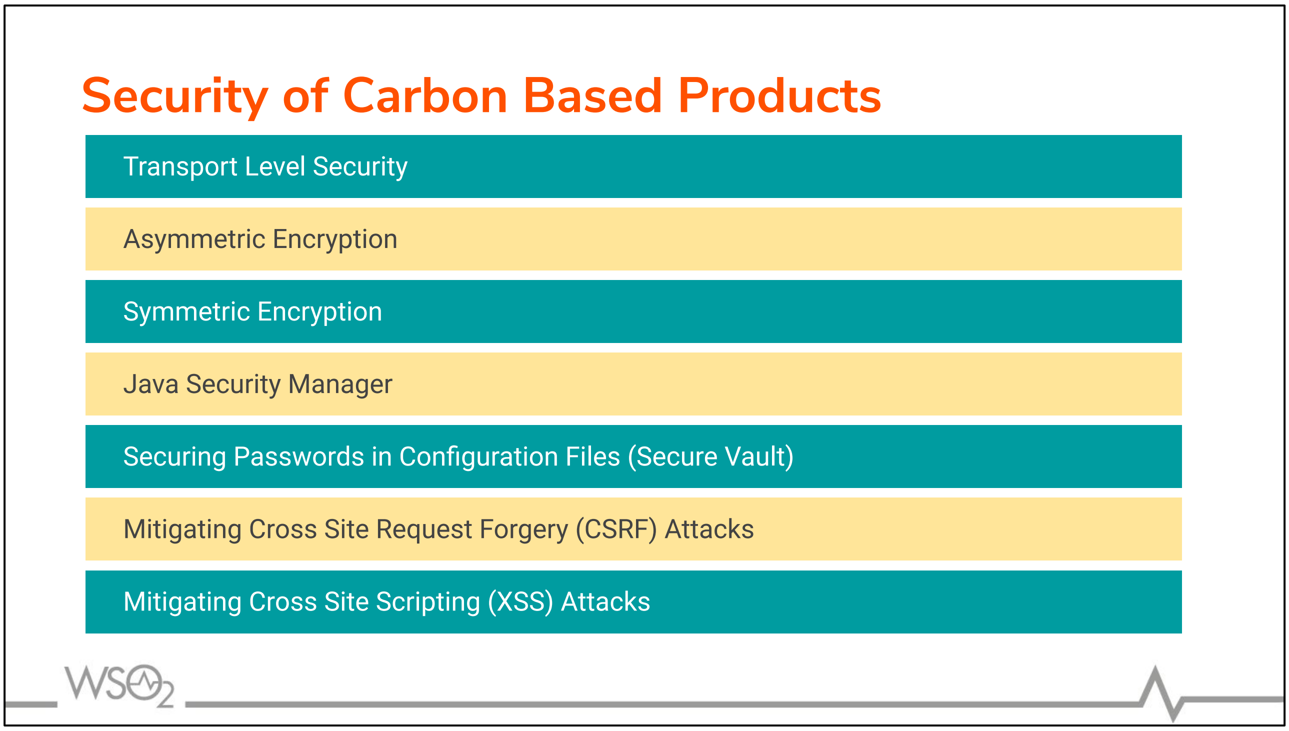
WSO2 Product Administration

WSO2 Training

Module 07 - Security

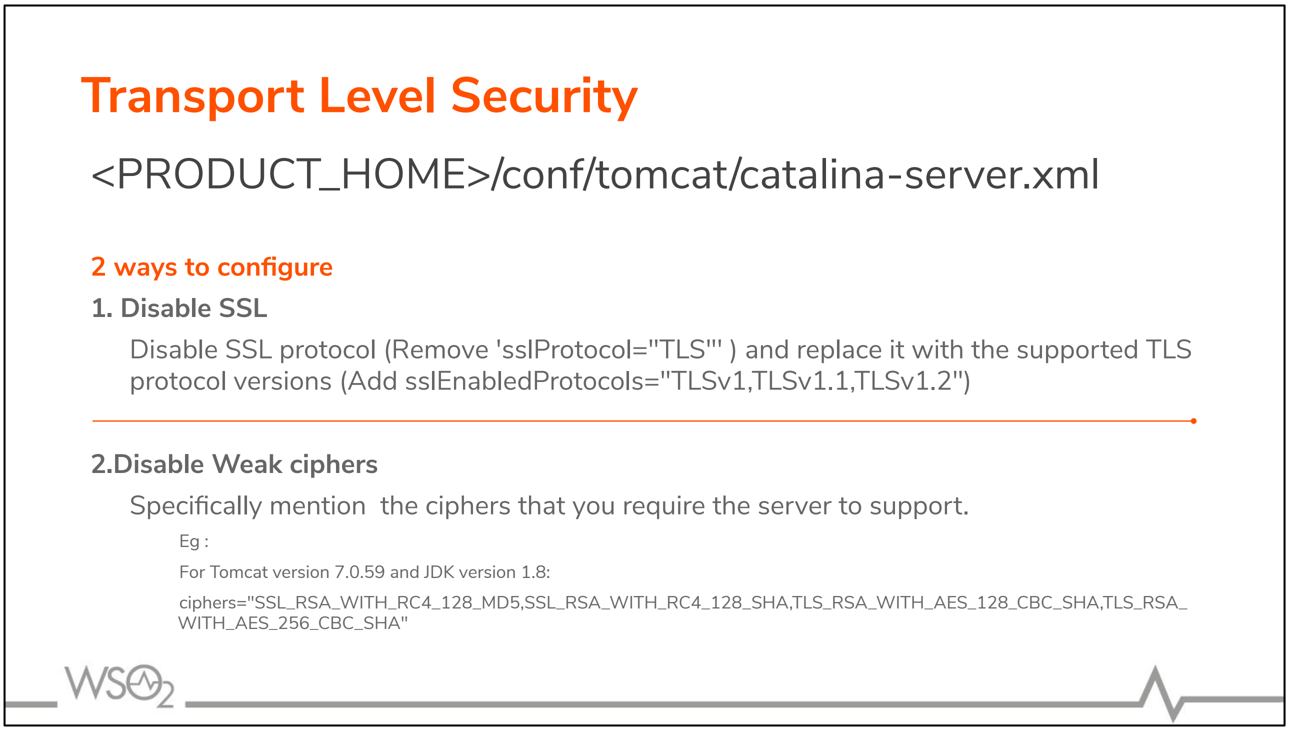


Security of Carbon Based Products

- Transport Level Security
- Asymmetric Encryption
- Symmetric Encryption
- Java Security Manager
- Securing Passwords in Configuration Files (Secure Vault)
- Mitigating Cross Site Request Forgery (CSRF) Attacks
- Mitigating Cross Site Scripting (XSS) Attacks

Security can be applied on WSO2 Carbon based products in various aspects.
All WSO2 products are shipped with a self-signed certificate.
The recommendation is to change this certificate before production deployment.
This can be done by changing the certificate in the Key Store.

https://docs.wso2.com/display/ADMIN44x/Security+Guidelines+for+Production+Deployment



Transport Level Security

<PRODUCT_HOME>/conf/tomcat/catalina-server.xml

2 ways to configure

1. Disable SSL

Disable SSL protocol (Remove 'sslProtocol="TLS"') and replace it with the supported TLS protocol versions (Add sslEnabledProtocols="TLSv1,TLSv1.1,TLSv1.2")

2.Disable Weak ciphers

Specifically mention the ciphers that you require the server to support.

Eg :

For Tomcat version 7.0.59 and JDK version 1.8:

ciphers="SSL_RSA_WITH_RC4_128_MD5,SSL_RSA_WITH_RC4_128_SHA,TLS_RSA_WITH_AES_128_CBC_SHA,TLS_RSA_WITH_AES_256_CBC_SHA"

The transport level security protocol of the Tomcat server is configured in the .
<PRODUCT_HOME>/conf/tomcat/catalina-server.xml.
By default the ssLprotocol attribute is set to "TLS"

There are 2 ways to configure transport level security.

