SQL Server Security Best Practices

Securing Amazon RDS for SQL Server resources on AWS

First published August 28, 2024

Last updated August 28, 2024

Notices

Customers are responsible for making their own independent assessment of the information in this document. This document: (a) is for informational purposes only, (b) represents current AWS product offerings and practices, which are subject to change without notice, and (c) does not create any commitments or assurances from AWS and its affiliates, suppliers, or licensors. AWS products or services are provided “as is” without warranties, representations, or conditions of any kind, whether express or implied. The responsibilities and liabilities of AWS to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.

Contents

Abstract and introduction

Abstract

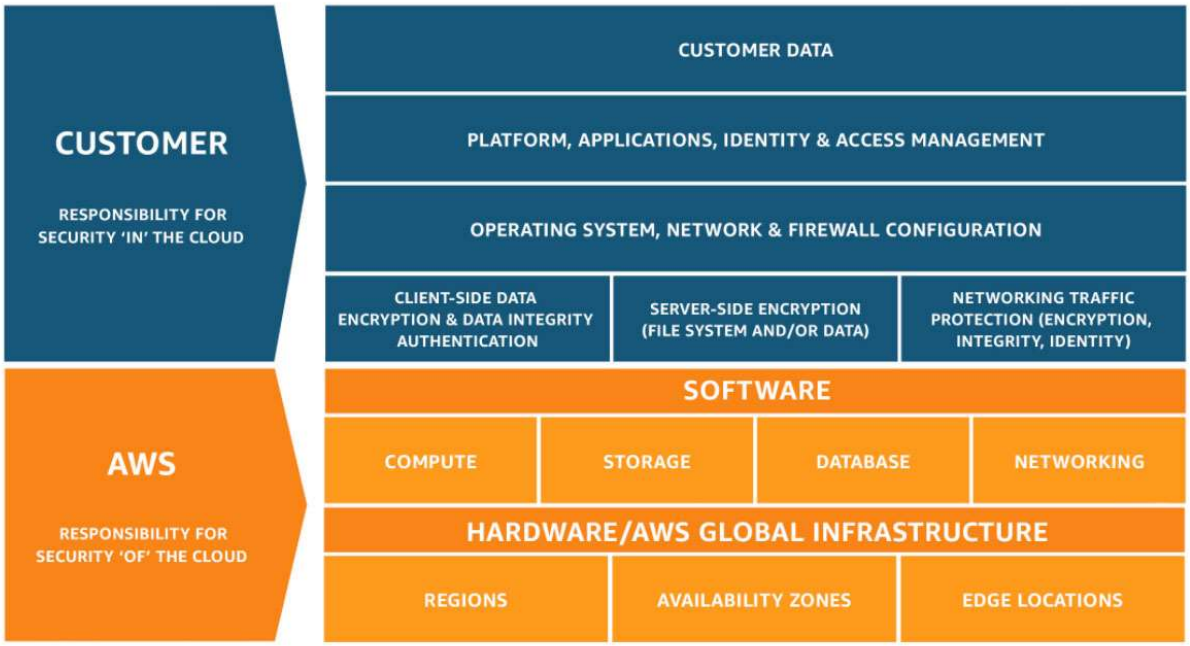
Amazon Relational Database Service (Amazon RDS) provides a managed platform on which customers can run a variety of relational databases, including MySQL, MariaDB, PostgreSQL, Microsoft SQL Server, Oracle, Amazon Aurora MySQL-Compatible Edition, and Amazon Aurora PostgreSQL-Compatible Edition. This whitepaper outlines best practices for securing SQL Server resources from data leaks, deletions, natural disasters and other calamities. The target audience for this whitepaper includes database administrators, enterprise architects, systems administrators, and developers who would like to run their database workloads on Amazon RDS.

Introduction

AWS has thousands of customers around the world running their SQL Server workloads on Amazon RDS. We also understand that data security is always of paramount importance for all our customers. Although hosting customer's data on Amazon RDS for SQL Server provides an improved security posture, we acknowledge this is not enough attention to customer security. This whitepaper provides a set of prescriptive authentication, authorization, and data encryption best practices specifically targeted for SQL Server workloads to help you strengthen the security of your SQL Server data. We also provide practical security recommendations to ensure that user databases deployed on RDS remain HIPAA eligible and PCI-DSS compliant, if required.

Shared responsibility model

Security *in* the cloud alongside the security of the cloud



AWS implements a shared responsibility model with regard to resources in the cloud. In short, AWS is responsible for the security of the cloud, and customers are responsible for security in the cloud.

