



ActiveBase Security

Data Leakage Prevention and Compliance

Table of Contents

How is ActiveBase different from other Security solutions?	1
How is ActiveBase different from native DBMS Security features and tools?	1
How can we prevent DBAs from bypassing ActiveBase, connecting directly to the database? .	1
How can we determine which application will be effected by ActiveBase and which will bypass ActiveBase without going through it?.....	2
Why is AB*Security 'Blocking action' better and safer then blocking provided by other	3
How can AB*Security mask personal and sensitive information (SPI) in production without touching source code or databases?.....	3
What exactly do I need to install?	4
How large is the download and what does it include?.....	4
Do I need to change my Applications or Database?	4
What permissions do I need in order to install AB*Security?	5
What are the resources AB*Security consumes as it is installed on the database server?.....	5
What Databases does AB*Security support?	5
How do you upgrade AB*Security?	5
What is maximum number of rules that can be implemented without effecting propagation?	5
How can we prevent AB*Security from being a single point of failure?	6

How is ActiveBase different from other Security solutions?

ActiveBase is the most powerful database security solution – enabling to intercept unauthorized user requests, auditing them and applying 'on-motion' masking functions, encryption, column and row level restrictions. In addition, it is the only solution that can block a specific user request (without killing the session where the user opens a false support call...), while returning a notification back to the user, without touching source code or databases.

The masking and encryption/decryption of personal and sensitive information (SPI) is achieved by blocking or rewriting incoming SQL 'select list' columns with predefined functions, thus only a secured request is accessed to the database, returning masked/encrypted/decrypted or blocked results. For example, a SQL request such as 'select ... 'credit_card'...from ...' will be automatically rewritten to 'select ... 'xxxx-xxxx-xxxx-'|substr(credit_card,-4)...from...' returning just the last 4 digits with a 'xxxx-xxxx-xxxx-' mask. For encrypting SPI, ActiveBase uses standard Oracle database encryption packages.

How is ActiveBase different from native DBMS Security features and tools?

Native DBMS security tools cause performance penalty, AB Security can be centrally configured to support tens and hundreds of different databases on a single installation and enables complete separation of duties, managed even by a non-DBA.

How can we prevent DBAs from bypassing ActiveBase, connecting directly to the database?

1. Install ActiveBase on the database server, configure ActiveBase listener port to REPLACE and use the current Oracle listener port address, and assign the Oracle listener to use a hidden port to the Oracle listener port. Changing Oracle listener port is done in seconds. In addition, the Oracle listener can be configured to receive connections ONLY from ActiveBase listener port, blocking all other connections that try to bypass ActiveBase listener.

