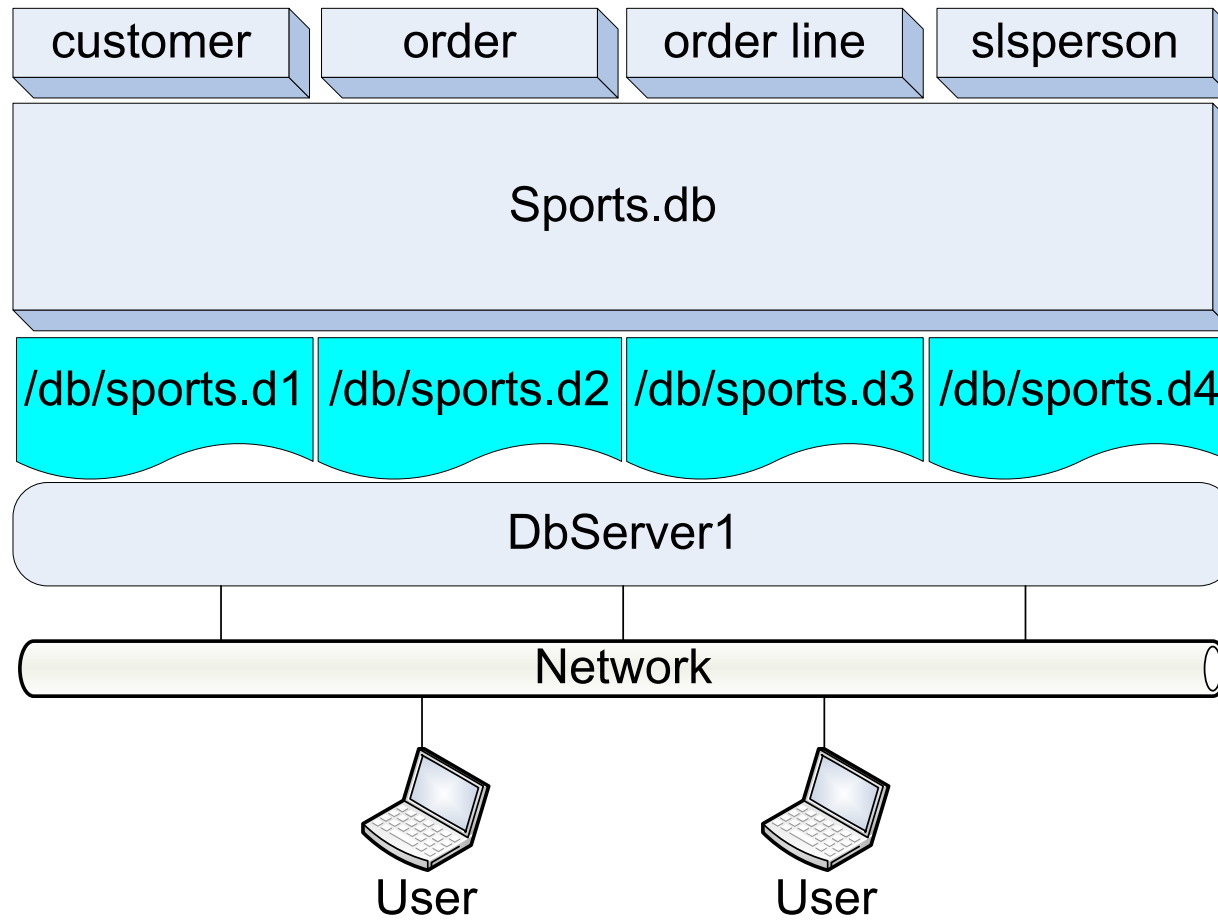
Selection: █

But but but...

- *“I’m running Application X on Progress 9.1E. I’d love to upgrade to 10.1B but it’s just not going to happen until next year.”*
- So....
 - No runtime security
 - Hard to use `_CAN*`
 - Some applications don’t even use `_USER`
-  Secure the next outer layer!



