WHITEPAPER

HVR Security Best Practices

hvr-software.com
info@hvr-software.com

Table of Contents

Executive Summary

ILLUSTRATION 1: BAD SECURITY PRACTICE

This paper serves as documentation for security when deploying HVR.

With the rise of Machine Learning and Artificial Intelligence (ML and AI), the value of data is increasing by the day. In order to get most value out of data, you consolidate, transform, integrate, and analyze data sets — typicaly in the cloud — to increase revenues, obtain competitive advantage, lower risk, or to come up with the next great invention. The value and privacy of this very important asset, your data, must be protected.

Security has multiple layers. For example, firewalls are generally not active between servers in the same data center because all servers operate behind a network firewall, protecting servers behind it against direct access from outside the data center. With the ongoing focus on data security, many organizations' IT departments have a set of security rules that any new software entering the data center will have to adhere to.

HVR provides real-time data integration to enable a continuous end-to-end data flow with minimum latency so that — considering the value of data — you can outperform your competition while using the technologies that best fit your (analytical) needs. HVR provides a rich set of features to keep your data secure. This document details these features for your consideration. To help you understand how HVR security best practices can be applied, first described in this document is the HVR architecture and runtime environment.

Architecture

HVR recommends the use of a distributed architecture with multiple installations of the software communicating to move data. Understanding this architecture is important because in most cases your data is most vulnerable when it is moving between systems.

In order to create a data flow (defined as a "Channel" in HVR), you must use an installation of the HVR software as the central point of control, the hub. The hub connects to a metadata repository stored in a relational database. The hub controls all activities related to the data flow.

HVR highly recommends the use of additional installations of the HVR software to act as agents, to:

- Distribute load to other systems.
- Maximize network performance using techniques like compression and large block transfer.
- Optimize efficiency by doing work close to the data.
- Improve security, the main topic of this paper.

Communication between hub and agents is over TCP/IP. Most deployments use an HVR agent on or very close to every source and every target.

HVR's architecture is flexible and modular. Although highly recommended, the use of agents is optional, and any installation of HVR can perform all tasks including extracting data from the source, operating as a hub, and delivering data into a destination. Most HVR deployments run a hub on a server with a User Interface running on a laptop or desktop remotely connected to the hub, but the UI can also be run on the hub server directly connecting to the local hub.

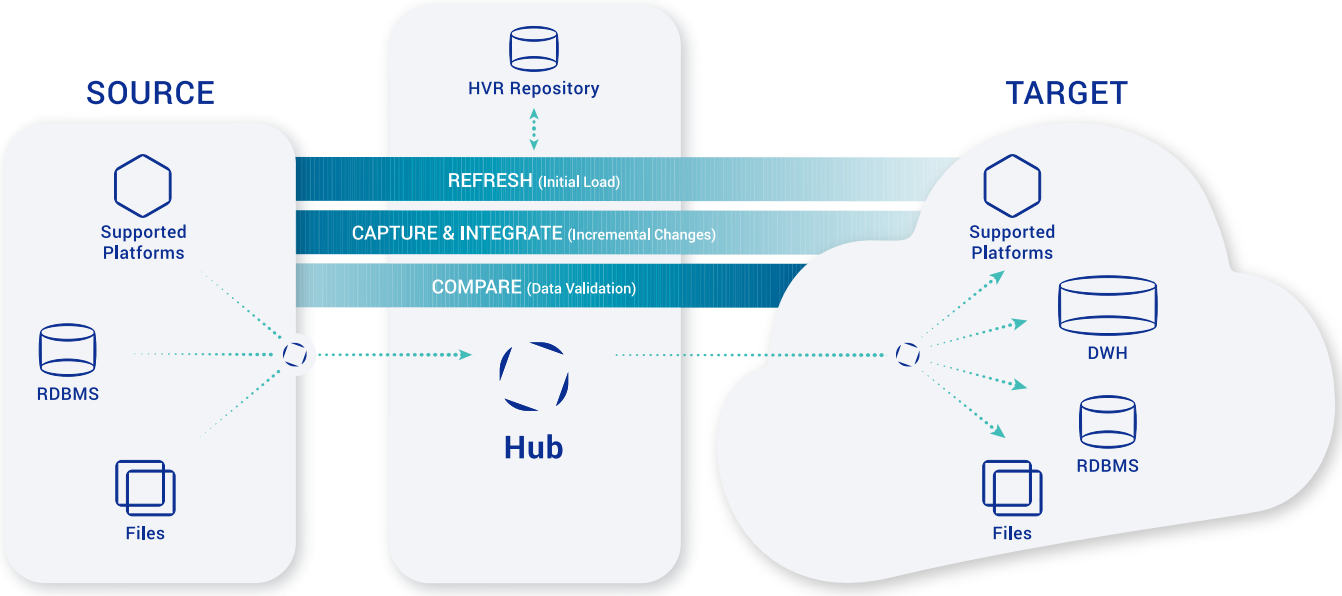

ILLUSTRATION 2: HVR'S RECOMMENDED ARCHITECTURE FEATURES THE USE OF AGENTS

Encryption Wallet

Starting with version 5.7, HVR supports an encryption wallet to secure passwords and data replicated by HVR when it is stored (temporarily) on disk. The use of a wallet is optional but highly recommended. HVR uses AES256 encryption.