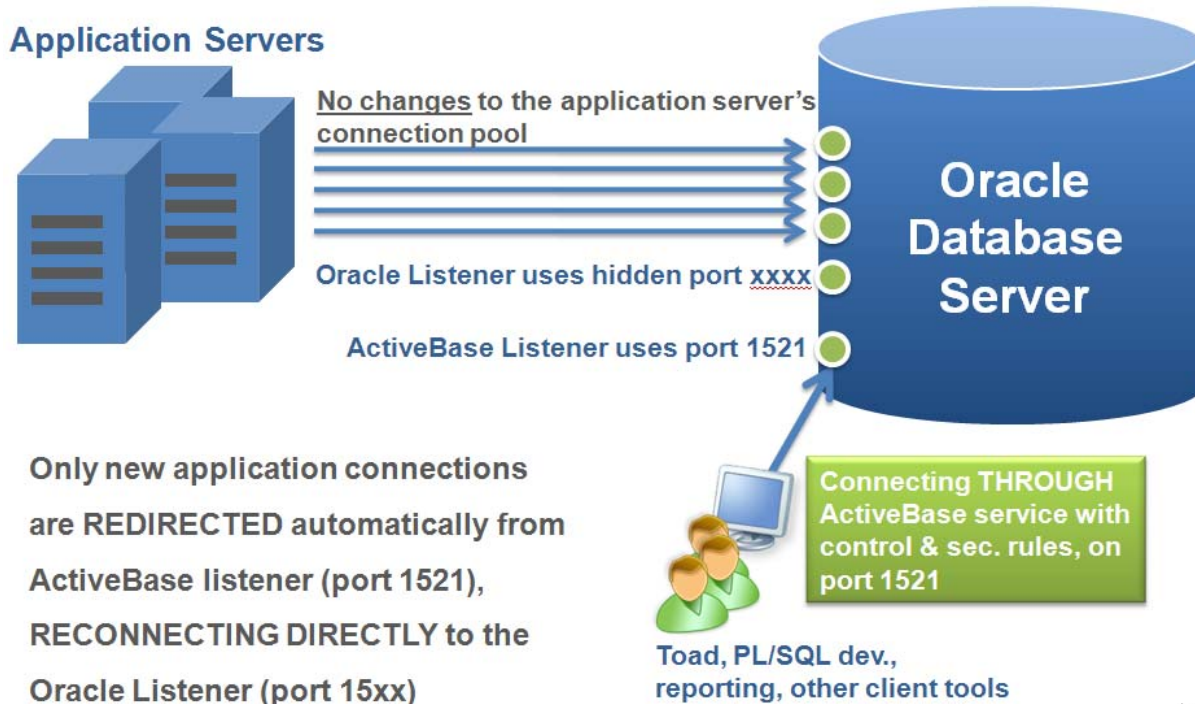


Diagram: ActiveBase listener is configured to use the existing Oracle listener port. Oracle listener is configured to use another hidden port – applying ActiveBase security without having to touch clients/applications

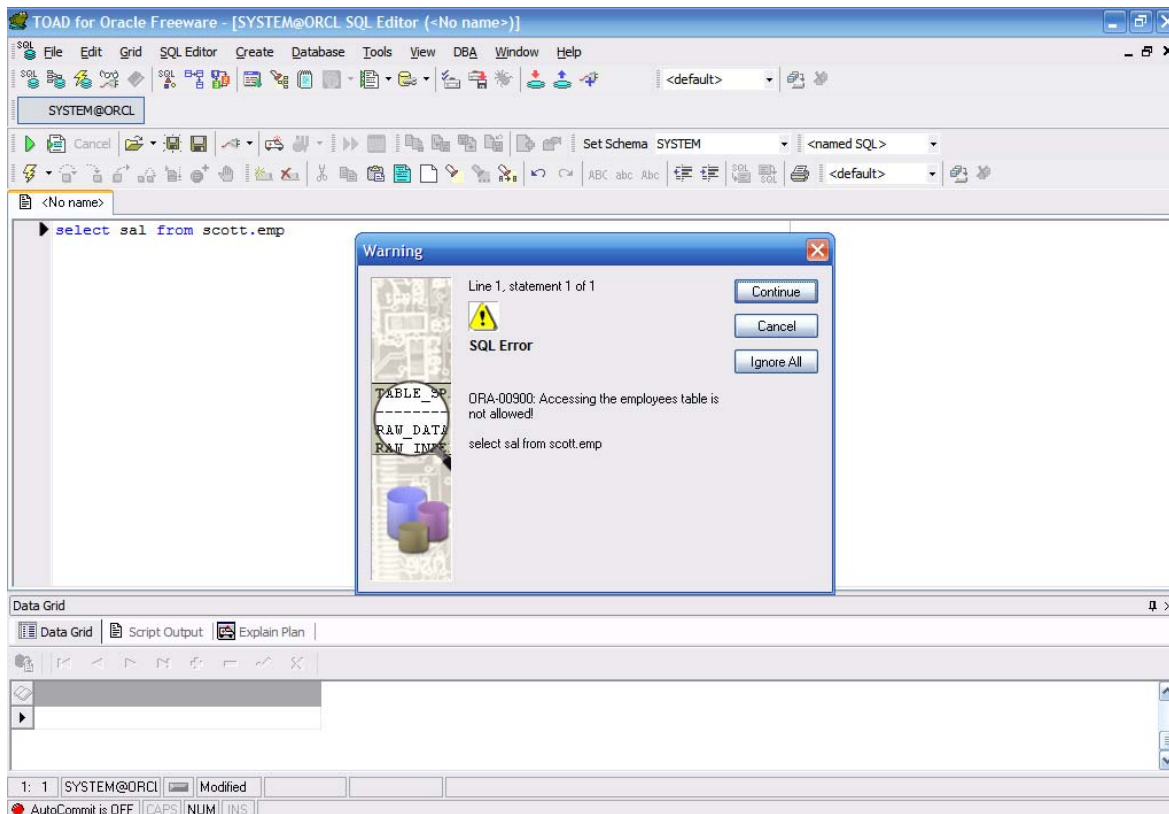
2. ActiveBase creates a logon trigger which controls users not connected through ActiveBase listener. In addition AB*Security automatically adds a prefix to the terminal name of all sessions connected through ActiveBase. This prefix is visible in the terminal column in v\$session view (e.g., administrators can identify all sessions NOT connected through ActiveBase by running 'select terminal from v\$session where terminal not like 'AB-%').