Remember our Walls and Gates




Secure the Environment

- Don't leave break-in tools lying around!!



Secure the Environment

- Don't leave break-in tools lying around
- Primary example: Full Dev key in production
\$DLC/progress.cfg
 - Everyone is running with a full dev license!
 - Your progress.cfg should be runtime only
 - Maybe Query/Results if absolutely necessary
-  Create a separate full.cfg for your compile needs

Environment – Locking the Gates

- General Rule: *If you have access to a full dev cfg then you do not have access to PROD databases.*
- *Will show how in multiple slides later on*
- Are you guilty?

Environment – Locking the Gates



- Use DBAUTHKEY/RCODEAUTHKEY
 - Restrict which R-Code can be run against DB
 - No key = no run
- WATCHOUT: There is an exploitable hole
 - No, I won't tell you what it is

Environment – Locking the Gates


- Be smart! The key shouldn't be stored with the lock!



Environment – Locking the Gates

- Lock down the PROPATH
- No “.” or relative path names
- No ad-hoc access to data by support personnel
 - *“...just set the P.O. status field to 0 and it will work...”*
- Developers forbidden from testing in production
 - Don't pretend like you don't know...

\$DLC – Locking the Gates

- Use root/administrator account to install OpenEdge products
-  ■ Install 4GL Dev and runtime licenses in separate CFG's
 - Make the 4GL Dev CFG readable by only those who need to read it!
- Leave the setuid bit

\$DLC – Locking the Gates




- Block what runtime users don't need
 - Many of the \$DLC procedure libraries
 - Example: `_edit.p`, `_admin.p`
 - Executables: `$DLC/bin/_proutil`, etc...
- No one should be able to modify files in \$DLC once it's installed
 - Exception: `$DLC/properties/*`
 - Give the DBA group write access to these files

\$PROPATH – Locking the Gates

- Every file and directory in the PROPATH should only be readable by the masses
 - Only deployment people should have write access

- No “.”; absolute directories only!

-  ■ Create a deploy group to manage code changes

```
-rw-rw-r-- koup    deploy    start.r  
-rwxrwxr-x deploy deploy    .
```