HVR provides two levels of encryption when using the wallet:

- Secrets-only, encrypting all passwords HVR stores.
- Secrets and confidential data, encrypting, in addition to the passwords HVR stores, any data passing through HVR when it is stored on disk. This includes temporary files, checkpoints, transaction files, and verbose compare results when using event-based compare[1].

Access to the wallet, as well as the encryption level, are configured on the hub. HVR supports two types of wallets following the Public Key Cryptography Standards (PKCS):

1. Software-based (implementing the Public Key Cryptography Standard (PKCS) #12)
2. AWS KMS (Key Management Service as part of Amazon Web Services)

Irrespective of the type of wallet, there are three ways to open it:

- Auto-open, which facilitates restarts without operator intervention. Consider auto-open only for wallets external to the hub server. Note that access to AWS KMS can be configured using an access key/secret access key combination, or using a more secure instance profile when running on an AWS EC2 server.
- Password-protected, requiring an operator to provide a password upon restart. To minimize interruption, the HVR scheduler, in charge of starting/retrying jobs, will keep the wallet open for the duration of its session. All HVR processes that must encrypt/decrypt will request wallet access from the HVR scheduler.
- Through a plug-in you write. The plug-in may retrieve credentials to open the wallet from a remote server, enabling a secure setup without the need for operator intervention in case of an HVR restart.

The use of the encryption wallet results in slightly higher CPU utilization for HVR agents and the hub server. Without its use, including when using HVR versions prior to 5.7, passwords are stored using a proprietary obfuscation method, and (temporary) HVR files on disk containing your data are not encrypted by HVR.

Software Installation

The HVR hub and the HVR agent, as part of the HVR installation on a server, should use a separate operating system account with minimum privileges to perform its tasks. Required privileges are platform and use case dependent and documented in the requirements section for individual platforms in the User Guide. Creating a separate operating system user protects storage areas that HVR does not need access to, and can help you avoid inadvertent issues with file systems filling up. Also, with a separate username, account access can be audited more easily.

The HVR hub needs access to a database to store its metadata repository, a variety of database types are supported in order to store the repository. Always create a separate database user, schema, and/or database, and assign minimum privileges to the database user. Doing so avoids confusion about tables in a schema. This approach also protects the database from filling up unnecessarily and avoids unnecessary access to other data. In addition, access to the user database can be audited. At present, all users that need access to the hub will use the same database user to connect.

[1] When using Snowflake internal staging, HVR temporarily writes data to local files on the integration server before pushing data into staging. As of the writing of this paper, Snowflake does not support supplying encrypted files through this approach. Consider file system encryption on the integration server to mitigate the risk of unencrypted data on disk.

HVR is built to be resilient and can automatically recover from issues like database or system restarts, network glitches, or even temporary outages. In order to continue processing where it left off, HVR stores authentication information in the repository tables. This includes passwords for the database users that HVR uses to access the source and target databases, which are typically highly privileged users.

Use the HVR encryption wallet to encrypt the stored passwords, or, when relying on the obfuscation method HVR has always provided, consider using either database encryption or file system encryption as an additional safeguard for access credentials to your source and target systems.

Access and Authentication – Hub

In a large organization with many data flows, multiple users will need access to the hub. Make sure every user uses his/her own account for access to the HVR hub. This can be achieved out-of-the-box using any one of the following methods:

- LDAP access. HVR provides scripts to configure LDAP authentication. LDAP may require two-factor authentication which would then be required to obtain access to HVR. Also access levels can be enforced through LDAP groups.
- Pluggable Authentication Module (PAM) on Linux or Unix.
- Manage a separate registry of usernames and passwords through *hvrvalidpw*. Disadvantages of this method include that usernames must be managed separately, and there are no rules around password requirements or password expiration. You can consider implementing your own equivalent of hvrvalidpw by using the out-of-the-box approach as an example.

Also set up listener encryption on the hub using a unique encryption certificate to avoid communication with any HVR installation that does not have the correct matching public key.

Configuring encryption also provides protection against a man-in-the-middle attack. Certificate-based authentication from GUI to hub is not supported.

Access and Authentication – Agent

The hub accesses data in the replication end point through an agent — if one is used — for the purpose of continuous data integration. If connectivity is interrupted for whatever reason, then HVR will automatically retry and recover data integration where it left off. To achieve this, passwords are stored in the HVR repository and generated into the runtime environment. Should passwords change then such change has to be propagated through the system to enable automatic retries.

To secure agent authentication and communication:

- Enable password authentication at the agent, either using OS username password, LDAP (as long as the password is consistent), PAM (Pluggable Access Module) or by using your own username/password through *hvrvalidpw*.