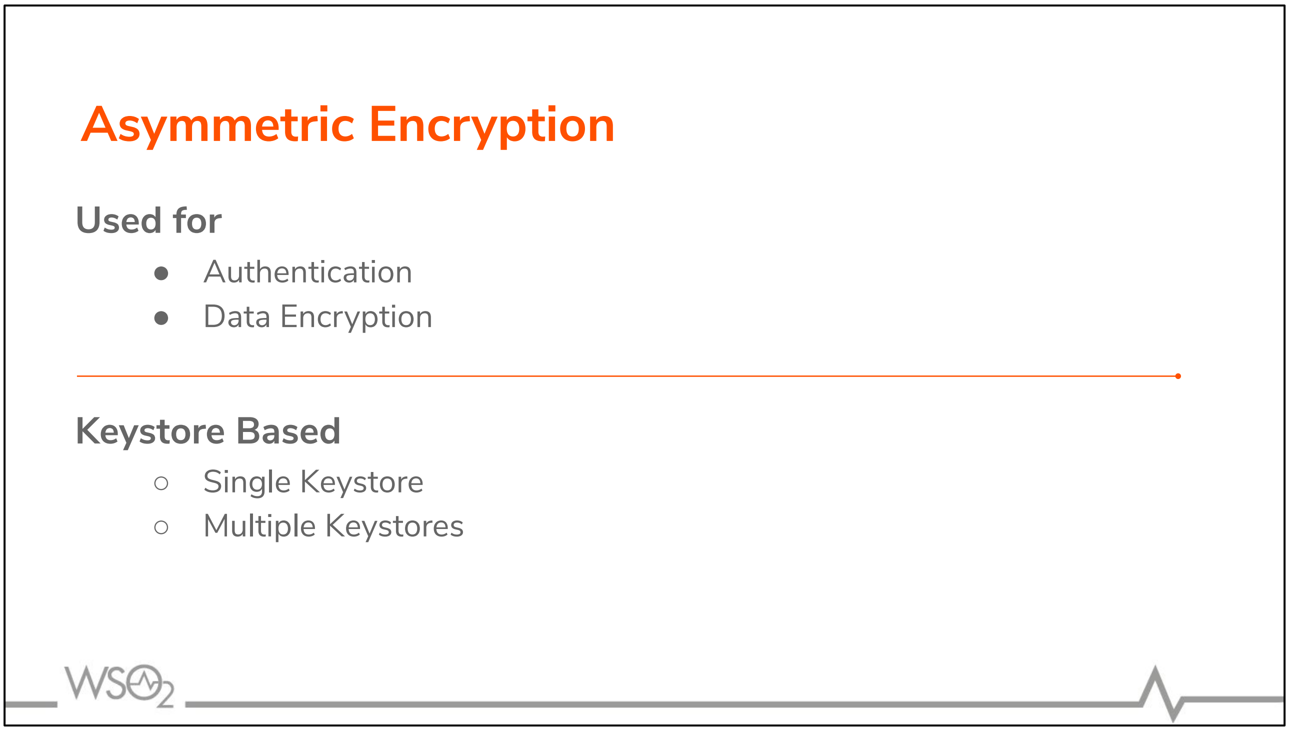
1. Disable SSL
It is recommended to disable SSL in Carbon servers due to a bug in the SSL protocol that exposes critical client and server related encrypted data encrypted.
This bug convinces the client , that the server does not support a more secure TLS protocol, there by forcing it to connect via SSL. Due to this, we need to disable SSL protocol (Remove 'sslProtocol="TLS"') and replace it with the supported TLS protocol versions (Add sslEnabledProtocols="TLSv1,TLSv1.1,TLSv1.2")

2.Disable Weak ciphers

Default weak ciphers create security risks and make the system vulnerable to attacks such as the Logjam attack on Diffie-Hellman key exchange.
If not specified, the browser will support all SSL ciphers including the weak ciphers.

To prevent this, it is recommended to disable weak ciphers and specify the ciphers that you require the server to support.
This is definedi the
<PRODUCT_HOME>/repository/conf/tomcat/catalina-server.xml

Eg :
For Tomcat version 7.0.59 and JDK version 1.8:
ciphers="SSL_RSA_WITH_RC4_128_MD5,SSL_RSA_WITH_RC4_128_SHA,
TLS_RSA_WITH_AES_128_CBC_SHA,TLS_RSA_WITH_AES_256_CBC_SH
A"