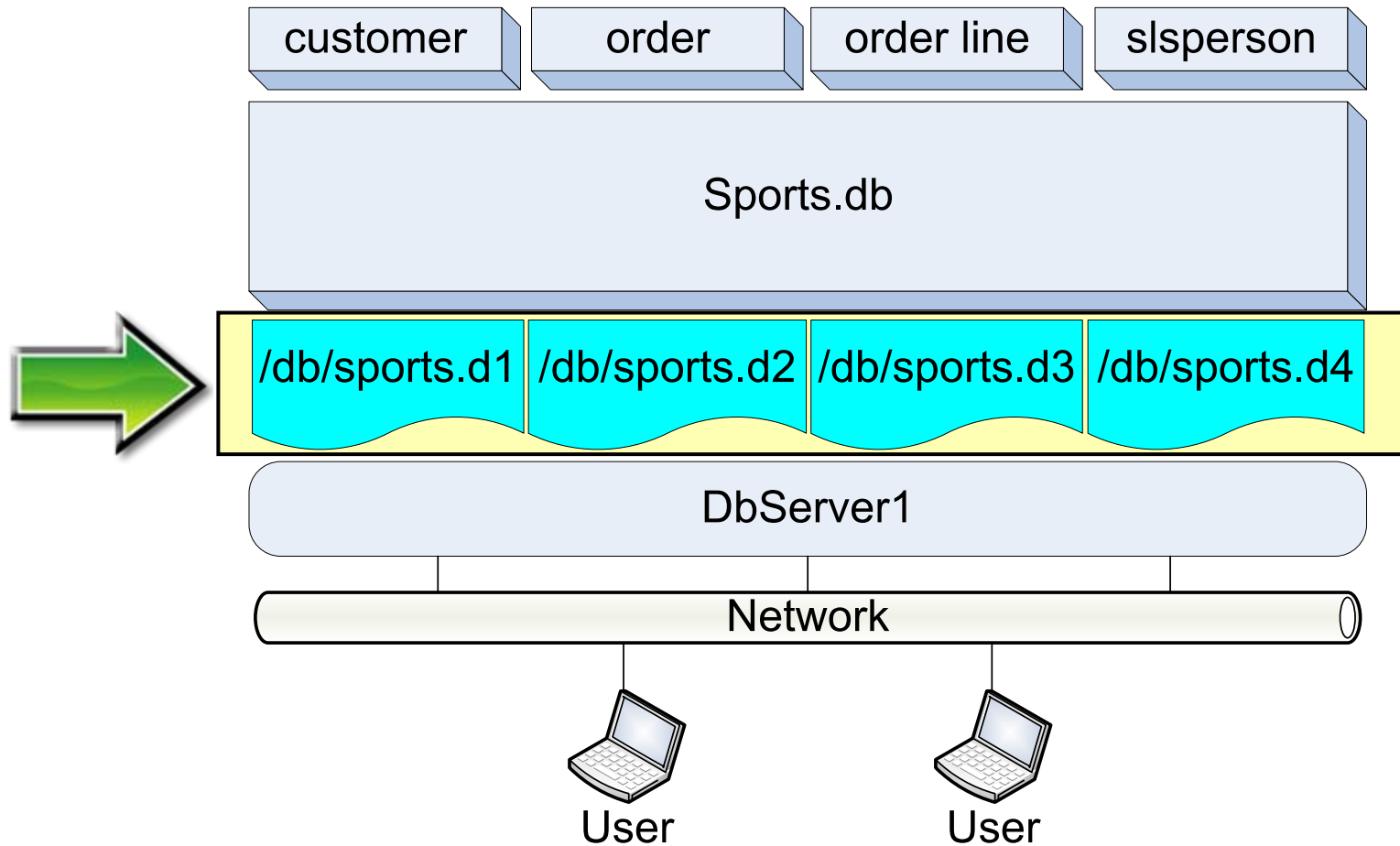
- Automate if you can

Break!

- Wake up!
- Stand up and stretch
- 2 minutes



Security Layers



DB Files

- Who has DB physical files that are protected from users?
 - What about the dir?

DB Files

- Data security useless without physical security
- With write access to the physical files:
 - Dump the blocks containing _CAN*
 - Physically change the data values:
 - “!,*” becomes “*,*”
 - Load the blocks back in to the DB

DB Files

- Lock down those files!



DB Files – Locking the Gates



- Only DBA group has write access
 - Files and directories!
- *Raison-d'être* of the setuid bit on the Progress executables
- Down side: No on-the-fly shared memory connections

DB Files – Locking the Gates

- Users connections restricted to:
 - Shared memory at startup
 - Client/Server
 - AppServer/Unified Broker

Other Files



■ Don't forget all the other files lying around!

- Client connection PF Files
- Server startup PF Files
- Scripts, scripts and more scripts!
 - For backup, AI rotation, etc...

OpenEdge and Other Processes


- No use of “root” or “administrator”
- Service account “dbserver” for databases
- Service account “appserver” for the AdminServer and all it’s children (AppServer, NameServer, WebSpeedTM, etc...)

OpenEdge and Other Processes



- Hint:
This is **not** good enough...



AdminServer/AppServer

- An AppServer™ is really just a client that runs ABL code
 - All the above environment suggestions apply
- PLUS...
-  Use the SESSION:EXPORT() function
 - List of programs that can be executed by Apsv
 - Can use wildcards




AdminServer/AppServer

-  Non-root **absolutely critical** with OpenEdge Management
 - Can run jobs on server
-  Use –admingroup parameter
 - Restrict access via Progress Explorer

Batch Queues

- Don't forget batch automation systems
 - QueueMaster, Appworx, etc...
- Must be secured
- No root/administrator

Shell Access

- Well this one's easy: none
-  ■ Use exec in .profile
-  ■ Change user's shell in /etc/passwd to restricted shell
- Restrict execution rights to ksh, csh, etc...
-  ■ Use “sudo” to run “OS-COMMAND” tasks

UNIX Username/Password

- Use all the standard password functions available:
 - Complex passwords
 - Password aging
 - Password history
- Users will hate you, I know...