Security of the cloud – AWS provides secure global facilities and infrastructure that spans regions, availability zones and edge locations. Layered on top of this infrastructure are compute, storage, database, and networking resources that serve as the foundation for every service offered by AWS. From the data centers, all the way up to the software that manages these services, AWS provides a secure cloud on which our customers can build secure and compliant applications.

Security in the cloud – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your organization's requirements, and applicable laws and regulations. AWS provides a set of features and services to help you secure your data. This paper helps you understand how to apply the shared responsibility model when using Amazon RDS.

IAM control over access

AWS Identity and Access Management (IAM) is the cornerstone of resource management on AWS. IAM defines the access permissions granted to entities to create, modify, and delete resources on AWS. For example, in order to create an RDS instance, one must first have the appropriate IAM permissions to do so.

Permissions within IAM are defined using policies. A policy is a document outlining a certain set of operations (create, modify, delete) that can be applied to certain services or resources. Once a policy is defined, that policy can be attached to an IAM identity (user, group, or role). These identities can be assumed by people directly accessing AWS, or by other resources on AWS. For example, it is common for an Amazon EC2 instance to assume a role that contains one or more policies. Let's say that one of those policies allows for writing files to an Amazon S3 bucket, but does not allow for reading from the same bucket. In this case, application code running on that EC2 instance can generate log files and write them to the specified S3 bucket, but the application will not have access to read or delete those files. Similarly, let's say that a person has an AWS CloudFormation template that defines an Amazon Aurora cluster. In order to provision that template, the user will need to assume a role associated with a policy that grants permissions to create the cluster.

It is of critical importance to secure IAM identities and use restrictive policies. In order to secure IAM identities, it is best practice to only use the AWS account root user to create other users and then lock away those root credentials. Likewise, when creating policies, it is best practice to only grant the very narrow and specific permissions required to accomplish the task at hand. For example, if you wish to create a policy that allows a user to modify a single Aurora cluster, you can restrict access to the specific Amazon Resource Name of that cluster. Or perhaps you would like to grant access to create new Aurora clusters but not to delete them. This can be controlled with a restrictive policy. This fine-grained access control is in addition to attaching the AmazonRDSDataFullAccess built-in policy to a given identity.

Networking

Network isolation

The main construct available to control network access to RDS resources is Amazon Virtual Private Cloud (Amazon VPC). RDS databases exist inside of a virtual private cloud (VPC). With a VPC, one can establish various subnets and define their connectivity to one another and the outside world. In nearly all circumstances, relational databases should be inaccessible, and they should not have access to network resources outside of the VPC. By adding your RDS databases to private subnets (no direct route to an internet gateway), devices outside of your VPC do not have direct access to your RDS database, and so long as there is no network address translation (NAT) device available to the subnet, your RDS databases do not have direct access to devices outside of your VPC.

VPCs also provide network access control lists (ACLs) to allow you to control network traffic at the subnet level. Traffic can be filtered based on protocol, port, and source.

Another component that allows you to control traffic are security groups. Security groups also allow you to control access at the network level, enabling traffic to be filtered based on protocol, port, and source. However, security groups also have the ability to filter another security group. For example, if you have a fleet of EC2 instances that need to communicate with your RDS instance, you can apply a security group named "ApplicationServers" to those EC2 instances and a second security group named "RdsResource" to

your RDS database. To allow the EC2 instances to communicate with the RDS instance, you would add a rule to the RdsResource security group that allows ingress from the ApplicationServers security group. Adding a rule is more convenient than limiting access based solely on source IP ranges and provides more flexibility should underlying IP resources change.

Now you have deployed your RDS resources in a private subnet, inaccessible from external network devices, you need to administratively connect to those RDS resources. In this case, the most common approach is to use a bastion host. A bastion host is a compute resource that has access to both RDS resources and the outside world. In this scenario, though the bastion host is publicly accessible, the security group associated with this host should only allow access from known IP ranges. In turn, that security group will be granted access to the RDS resources that are not publicly accessible. From a user's perspective, one can directly SSH to the bastion host and issue commands from the bastion host. Alternatively, a user can opt to create an SSH tunnel that will allow them to use local applications on their computer and tunnel the communication through the bastion host.

VPC flow logs

Once controls are put into place to manage network isolation, the next step is to audit network traffic. The primary tool to accomplish this is VPC flow logs. VPC flow logs identify network traffic flow. Flow logs can be useful in diagnosing overly restrictive network ACLs or security groups, or they can help you uncover gaps in your network controls that are allowing traffic that should be prohibited.