Asymmetric Encryption

Used for

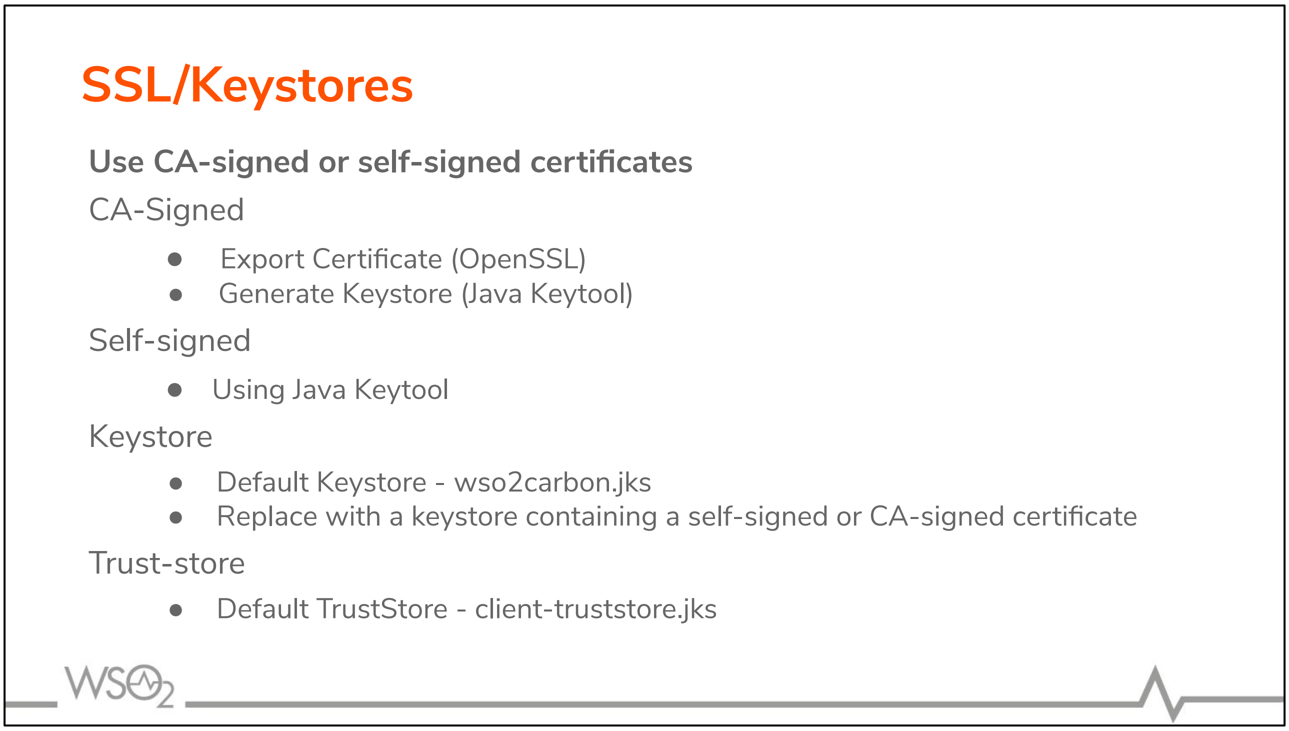
- Authentication
- Data Encryption

Keystore Based

- Single Keystore
- Multiple Keystores

Default Authentication and Data encryption mechanisms is WSO2 products uses asymmetric encryption.

In asymmetric encryption, keystores (Containing Key pairs and Certificates) and made available for each product.
You can configure multiple key store enabling the usages of different keys for different use cases.



SSL/Keystores

Use CA-signed or self-signed certificates

CA-Signed

- Export Certificate (OpenSSL)
- Generate Keystore (Java Keytool)

Self-signed

- Using Java Keytool

Keystore

- Default Keystore - wso2carbon.jks
- Replace with a keystore containing a self-signed or CA-signed certificate

Trust-store

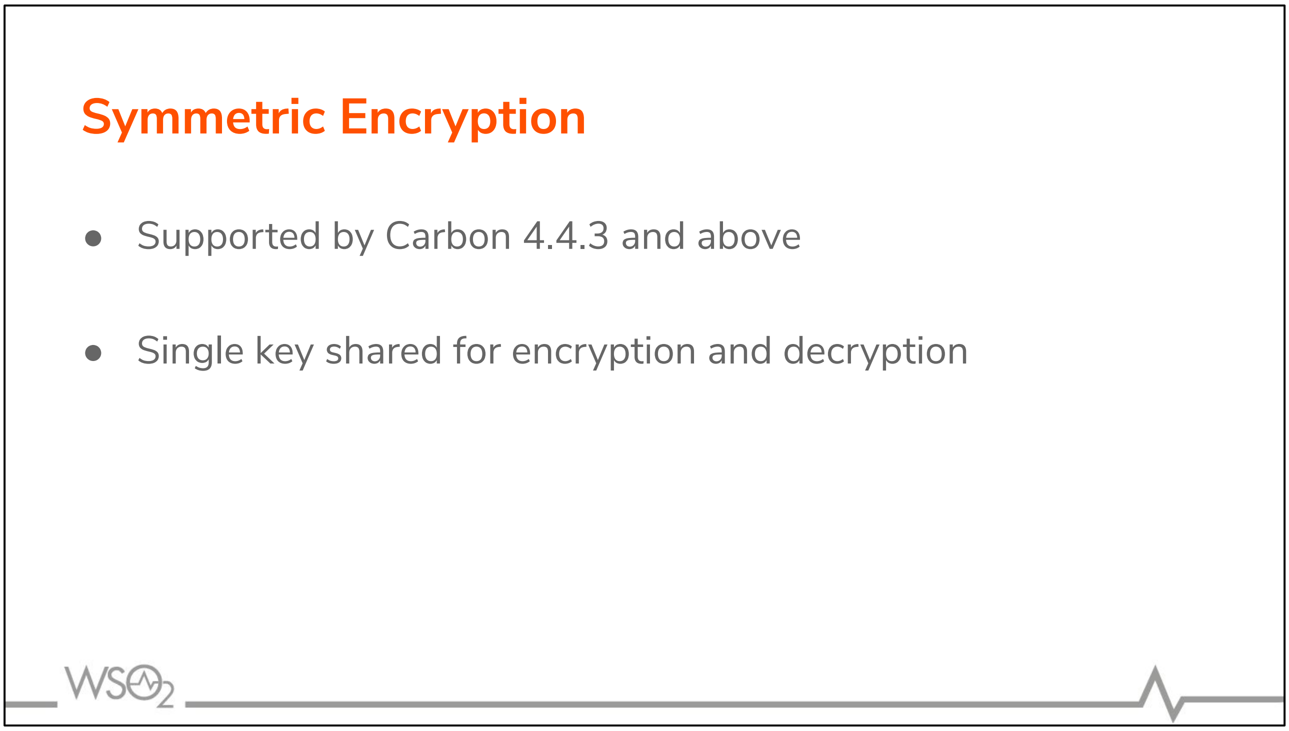
- Default TrustStore - client-truststore.jks