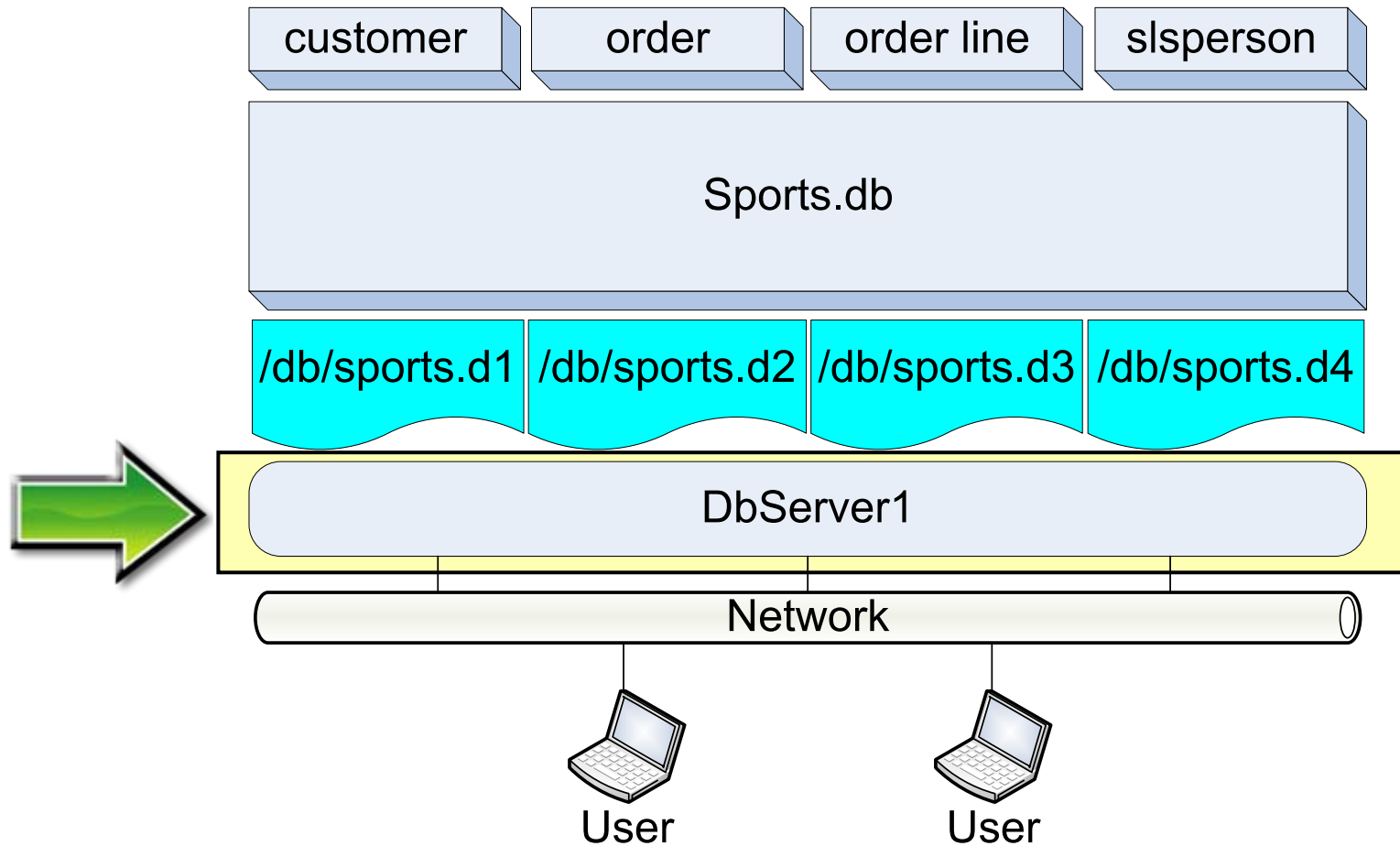
UNIX Username/Password

- Or maybe use a generic UNIX username with no password
 - Heresy!!!
- Wait! It can be done
- You are exec'ing the user straight into their _progres session anyways
- They better not have shell access!