VPC flow logs have no impact on network performance and can be published to Amazon CloudWatch Logs, Amazon Simple Storage Service (Amazon S3), or Amazon Data Firehose. Each of these destinations provides a different benefit. Amazon S3 is a cost-effective way to capture VPC flow logs as part of an audit trail that might be infrequently accessed. CloudWatch Logs provides a common logging location, along with other log types, and it is straightforward to query from the AWS Management Console. Firehose is a great tool for ongoing, near real-time analysis of your VPC flow log records.

VPC endpoints for RDS API access

In some circumstances, you might run applications in your VPC that need the ability to provision, modify, or delete RDS resources. For example, you might run an EC2 instance that has assumed a role that gives it the ability to stop RDS instances in your development environment during off hours. Normally, the API call made to stop the RDS instance would need to travel on the public internet. If your EC2 instance is on a private subnet without an NAT gateway it will not be able to connect to the RDS API to issue the command to stop the instance. By enabling AWS PrivateLink, your EC2 instance can now issue the stop command without being on a public subnet or using an NAT gateway. Traffic remains on the Amazon network and neither your EC2 instance nor the relevant RDS instance is exposed to the public internet.

Encryption

AWS KMS

One of the most basic necessities of security when working with databases is to encrypt your data. To encrypt data requires encryption keys. The management of those keys is a critical consideration when securing your databases. Fortunately, the AWS Key Management Service (AWS KMS) is specifically designed to create, manage, and rotate encryption keys, and it seamlessly integrates with the RDS platform. AWS KMS allows users to create symmetric and asymmetric keys for encryption and decryption, as well as for HMAC authentication. AWS KMS is not specific to RDS and is integrated with a variety of other AWS services. This integration allows for a unified key management system across AWS. AWS KMS offers keys that are scoped to a single AWS Region to provide key isolation, as well as keys that can be replicated between Regions for seamless cryptographic functionality around the globe.

Encryption at rest

Encryption at rest is straightforward to implement on the RDS platform. Whenever you create an RDS database instance or an Aurora cluster, you need only check the box that indicates that your volume is to be encrypted, and then select the appropriate key to use for that encryption. From that point forward, the RDS platform will encrypt the entire database volume using the specified key. Snapshots and automated backups created from this volume will also be encrypted using the same key. When selecting a key, you can either choose a customer managed key that you have created or the AWS KMS key. Using a customer managed key offers you greater control with regard to key rotation, key material origin, and Regionality. Additionally, if you are planning to share an encrypted snapshot, it is possible to grant access to a customer managed key, but you cannot grant access to an AWS KMS key.

So far, we have discussed the encryption of on-disk data. However, cell-level or column-level encryption will depend on the specific database engine you are running on RDS.

Encryption in transit

Please refer to the In-transit data encryption section below under the SQL Server–specific security recommendations.

Secrets Manager and password rotation

Although there are various ways to authenticate to a database instance, perhaps the most common is using a username and password. Often, these credentials are stored unencrypted in code or configuration files. This presents a significant security risk. If your application requires username and password authentication, you should consider storing those credentials in AWS Secrets Manager. Secrets Manager allows you to encrypt your credentials and then access them using an IAM role, as discussed previously in this whitepaper. Encryption ensures that usernames and passwords are never stored in plaintext and are

not embedded in application code. AWS provides tutorials on how to use Secrets Manager in your organization.

Auditing and monitoring

CloudTrail integration

AWS CloudTrail allows users to log actions performed in an AWS account. Actions can be initiated by a user, role, or AWS service. Regardless of the source (the console, CLI, SDK), these actions are recorded for future auditing, governance, and compliance of your AWS account. CloudTrail is automatically enabled and does not require any manual setup.

CloudTrail is an important part of RDS security, as it provides an audit mechanism for changes made to database resources. If, for example, an RDS instance is created and you need to know who created that instance, that information can be identified using CloudTrail. Additionally, CloudTrail provides the AWS CloudTrail Insights feature that helps you detect anomalies that CloudTrail discovers, alerting you to a potential security risk.

Event notifications

RDS event notification provides a mechanism to invoke an Amazon Simple Notification Service (Amazon SNS) topic when events related to RDS take place. For example, you might create an Amazon SNS topic that sends an email when invoked. You could then tie this SNS topic to an RDS event that is invoked when a database instance has an availability issue such as a shutdown or restart. Alternately, you can use an AWS Lambda SNS endpoint, so that programmatic action takes place in response to the invoked event.