SSL protocol enables secure communication among systems.
It uses a public key, private key and a random symmetric key to encrypt data.
Certificates that can be reused may already exist. You can use these CA-signed certificates to generate a Java keystore using OpenSSL(To export the certificate) and the Java keytool (To generate the key store).
You can also create a keystore and a new certificate using the Java keytool

Key store
WSO2 Carbon-based products are shipped with a default keystore named wso2carbon.jks in the <PRODUCT_HOME>/repository/resources/security directory.
This comes with a private/public key pair used to encrypt sensitive information, when communicating over SSL and for encryption/signature purposes in WS-Security.
The recommendation for production deployments is to replace this keystore as anyone can have access other private key of this.

Truststore
In the SSL handshake, the client verifies the server certificate. For this, the client needs to have a stored list of trusted certificates , in a trust store. All WSO2 products are shipped with the trust store named client-truststore.jks, in the <PRODUCT_HOME>/repository/resources/security/ directory.

They public certificate of the keystore server, needs to be imported into this truststore during SSL based communication of WSO2 products.

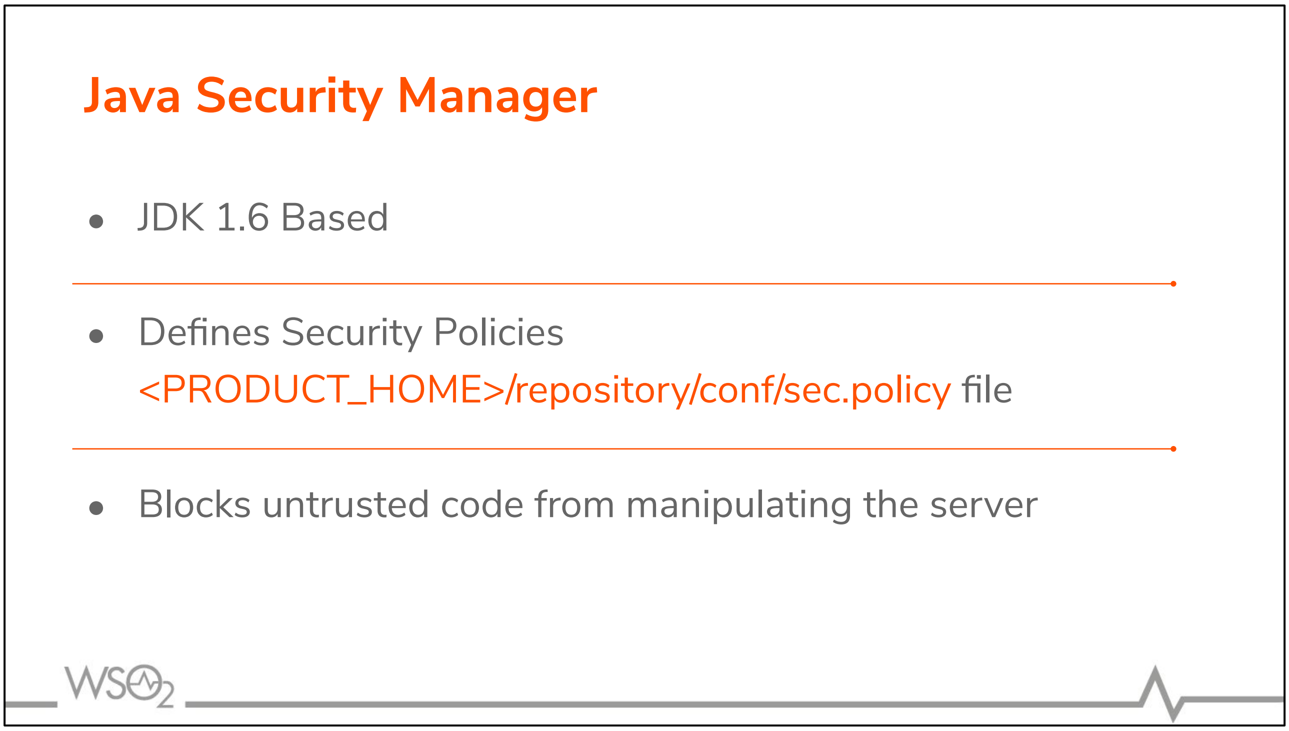


Symmetric Encryption

- Supported by Carbon 4.4.3 and above
- Single key shared for encryption and decryption

WSO2 Products based on Carbon 4.4.3 and above provide the option of switching between asymmetric and symmetric encryption.

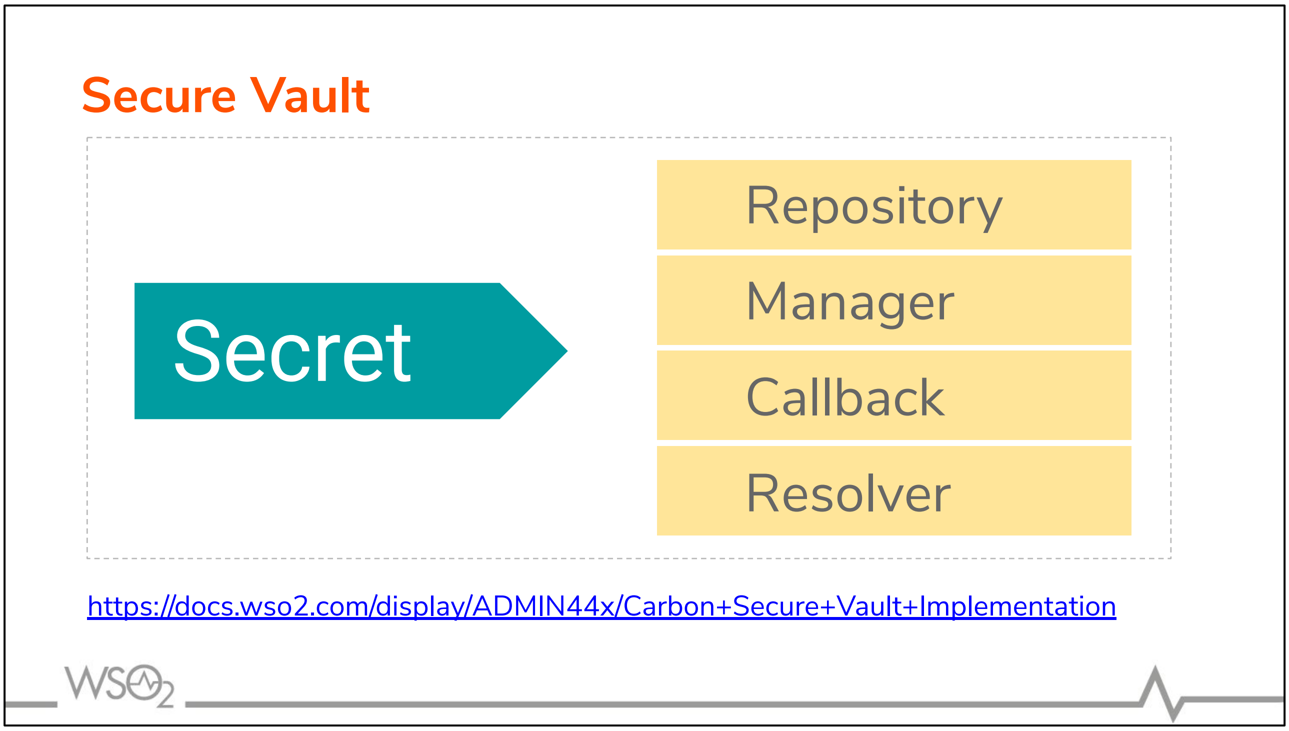
Symmetric encryption allows a single key to be shared for encryption and decryption purposes.