Single Sign-On??

- It's coming...or is it here already?
- See *Identity Management* chapter of “Core Business Services” PDF
- Ask Michael Jacobs!

Security Layers



The Database Server

- So many holes by default
- Boot your box and run


```
# netstat -a | grep LISTEN
```

 - What are all those listeners?

The Database Server



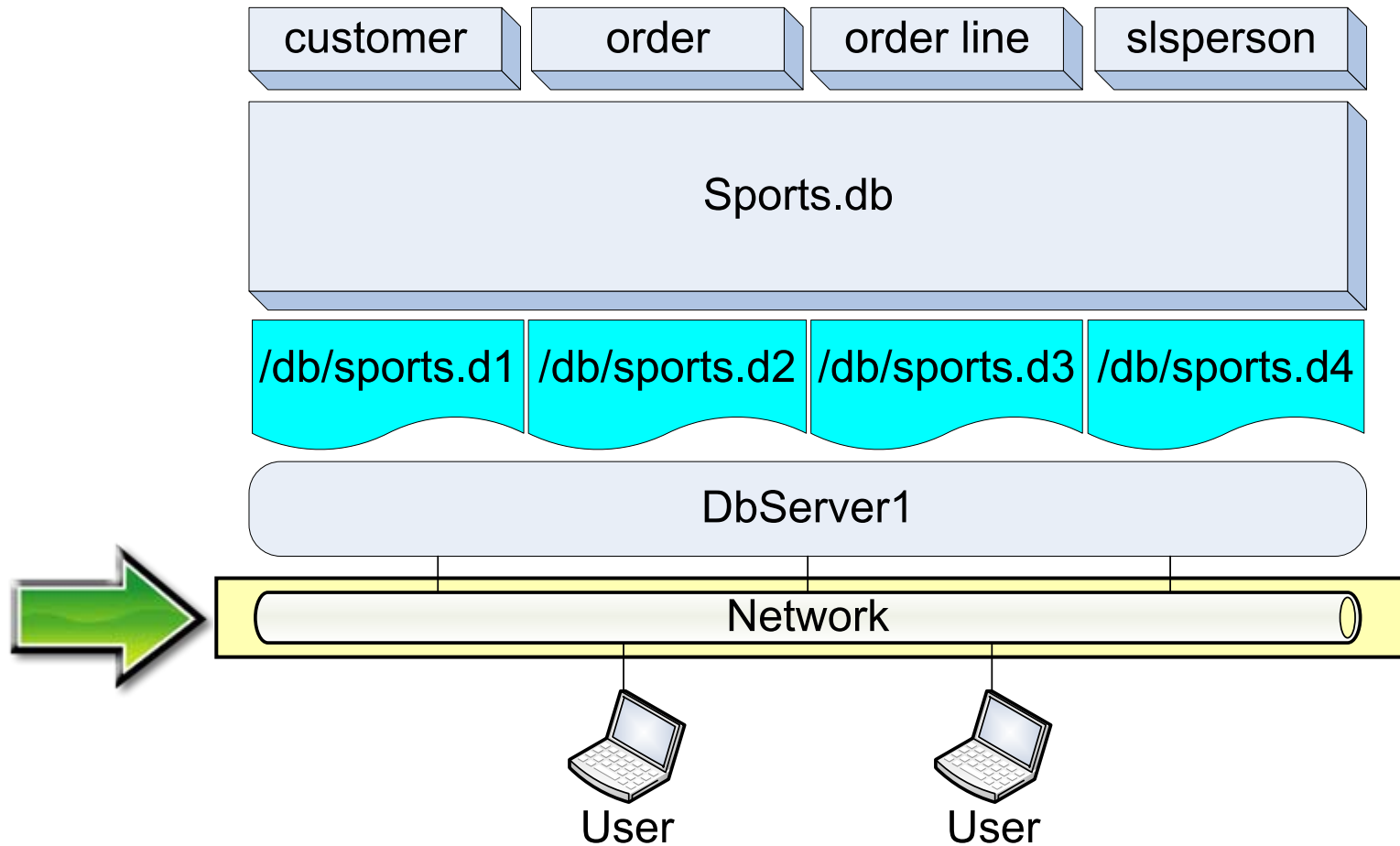
The Database Server

- Start with the basics:
 - telnet
 - ftp
 - rcp
 - rsh
- All unencrypted
-  ■ Switch to ssh, scp, sftp
 - Secure and encrypted

The Database Server

- Client/Server Database connections:
 - Use SSL
- File access via Samba or NFS
 - Often run as root
 - Easy to drop a .r via into a supposedly restricted PROPATH directory

Security Layers



Network and the Rest of the World

- Put DB Server behind firewall and restrict access
- You should know and control what is installed and where
 - ODBC
 - Full Dev OpenEdge
 - Runtime OpenEdge

Network and the Rest of the World

- **AGAIN: FULL DEV = NO PROD ACCESS**
 - Boxes with full 4GL licenses should **not** be able to access production database servers
 - Separate VLANs
- But they need to support production...
 - Second PC without full dev
 - Citrix



Network and the Rest of the World

- No regular users should have 4GL/ABL licenses
- Check and double check
 - I've seen desktop support install full PC ghost images with 4GL/ABL licenses for regular users
 - They don't know...

Network and the Rest of the World

- There should be no excuse for locking down the SQL side
 - Real usernames with real grants
- No generic “odbcuser”
 - Do you really want everyone to have read access to all data?

Network and the Rest of the World

- Lock it down!