CloudWatch metrics and alerting

Amazon CloudWatch metrics are a critical component of managing any RDS database. Although most RDS metrics are performance related, when certain performance metrics such as CPU increase beyond what is expected, it can be an indication of a bigger problem. CloudWatch metrics provides an alerting feature called Amazon CloudWatch alarms. These alarms can be configured like RDS events to invoke an SNS topic in response to CloudWatch metrics crossing specified thresholds.

By coupling CloudWatch metrics with machine learning, Amazon DevOps Guru can automatically detect anomalies in workloads and, like CloudWatch alarms and RDS Events, use SNS topics to respond to those anomalies.

Publish database logs to CloudWatch

RDS databases capture a variety of logs. Those logs vary by engine and are stored by default on storage local to the database instance. With RDS for SQL Server instances, a maximum of 30 error log files and seven SQL Server Agent log files are retained in the local storage. Error log files and SQL Agent log files

older than 7 days are automatically deleted from local storage. When working with a large number of database instances, it is not practical to connect to each instance to review its logs. Furthermore, processing the logs to find patterns or anomalies is also not a trivial undertaking. Fortunately, RDS offers the ability to publish database logs to Amazon CloudWatch Logs. By sending your logs to CloudWatch Logs, you now have a single, centralized repository to view logs. What's more, CloudWatch Logs offers the ability to query your logs through CloudWatch Logs Insights and detect anomalies in your logs using a log anomaly detector.

Configuration

Master user

The master user is the SQL Server login closest to the "sysadmin" server role that is provided on an RDS for SQL Server instance. This SQL Server login is created when an RDS instance is created and has the highest level of permissions available for the RDS instance. It is recommended best practice to provide a strong password for this login, create other logins with some subset of the master user's permissions, and then store the master user credentials in a secure location. This practice helps ensure the principle of least privilege and keeps your database secure.

Parameter groups

RDS Parameter groups allow you to create a set of parameters that can be applied to one or many RDS instances of the same database engine type. For example, you might create a parameter group for SQL Server 2019 Standard edition that can then be applied to all SQL Server 2019 Standard edition instances in your account in the specified Region. You benefit from centralized management and the standardization of parameters. Inside of the parameter group you can configure engine-specific security features. Those features are explored further later in this whitepaper.

Patch management

Patch management is critical to database security. Over time, new patches are released for any given database engine that can contain performance or stability improvements but often contain additional security fixes. In the case of RDS for SQL Server, patches are applied at the OS level and the database-engine level. In most cases, it is best practice to enable the automatic minor version upgrades feature. This feature ensures that your database instances are always kept up to date with the latest security patches. In order to minimize disruption to existing workloads, the minor version upgrade happens during your specified maintenance window. Operating system updates are automatically applied every time a host replacement scenario is invoked.

SQL Server–specific security recommendations

Authentication

SQL Server supports two authentication modes, 1) Windows authentication mode and 2) SQL Server and Windows authentication mode (mixed mode). In this section we will describe both approaches in more detail, highlight benefits and disadvantages, and provide recommended security best practices.

SQL Server authentication

When using SQL Server authentication, logins are created in SQL Server, and both the user's name and password are created using SQL Server and stored in SQL Server. Users connecting using SQL Server authentication must provide their credentials (login and password) every time that they connect.

The key advantages of SQL Server authentication are:

- Allows SQL Server to support older applications and applications provided by third parties that require SQL Server authentication.
- Allows SQL Server to support environments with mixed operating systems, where all users aren't authenticated by a Windows domain.
- Allows users to connect from unknown or untrusted domains, such as an application where established customers connect with assigned SQL Server logins to receive the status of their orders.
- Allows SQL Server to support web-based applications where users create their own identities.
- Allows software developers to distribute their applications using a complex permission hierarchy based on known, preset SQL Server logins.

The key disadvantages of SQL Server authentication are:

- If you are a Windows domain user with a login and password for Windows, you must still provide another (SQL Server) login and password to connect. Keeping track of multiple names and passwords is difficult for many users. Having to provide SQL Server credentials every time you connect to the database can be time-consuming.
- SQL Server authentication cannot use the Kerberos security protocol.
- Windows offers additional password policies that aren't available for SQL Server logins.
- The encrypted SQL Server authentication login password must be passed over the network at the time of the connection. Some applications that automatically connect will store the password at the client. These are potential points of insecurity.