Java Security Manager

- JDK 1.6 Based
- Defines Security Policies
  <PRODUCT_HOME>/repository/conf/sec.policy file
- Blocks untrusted code from manipulating the server

The Java Security Manager defines security policies that prohibit untrusted code from manipulating the server.
Enabling this activates the Java permissions in <PRODUCT_HOME>/repository/conf/sec.policy file.
These permissions can be altered as required.



Secure Vault

https://docs.wso2.com/display/ADMIN44x/Carbon+Secure+Vault+Implementation

Plain text passwords in WSO2 product configuration files can be secured using the 'Secure Vault' implementation built into Carbon products.
The cipher tool (<PRODUCT_HOME>/bin/ciphertool.sh) enables this.

Secure vault implementation consists of 4 elements.

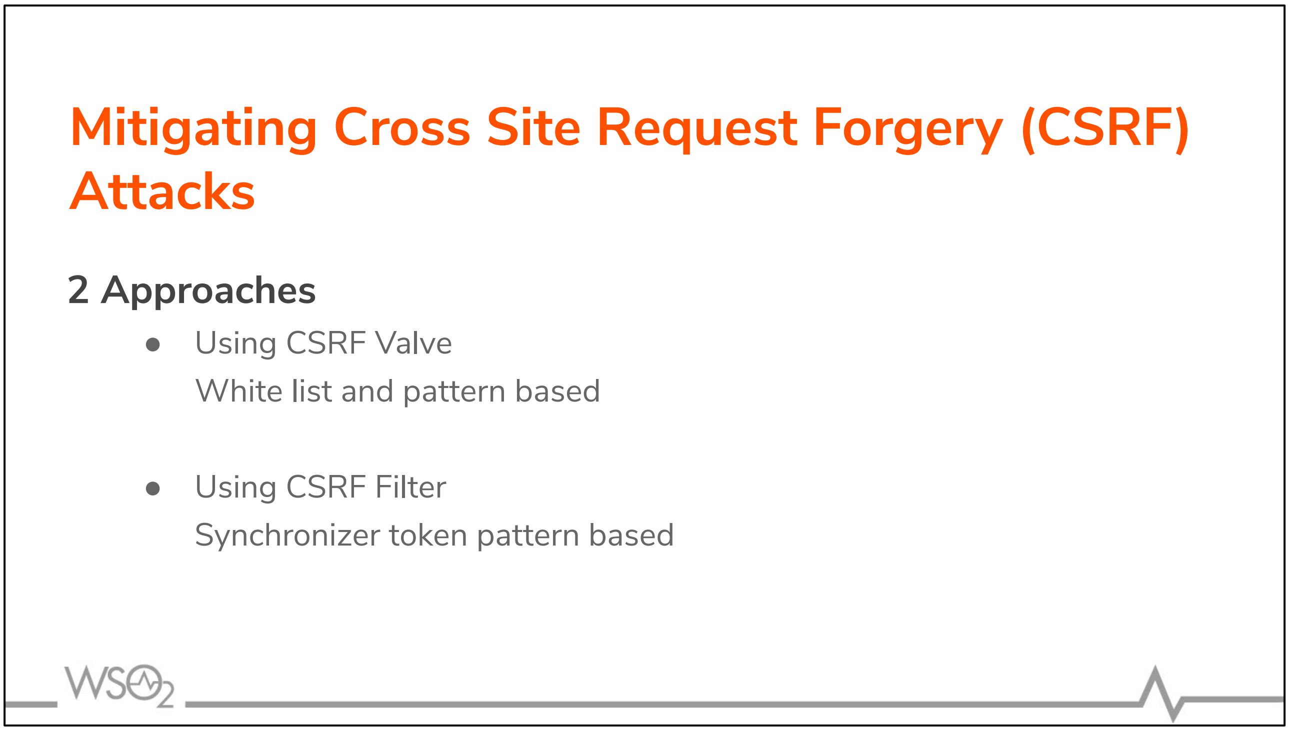
Secret repository:
This stores encrypted values of the plain text passwords in the <PRODUCT_HOME>/repository/conf/security/cipher-text.properties file against the alias defined in cipher-tool.properties file.

Secret Manager:
<PRODUCT_HOME>/repository/conf/security/cipher-tool.properties file stores password aliases against their xpaths.
Secret manager initializes the secret repository and keystore of the carbon server. The secrets (passwords) of the secret repository are accessed using aliases indicated in the cipher-tool.properties file.
The decryption crypt, used to resolve encrypted secret values is created using the keystore.

Secret Callback:
The SecretManagerSecretCallbackHandler together with the secret Manager

resolve the secret to identify the actual password.