Think Like a Firewall

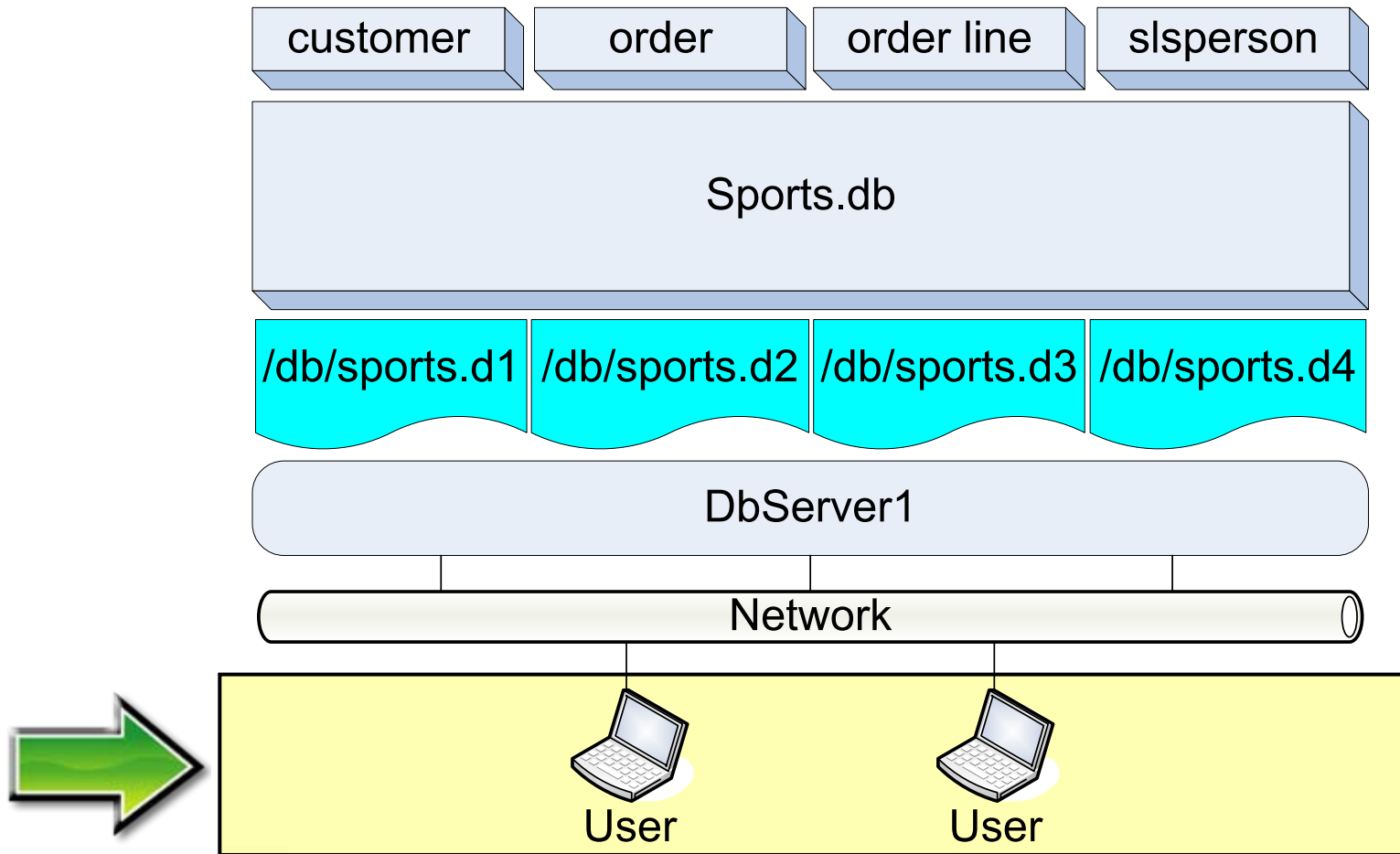
- Close everything
- Open only as needed
 - Control access to both sides of opening

Think Like a Firewall

- But not like in the movies!
 - Too easy to break in...



Security Layers



Users

- Life would be so much easier...



Users

- The most dangerous of all – your own people
- You don't pay them enough
- How segregated are your roles?
 - DBA?
 - Developer
 - QA?

How easy is it?

- Security firm randomly dropped USB keys on desks in an office
- Keys were infected
- Most employees said “Hey! Free stuff!”
 - Stuck it in their PC



Users – Locking the Gate

- No ad-hoc access
- No generic ODBC accounts
- Separation of roles
 - No one person can write, compile and promote code



- Table and field security
 - Runtime if OpenEdge 10.1A+
- DB username/pwd security
 - Disable blank userid access
 - Disable generic accounts
 - Develop your own complex pwd, aging, etc
- Designate at least two security administrators



- Secure the environment
 - No full.cfg
 - Lock down \$DLC
 - Lock down PROPATH directories
 - Lock down DB files
 - No shell access
- Compile with DBAUTHKEY

Recap



- No more root/administrator
- Use –admingroup on AdminServer
- Turn off any unneeded UNIX services
- Replace old standard services with shiny new encrypted versions



- Take inventory: Know who has what installed where!
- Segregate segregate segregate
 - Dev users see dev and prod users see prod

Remember our Castle Walls



Credits

- An extra special thanks to:
 - Michael Jacobs
 - Andrew McCarthy
 - James Lundy



Relevant Exchange Sessions

- COMP-1: Securing Your Web App
- DEV-4: OE in an LDAP World
- DB-8: Jump-Start OE Auditing
- DB-14: OE Runtime Security Revealed
- COMP-10: OE Mgmt + Replication
- DB-19: OE Authentication w/out _USER