[Source: Microsoft Technical Documentation, SQL Server Authentication]

By default, all RDS for SQL Server instances are deployed with support for SQL Server authentication and an administrator (master) SQL Server login. This SQL Server login is a not a member of the "sysadmin"

server role but is the account with highest security privileges in the instance, which can be delegated as needed. Assuming that SQL Server authentication is the authentication method of choice for your RDS for SQL Server deployment, the following best practices and guidelines are included to help secure your database solution:

- Use strong and complex passwords that cannot be easily guessed and are not used for any other accounts or purposes.
- Rotate passwords on a regular basis.
- Use Secrets Manager to store SQL Server credentials for your applications whenever possible.

Windows authentication

When a user connects through a Windows user account, SQL Server validates the account name and password using the Windows principal token in the operating system. The user identity is confirmed by Windows. SQL Server does not ask for the password and does not perform the identity validation. Windows authentication uses the Kerberos security protocol, provides password policy enforcement with regard to complexity validation for strong passwords, ensures support for account lockout, and supports password expiration. A connection made using Windows authentication is sometimes called a trusted connection because SQL Server trusts the credentials provided by Windows.

[Source: Microsoft Technical Documentation, Windows Authentication]

The key advantages of Windows authentication are:

- It is significantly more secure than SQL Server authentication. It relies on Windows for authentication with much stronger password policies, and credentials are never stored in the SQL Server instance.
- It supports the Kerberos security protocol.
- Access management is streamlined through Active Directory groups, including using already defined Active Directory groups to simplify security design and access control.
- The password is never be passed over the network at the time of connection.

The key disadvantages of Windows authentication are:

- Does not support web-application use cases where users create their own identities.
- Does not support applications or solutions deployed on environments with mixed operating systems, where all users aren't authenticated by a Windows domain.
- Does not support applications or solutions with users that need to connect from unknown or untrusted domains.

Windows authentication can be enabled for RDS for SQL Server in two ways: through an intermediate AWS Directory Service for Microsoft Active Directory or directly to a self-managed Microsoft Active Directory. The intermediate AWS Managed Microsoft AD offers the advantage of Kerberos support but

also requires that a one-way trust is set up between the managed Active Directory and the customer's self-managed Active Directory, an issue for many organizations. Direct integration with the customer's self-managed Active Directory is a simpler, more effective approach that avoids setting up a trust between Active Directory forests, sharing your AWS Managed Microsoft AD with other accounts, or extending your AWS Managed Microsoft AD into additional Regions. The key drawback for this approach is the current lack of Kerberos support.

If you choose Windows authentication to authenticate your RDS for SQL Server deployment, the following best practices and guidelines are included to help secure your database solution:

- Favor RDS for SQL Server direct integration with the self-managed Active Directory implementation option by default.
- Use AWS Managed Microsoft AD as an intermediate entity to connect to the customer's self-managed Active Directory forest, if your solution already envisions a managed Active Directory as a component, requires Kerberos support, or other RDS database engines require interacting with the Active Directory.
- Create SQL Server logins based on Windows groups instead of Windows users. Directly manage group membership in the corresponding Microsoft Active Directory forest.
- Enable multifactor authentication on the self-managed Active Directory forest hosting the customer identities.

Using other identities

The final identity type supported in RDS for SQL Server is contained database users. Contained database users authenticate SQL Server connections at the database level. A contained database is a database that is isolated from other databases and from the instance of SQL Server (and the master database) that hosts the database. SQL Server supports contained database users for both Windows and SQL Server authentication.

This identity type is only relevant if the solution being built is using contained databases. This approach presents advantages and limitations. The key advantage is the portability that your database gains by having the capability of more easily switching hosts (SQL Server instances) without any dependencies on the SQL Server master database. Other advantages are simplified AlwaysOn failover requirements (no need to set up server logins on the secondary node), database-contained administration options, and a streamlined development experience.

For contained database users, the same best practices suggested above for SQL Server authentication and Windows authentication still apply.

Authorization

Granting, revoking, and denying permissions

Permissions in the database engine are managed at the server level through logins and server roles, and at the database level through database users and database roles. Logins are separate from database users. First, separately map logins or Windows groups to database users or roles. Next, grant permissions to users, server roles, or database roles to access database objects.

RDS supports all SQL Server built-in authorization and permission management features available to control data access. The following best practices and guidelines are included to help secure access to your database solution:

- Choose Windows authentication over SQL Server authentication, whenever possible.
- For Windows authentication scenarios: use least-privilege role-based security strategies to improve security management. Place Active Directory users in Active Directory groups, make these Active Directory groups members of the appropriate user-defined database roles, and grant these database roles the minimum permissions required by the application.
- For SQL Server authentication scenarios: use least-privilege role-based security strategies to improve security management. Create a minimum set of service accounts depending on the roles needed to operate the application, and create corresponding SQL Server logins for these service accounts. Make SQL Server logins members of the appropriate user-defined database roles, and then grant these database roles the minimum permissions required by each operating role within the application.