How can we determine which application will be effected by ActiveBase and which will bypass ActiveBase without going through it?

AB*Security has a unique connection-switching capability called 'bypass routing'. Bypass routing enables certain connections (based on a matrix of applications/hosts/program names/OS users) to completely bypass AB*Security completely – to directly connect to the database without passing through ActiveBase. For example, all DBA and development tools will be audited, masked and secured, while ETL connections will not connect through AB*Security while preventing any potential overhead.

Why is AB*Security 'Blocking action' better and safer than blocking provided by other solutions?

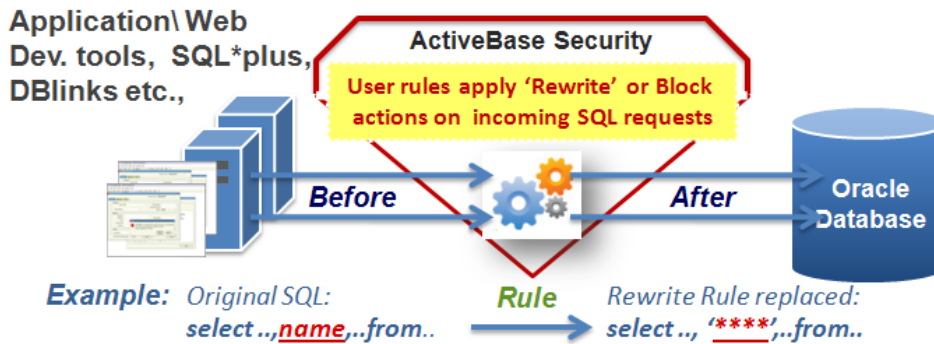
Only AB*Security enables to block a specific SQL requests in any application (2, 3 or n-tier applications) without touching application code, while returning a customized notification to the user (multi-language supported). Today, existing security solutions tear Application connections (when applying obtrusive TCP/IP reset, Packet dropping, tearing all sessions on the pooled connection) or kill user sessions (due to the risk of end-user opening false support calls). ActiveBase blocking action only blocks the specific offensive user request while returning a customized message to the user without connection tear or session kill.



Toad screen: ActiveBase blocks the user request while returning a customized message back to the user

How can AB*Security mask personal and sensitive information (SPI) in production without touching source code or databases?

AB*Security identifies information request retrieving SPI, and changes the SPI column with a mask function. It DOES NOT mask the result set returned by the user. For example, 'select ename from table emp' will be changed by the masking rule on-motion into: 'select substr(0,3, ename)||'****' which returns only the first two letters of the name followed by four '*' characters (e.g., eric is changed into 'er****').



Masking Rules:	Scrambling Rules:	Hiding Rules:	Blocking Rules:																														
Original SQL: <code>Select name,..from..</code>	Original SQL: <code>select name,..from..</code>	Original SQL: <code>select ..,name,..from..</code>	Original SQL: <code>select ..,name,..from..</code>																														
After Rule: <code>Select substr(name,1,2) '***'</code>	After Rule: <code>Select scrmbl(name)..</code>	After Rule: <code>select ..,,",..from..</code>	After Rule: Returned message: <code>You are not allowed to access this personal information!</code>																														
Result:	Result:	Result:																															
<table border="1"> <thead> <tr><th>Name</th><th>Name</th></tr> </thead> <tbody> <tr><td>Tiger</td><td>Ti***</td></tr> <tr><td>Nelson</td><td>Ne***</td></tr> <tr><td>Rogers</td><td>Ro***</td></tr> <tr><td>Rosen</td><td>Ro***</td></tr> </tbody> </table>	Name	Name	Tiger	Ti***	Nelson	Ne***	Rogers	Ro***	Rosen	Ro***	<table border="1"> <thead> <tr><th>Name</th><th>Name</th></tr> </thead> <tbody> <tr><td>Tiger</td><td>Bell</td></tr> <tr><td>Nelson</td><td>Cave</td></tr> <tr><td>Rogers</td><td>Lennon</td></tr> <tr><td>Rosen</td><td>Lenin</td></tr> </tbody> </table>	Name	Name	Tiger	Bell	Nelson	Cave	Rogers	Lennon	Rosen	Lenin	<table border="1"> <thead> <tr><th>Name</th><th>Name</th></tr> </thead> <tbody> <tr><td>Tiger</td><td></td></tr> <tr><td>Nelson</td><td></td></tr> <tr><td>Rogers</td><td></td></tr> <tr><td>Rosen</td><td></td></tr> </tbody> </table>	Name	Name	Tiger		Nelson		Rogers		Rosen		
Name	Name																																
Tiger	Ti***																																
Nelson	Ne***																																
Rogers	Ro***																																
Rosen	Ro***																																
Name	Name																																
Tiger	Bell																																
Nelson	Cave																																
Rogers	Lennon																																
Rosen	Lenin																																
Name	Name																																
Tiger																																	
Nelson																																	
Rogers																																	
Rosen																																	

Diagram: ActiveBase rewrite the incoming requests while masking/scrambling or hiding SPI column information requests

What exactly do I need to install?