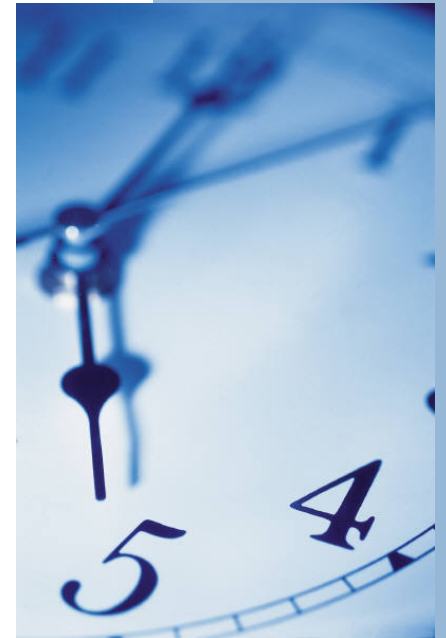
Questions?



Progresswiz Consulting

- Questions or comments? Send me an email:
pk@progresswiz.com
- And feel free to visit my website for this and other interesting information:
www.progresswiz.com

Thank you for your time



PROGRESS ***S O F T W A R E***

Appendix A – Understanding _CAN-*

- A deep and ugly look at _CAN* for schema security

_CAN-* Fields

■ _FILE

- _CAN-READ, _CAN-WRITE
- _CAN-DELETE, _CAN-CREATE
- _CAN-DUMP, _CAN-LOAD

■ _FIELD

- _CAN-READ, _CAN-WRITE

Example

- Look at _CAN-* for:
 - _FILE where _FILE-NAME = “_FILE”
 - _FILE where _FILE-NAME = “_FIELD”
 - _FILE where _FILE-NAME = “_USER”
 - _FIELD children of the above _FILEs
 - Other _FILE records (customer, order, etc...)
 - Other _FIELD records (name, address, etc...)

_FILE where _FILE-NAME <> “_USER”

```
FOR EACH _FILE WHERE _FILE-NAME <> “_USER”:  
    DISPLAY _CAN-READ _CAN-WRITE _CAN-DELETE  
           _CAN-CREATE _CAN-DUMP _CAN-LOAD.  
END.
```

- All CAN-* fields = “!,*”
- This means any non-blank user can modify or delete a table

_FIELD of _FILE

```
FIND _FILE WHERE _FILE-NAME = "_FILE".  
FOR EACH _FIELD OF _FILE  
  WHERE _FIELD-NAME BEGINS "_CAN":  
    DISPLAY _FIELD-NAME _CAN-READ _CAN-WRITE.  
END.
```

- All _CAN-READ = "*"
- All _CAN-WRITE = "Bob"
- Only Bob can create a table because only Bob can write in the _can-write field of the _FIELD rows

`_FILE` where `_FILE-NAME = "_USER"`

```
FOR EACH _FILE WHERE _FILE-NAME = "_USER":  
    DISPLAY _CAN-READ _CAN-WRITE _CAN-DELETE  
           _CAN-CREATE _CAN-DUMP _CAN-LOAD.  
END.
```

- `CAN-DELETE` and `CAN_CREATE` = "Bob"
 - Only SecAdmin can add or delete users
- The users themselves can change the data in their own `_USER` record.
 - Not managed by `_CAN*` field but rather built-in to ABL

Are we Lost Yet?

- To create or delete a table and fields, user needs the right to add/delete rows in `_FILE` and `_FIELD`.
 - We already said these are set to “!,*”
- But only the SecAdmin can modify the `_CAN-*` fields

Example

- UserID = "Paul"

```
FIND _FILE WHERE _FILE-NAME = "CUSTOMER".  
FIND _FIELD OF _FILE WHERE _FIELD-NAME =  
    "ADDRESS2".
```

```
UPDATE _FIELD.
```

- Result: Insufficient access privileges for field _can-read (233)

```
UPDATE _FIELD._LABEL.
```

- Result: Success

```
DELETE _FIELD
```

- Result: Success