Restricting data access to selected data elements

Organizations often need to protect data at the column level, as data regarding customers, employees, trade secrets, product data, healthcare, financial, and other sensitive data is often stored in SQL Server databases. Sensitive columns often include identification or social security numbers, mobile phone numbers, first names, family names, financial account identification, and other data that could be deemed personally identifiable information (PII). Similarly, personal health information (PHI) requires equivalent levels of protection. Row-level data restrictions driven by user execution context are also often required by customers in the field.

SQL Server offers a number of data access control features specifically designed to address these security requirements, fully supported by RDS. The following best practices and guidelines are included to help address them:

- Restrict access to sensitive columns (PII or PHI data) with more granular column-level GRANT permissions using the same least-privilege role-based security strategies described above.
- Obfuscate data for sensitive columns (PII and PHI data) with dynamic data masking.
- Restrict access to select rows based on user execution context with row-level security, if needed. Typical use cases include: creating a security policy that allows nurses to view data rows for only their patients, creating a policy to restrict access to financial data rows based on an

employee's business division or role in the company, or creating a policy to enforce logical separation of each tenant's data rows in a multitenant application.

Data encryption

File-level data encryption

SQL Server offers the transparent data encryption (TDE) feature to implement data encryption at rest. TDE ensures that database files, backup files, and TempDB files can't be attached and read without proper certificates decrypting database files.

RDS offers similar functionality, but at the storage volume level. You can choose to encrypt the volume where all database files are hosted, and RDS uses the same AES_256 encryption algorithm based on the Advanced Encryption Standard. The key difference between SQL Server TDE and RDS volume encryption is database granularity. With SQL Server TDE you can choose which databases on your instance will be encrypted. The following best practices and guidelines are for file-level data encryption of your database solution:

- Favor RDS storage-level encryption over SQL Server TDE, unless database granularity is required for your database solution. TDE is an Enterprise edition feature for all SQL Server versions older than 2019.
- Avoid implementing both encryption methods at the same time on an RDS instance. Simultaneously using TDE and RDS encryption at rest might slightly affect the performance of your database.
- If SQL Server TDE is used, ensure that TDE certificates are properly backed up and archived for further reference.

Column-level data encryption

SQL Server offers two column-level data encryption features specifically designed to address these security requirements, fully supported by RDS: column-level encryption and Always Encrypted. The key difference between these features is that Always Encrypted allows clients to encrypt sensitive data inside client applications and never reveal the encryption keys to the database engine. This provides a separation between those who own the data and can view it and those who manage the data but shouldn't have access: database administrators, cloud database operators, or other high-privileged unauthorized users. On the other hand, the key drawbacks of Always Encrypted are that it requires a special database driver, and it imposes certain limitations on the type of queries supported for these encrypted columns. The following best practices and guidelines are included here for your reference:

- Favor Always Encrypted as the column encryption method in RDS, unless technical prerequisites or documented limitations prevent the use of this option.

In-transit data encryption

In addition to protecting data at rest, protection of data over the wire is of paramount importance. SQL Server supports both transport layer security (TLS) and secure sockets layer (SSL) protocols. The TLS and SSL protocols are located between the application protocol layer and the TCP/IP layer, where they can secure and send application data to the transport layer. TLS and SSL assume that a connection-oriented transport, typically TCP, is in use. The protocol allows client and server applications to detect message tampering, interception, or forgery.

RDS offers the possibility of forcing all client connections to be encrypted (rds.force_ssl parameter), disabling older versions of TLS such as 1.0 and 1.1, and disabling older cyphers such as RC4 and Triple DES. The following best practices and guidelines are included here for your reference:

- Use the RDS parameter "*rds.force_ssl*" to force all client connections to be encrypted whenever possible.
- Use the RDS parameters "*rds.tls11"* and "*rds.tls10"* to disable older versions of TLS whenever possible.

Auditing and compliance

Database auditing is a critical part of compliance and security audits, aimed at protecting corporate data. In environments where SQL Server is used, auditing is a basic requirement for security, as well as for compliance standards including ISO-27001, PCI-DSS, BASEL3, GDPR, IG, and HIPAA eligibility.

SQL Server auditing

Auditing an instance of the SQL Server Database Engine or an individual database involves tracking and logging events that occur on the database engine. SQL Server audit lets you create server audits, which can contain server audit specifications for server-level events and database audit specifications for database-level events.