In order to run AB*Security, you need to download and open a tar file - the AB*Security server software. It can be installed on a dedicated server or on the database server, or on one of the application servers with Unix HP/Solaris, AIX, Linux or Windows OS. For testing purposes, even a desktop is enough (though we would not recommend it for a production environment).

How large is the download and what does it include?

The download for the AB*Security server software ~35MB, dependant on your OS version, and includes the installation as well as documentation and a management console installation.

How long will it take to install and configure AB*Security on a single database?

Installation only takes about an hour. You will first install the server software (opening a tar), followed by a PC – administrator management consol.

Predefined rules will be available immediately, and you can extend them by defining custom rules

Do I need to change my Applications or Database?

No! AB*Security is completely transparent to both applications and databases. NO DATABASE CONFIGURATION CHANG EIS REQUIRED!.

The only configuration change required, is by changing the database listener port to another port (hidden) and configure ActiveBase listener port to listen to the primary port (e.g. Oracle port 1521). ActiveBase includes specific functionality to support this type of configuration by

enabling selective bypass to specific clients (clients are identified by an include/exclude list of program/host name and OSusers). This bypass enables all clients to connect to ActiveBase listener port, where specific clients will be routed with ActiveBase policies applied, and other applications will be routed directly to the hidden database listener, thus bypassing ActiveBase policies (e.g. ETL processes that do not require auditing or security policies).

What permissions do I need in order to install AB*Security?

You create user active, and open AB*Security tar file. It also comes with its OWN JVM installation, inside the installation tar. You 'start' AB*Security using running ./start from the installation directory. Edit the setPath file for specifying your specific installation directory. After installation, you define the Oracle databases that run on the server.

What are the resources AB*Security consumes as it is installed on the database server?

Overhead is negligible - about 1% CPU load, where propagation delay is about 150 microsecond per SQL statement. Keep note that AB*Security switch capability enables to define the specific modules, clients and IP's that will go through the rule processing (and overhead – 'use-rules' routing action), and which will automatically bypass AB*Security rules and connect directly to the Oracle listener ('direct' routing action).

A rule tree has been included, that enables ActiveBase to bypass all fast and efficient SQL statements with no delay, while only the long and inefficient SQL statement are caught and manipulated by the rules – accelerating them by x10-1000 times.

What Databases does AB*Security support?

AB*Security is currently available for Oracle 8 – 11g running on UNIX (HP, Sun or IBM AIX), Linux or Windows.

How do you upgrade AB*Security?

Simply stop the server process (using ./stop command within the activeknowledge directory), download the tar file into active directory and untar it. Upgrading does not delete your existing installation configurations. You also will need to uninstall ((control panel -> install/uninstall programs->ActiveBase),) and reinstall the Windows client to be able to connect and administer the new version.

What is maximum number of rules that can be implemented without effecting propagation?

No real limit or affect. AB*Security testing is done with up to 10000 rules.

Nevertheless based on our experience large installations do not have more than 30 - 50 active rules (although each rule fixes many different SQL statements – using 'search and replace' rewrite and regular expression tag place holders, partial text matching, partial execution plan matching etc.- resulting in a small BUT VERY EFFECTIVE number of rules in large production

implementations. Note that in massive on-line-transaction-processing (OLTP) application, we apply rules only on selected modules as well as use only regular expression matchers and rewrite actions for best speed.

How can we prevent AB*Security from being a single point of failure?

AB*Security uses several built-in mechanisms to ensure that it will never be a single point of failure. In addition, Oracle self fail-over mechanism (Failover and TAF – Transparent Application Failover) guarantees that in extreme circumstances all clients will automatically and immediately reconnect to another AB*Security listener or directly to the database server, completely bypassing AB*Security server. AB*Security can also be installed on high-availability cluster.

www.active-base.com

400-00101-303 | 02/09 | © 2009 ActiveBase, Ltd. All rights reserved. All other third-party trademarks are the property of their respective owners.