- Set up certificate authentication using an access_conf.xml file (see https://www.hvr-software.com/docs/commands/hvrremotelistener) as a secondary layer of protection
- Use encryption on the HVR network communication agent to avoid data getting compromised, and to protect against a man-in-the-middle attack. Network encryption is particularly important when data crosses a Wide Area Network (WAN). The need for a unique public encryption certificate is an implicit way to authenticate the communication, **but encryption should never be used as the sole authentication method.**

Access and Authentication – Database

Most HVR data integration configurations include one or more databases, with some of the databases supporting OS-level authentication like Microsoft SQL Server or PostgreSQL. Generally, a username and password are needed to obtain access and of course the use of strong database user passwords is highly recommended.

Database authentication should be done through a database user with minimum database-level privileges. Different scenarios require different database privileges. For example, to be able to automatically replicate DDL such as create table the HVR user will need elevated privileges to enable select from a table that does not exist yet, and to enable supplemental logging. Consider, depending on the use case and scenario, whether some of the following options are acceptable to lower the HVR user's database-level privileges.

On the source database:

- Put a process in place for the DBA to add supplemental logging so HVR does not need a privilege to do it. HVR can still be used to generate the script with the "alter table add supplemental log data" statements to be executed by the DBA.
- Grant individual select privileges to the source database tables to avoid the need for a select any table privilege.
- Grant granular select access to dictionary tables to support the runtime needs.

On the target database:

- Consider if target tables can be created once (through HVR) following which the elevated privilege to create any tables can be limited. Note HVR may create tables it needs as part of regular processing (state tables, burst tables, external tables) but these get created in the default schema for the user connecting to the database.
- Limit explicit insert, update and delete privileges to only the tables being replicated.

HVR uses a database-dependent method to access the database, which may be the native database access mode such as TNS for an Oracle Database, or ODBC for databases like PostgreSQL, Teradata, Redshift, Snowflake and Greenplum. If HVR makes a remote connection (across a network) to the database then data may again be exposed. Use driver-specific options to secure this connection e.g. set up encryption.

Encryption

With the increase use of data integration scenarios that include the cloud as at least one side of the data integration, the use of network encryption can no longer be optional. Always use encryption when data crosses a public internet connection. With encryption enabled HVR defaults to AES256-level encryption.

The use of the HVR encryption wallet and network encryption are complementary. The use of one of the encryption features does not negate the need for the other.

During real-time data integration, HVR temporarily stores transaction files on the hub server. Use HVR's encryption wallet to encrypt data that is momentarily stored on disk, or as an alternative, consider using an encrypted file system for *HVR_CONFIG* to protect such data.

Connections to the data source and target are equally critical as connections across a WAN. If a database connection is possibly exposed then encrypt the connection. This requires a database-specific configuration.

To securely deliver data to or from a file location, use encrypted communication such as client-side KMS (Key Management System) encryption into S3, or server-side encryption with https connectivity.

Firewall Setup

Lock down firewalls as much as possible. In the HVR architecture, the hub always initiates the connection, and its placement determines the direction in which the firewall must be opened. The state-full nature of firewalls will allow the reverse connection without the need to open the firewall in the other direction.

For example, an on-premises HVR hub sending data into the cloud requires the firewall into the cloud to be opened, but there is no need to open the firewall into the on-premises data center. The opening in the firewall should be locked down to the originating IP address based on the host (the hub machine), and connections between systems in the same data center – on-premises or in an availability zone in the cloud – should use internal (VPC – Virtual Private Cloud) IP addresses rather than public ones.

Proxy

HVR can also be configured to run as a proxy. The proxy acts as a gatekeeper for connections to system(s) running the HVR agent, and avoids the need to directly expose the server running the agent in the firewall. Customers running the HVR hub in the cloud will commonly use a proxy in a DMZ (De-Militarized Zone) in the on-premises data center to route connections to systems behind the firewall. Consider the use of a proxy if more than one system in the same data center (or cloud availability zone) would otherwise have to be exposed through the firewall.

Logging / Auditing

Starting with version 5.6, HVR audits all interactions with the repository that may affect data replication. This includes any definition changes, (re-)initialization of a setup, and starting or stopping of jobs.

The audit information is stored in the HVR repository database and can be retrieved through a browser-based report (Event Audit Trail in the context menu on the hub), invoked from the HVR UI.

Conclusion

Data is one of your organizations' most important assets. Data breaches can be costly and embarrassing. Security is only as strong as its weakest link, so consider security especially when data is integrated between systems.

HVR supports a flexible and strong set of data security capabilities that you should use to keep your data secure. We hope you found this paper helpful in how to leverage numerous HVR product features to secure your data.

For more information on HVR and to request a trial and consulatation please refer to the links below:

hvr-software.com

info@hvr-software.com

hvr-software.com/resources/videos

twitter.com/hvr_software