[Source: Microsoft Technical Documentation, Auditing]

RDS fully supports SQL Server auditing at both the server and database levels. This feature is enabled through an RDS option where you define the destination S3 bucket, compression options, and data-retention options. The following best practices and guidelines are included here for your reference:

- Audit tables and columns with sensitive data that have security measures applied to them. If a table or column is important enough to need protection by a security capability, then it should be considered important enough to audit.
- Audit and regularly review tables that contain sensitive information but where it is not possible to apply desired security measures due to some kind of application or architectural limitation.

- Set up a retention time window in Amazon S3 for your audit files, to keep your audit file set manageable and avoid unnecessary storage costs.

HIPAA eligibility with SQL Server

The Health Insurance Portability and Accountability Act (HIPAA) requires the confidentiality, integrity, and availability of PHI to be maintained with administrative, physical, and technical safeguards.

Your RDS for SQL Server instance is already by default HIPAA eligible at launch. To help your HIPAA compliance efforts, as user databases are deployed and operated on the RDS for SQL Server instance, adhere to the following requirements:

Requirement #1 (data encryption): Data at rest (stored data) and data in motion (transmitted data) must be encrypted to prevent unauthorized access. Use RDS for SQL Server storage encryption to satisfy the "encryption at rest" portion of the requirement and the RDS for SQL Server force SSL feature to satisfy the "encryption in motion" portion of the requirement.

Requirement #2 (encryption key management): Decryption keys must be adequately protected and backed up to prevent unauthorized access to PHI. Use AWS KMS built-in security features to satisfy this requirement.

Requirement #3 (unique login credentials): Unique login credentials allow access to PHI to be tracked and managed. Employees should never share their login credentials. RDS for SQL Server Windows authentication and role-based security are the ideal mechanisms to satisfy this requirement.

Requirement #4 (access controls): Designate different levels of access to PHI to employees based on their job functions. HIPAA requires PHI access to adhere to the minimum-necessary standard, meaning that access to PHI should be restricted to only the information an entity or worker needs to perform their job role. Using the least-privilege role-based security strategies described in the sections above will help satisfy this requirement.

Requirement #5 (audit logs): Track access to PHI to ensure that it is not being accessed (or accessed excessively) without authorization. Audit logs establish normal access patterns for employees using unique login credentials. Tracking access to PHI limits the risk of insider threats to PHI, as it makes it easier to determine if employees are abusing their access privileges. Use the RDS for SQL Server audit option to satisfy this requirement.

Requirement #6 (data backup): Data backup ensures that in the event of a breach or natural disaster, data can be quickly restored. Use the RDS for SQL Server automated backups feature, which provides cross-Region point in time recovery capabilities with a 5-minute RPO, to satisfy this requirement

Requirement #7 (dedicated infrastructure): HIPAA rules for database security require sensitive data in a high-security infrastructure. This requirement is met by the network isolation capabilities already

available in RDS for SQL Server. The requirement is met at the service level and is not affected by user databases deployed on the instance.

Requirement #8 (patch management and software updates): Ensure that all software is up to date, mitigating the risk of hackers exploiting vulnerabilities in software. Without managed updates, it might be difficult to implement software patches in a timely manner. Use RDS for SQL Server built-in automated patch management and software update features to satisfy this requirement.

Requirement #9 (data disposal): Requires PHI to be adequately destroyed, preventing data from being reconstructed. Paper records should be shredded, pulped, or burned. Electronic protected health information (ePHI) disposal requires hard drives to be either shredded or degaussed to permanently erase data. Use SQL Server built-in data management (partitioning, for example) features to satisfy this requirement.

Requirement #10 (business associate agreements): Business associate agreements must be completed with all business associates (BAs) before they are permitted to create, store, transmit, or maintain PHI on behalf of a covered entity. BAs can include third-party database administrators, development teams, and software support vendors. This requirement is out of scope for RDS for SQL Server but is listed here for comprehensiveness of HIPAA requirements. This requirement is a more legal and procedural requirement regarding sharing PHI data with third parties.

PCI-DSS compliance with SQL Server

The Payment Card Industry Data Security Standard (PCI-DSS or PCI Standard), defined and controlled by https://www.pcisecuritystandards.org/, is designed to prevent credit card information fraud through increased controls around credit card data. Obtaining PCI-DSS compliance is a requirement for all organizations that accept credit card payments, process credit card transactions, or transmit or store credit card data.

Your RDS for SQL Server instance is already by default PCI-DSS compliant at launch. In order to maintain PCI-DSS compliance, as user databases are deployed and operated on the RDS for SQL Server instance, adhere to the following requirements:

Requirement #1 (implement firewalls): Customers must install and maintain a firewall configuration to protect cardholder data. This requirement is met and exceeded by the multiple network isolation artifacts available on AWS (VPCs, subnets, route tables, network ACLs, and security groups). Every RDS for SQL Server instance is by default protected by a firewall; therefore all user databases deployed on the instance are also protected by a firewall.

Requirement #2 (vendor-supplied system passwords): Customers are not to use vendor-supplied defaults for system passwords and other security parameters. These other security parameters pertain to handling things like wireless router password rotations, developing configuration standards to address known security vulnerabilities, and limiting responsibility of systems. RDS for SQL Server Windows authentication

and role-based security are the ideal mechanisms to satisfy this requirement. If SQL Server logins are required for other third-party components to access any PCI-relevant user databases, ensure that any default vendor-provided passwords are changed.

Requirement #3 (stored cardholder data): Customers must protect stored cardholder data. This requirement refers to the need for data protection at rest. Follow the best practices documented in the File-level data encryption and Column-level data encryption sections above to satisfy this requirement.

Requirement #4 (in-flight cardholder data): Customers are to encrypt transmission of cardholder data across open, public networks. This requirement covers using strong cryptography and security protocols, securing any wireless networks, not sending unprotected primary account numbers across messaging applications, and having proper security policies in place. Follow the best practices documented in the In-transit data encryption section above to satisfy this requirement.

Requirement #5 (protect all systems against malware): Customers are to protect all systems against malware and regularly update antivirus software or programs. This requirement is already met by hosting your SQL Server databases on RDS.

Requirement #6 (secure systems and applications): Customers are to develop and maintain secure systems and applications. This requirement is a shared responsibility between AWS and the customer. AWS provides many resources to help you monitor your security state and securely communicate with AWS resources. Use AWS Security Hub to strengthen your solution's security posture on an ongoing basis. Security Hub is a cloud security posture management service that performs automated, continual security best-practice checks against your AWS resources to help you identify misconfigurations. Security Hub aggregates your security alerts (its findings) in a standardized format so that you can more easily enrich, investigate, and remediate them.

Requirement #7 (protect access to cardholder data): Customers are to restrict access to cardholder data by business "need to know." This requirement is on the customer to manage and control. However, AWS provides many features and services to help meet the requirement. Follow the best practices documented in the Authorization section above.

Requirement #8 (identify and authenticate access to system components): Customers are to identify and authenticate access to system components. Use Windows authentication for identities accessing PII data in your system and follow the best practices documented in the Windows authentication section above. Make sure that multifactor authentication is enabled for the Active Directory hosting these identities.

Requirement #9 (restrict physical access to cardholder data): Customers are to restrict physical access to cardholder data. This requirement is already met by hosting your SQL Server databases on RDS.

Requirement #10 (monitor access to cardholder data): Customers must track and monitor all access to network resources and cardholder data. Follow the best practices documented in the SQL Server auditing

section above to satisfy this requirement. Additionally, use SQL Server event logs that are already stored and archived by RDS.

Requirement #11 (regularly test security systems and processes): Customers must regularly test security systems and processes. AWS facilitates meeting this requirement with the security testing policies and processes already in place for RDS, including automated patching. However, this requirement is a shared responsibility, and you must have mechanisms and processes in place for regular security testing of your systems as well as identification, prioritization, and remediation of internal vulnerabilities. You should also enlist external companies to do a routine vulnerability and penetration test at least once a year to satisfy this requirement.

Requirement #12 (information security policy): Customers must maintain a policy that addresses information security for all personnel. This requirement is essentially owned by the customer. In order to satisfy this requirement, you must: maintain a comprehensive information security policy that governs and provides direction for protection of your information assets; provide a clearly defined acceptable use policy for end-user technologies; create ongoing security awareness education; screen personnel to reduce risk from insider threats; and properly manage risks to information assets associated with third-party service provider relationships.

Conclusion

Data security is a top priority for all customers. In this document we have described in detail the different approaches available in Amazon RDS for SQL Server to strengthen your overall security posture, including prescriptive best practices recommendations for authentication (validating users), authorization (data access management), data encryption, auditing, and compliance.

Contributors

Contributors to this document include:

- Camilo Leon, Principal Database Specialist Solutions Architect, AWS

Further Reading

For additional information, refer to:

- AWS Architecture Center
- Security in Amazon RDS
- Security Pillar of the AWS Well-Architected Framework
- Amazon RDS for SQL Server

- Amazon RDS Custom for SQL Server

Document revisions

Date	Description
August 28, 2024	First publication