Secret Resolver:
Configuration builders that contain configuration files with secret information should be initialized with the secret resolver when building the configurations.
This keeps a list of secured elements that should be defined in the configuration file with secret aliases.
It initializes the Secret Callback handler class defined in the configuration file.



Mitigating Cross Site Request Forgery (CSRF) Attacks

2 Approaches

- Using CSRF Valve
  White list and pattern based

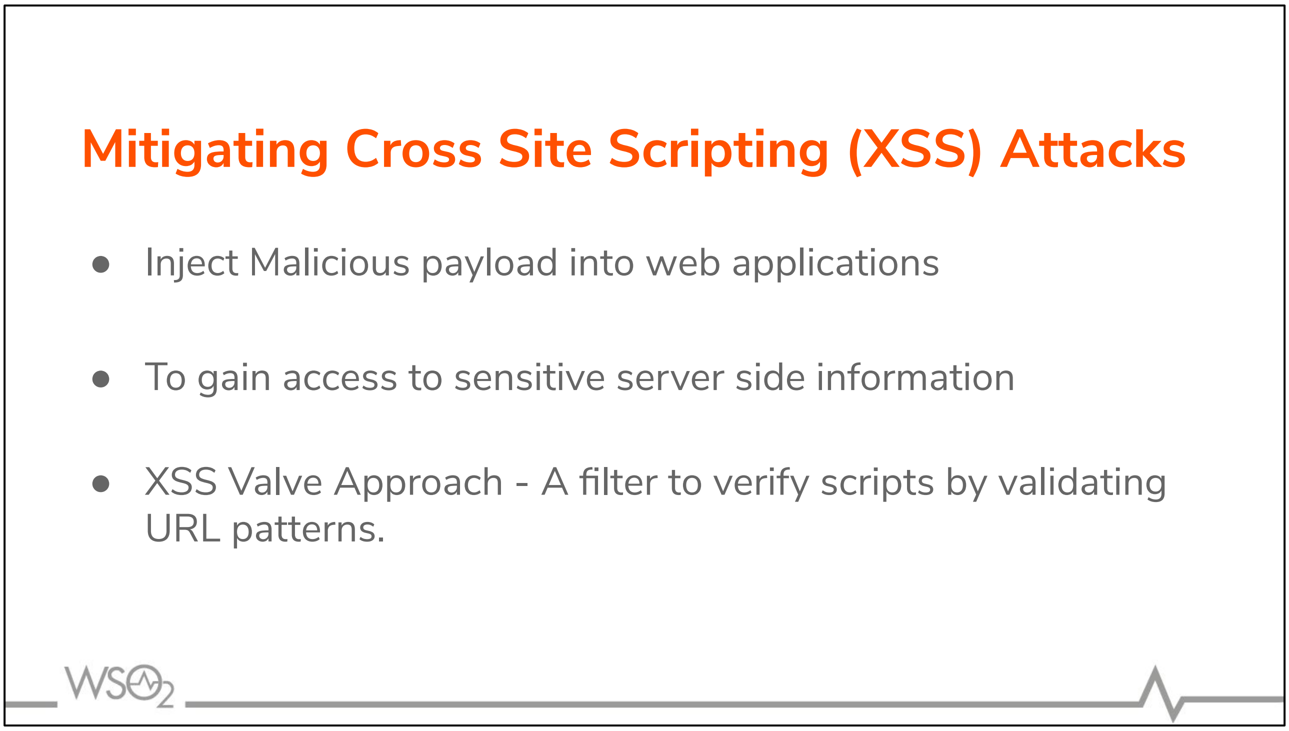
- Using CSRF Filter
  Synchronizer token pattern based

Cross Site Request Forgery (CSRF) attacks trick users into sending a malicious request, forcing the them to execute unwanted actions on a web browser where they are already authenticated. The attacker uses the same sessions in which the user has logged in to the web application on the browser , to bypass the authentication step.
There are 2 approaches to mitigate CSRF attacks.

1. Mitigating using the CSRF Valve
CSRF Valve is a filter that differentiates between malicious and legitimate requests by verifying the source of the request.
You can specify the list of sources and patterns associated with legitimate requests for the value to refer to.
These sources are specified in the <Whitelist> tag within <Security> element of the <PRODUCT_HOME>/repository/conf/carbon.xml

2. Mitigating using the CSRF Filter
The CSRF Filter uses the Synchronizer Token Pattern to mitigate CSRF attacks.
This requires generating a random challenge token associated with the user's current session.
It adds this token in the form of a hidden parameter in HTML forms and links associated with sensitive server-side operations.

When invoking such operations, HTTP requests should include these challenge tokens as well, which will then be verified by the server.

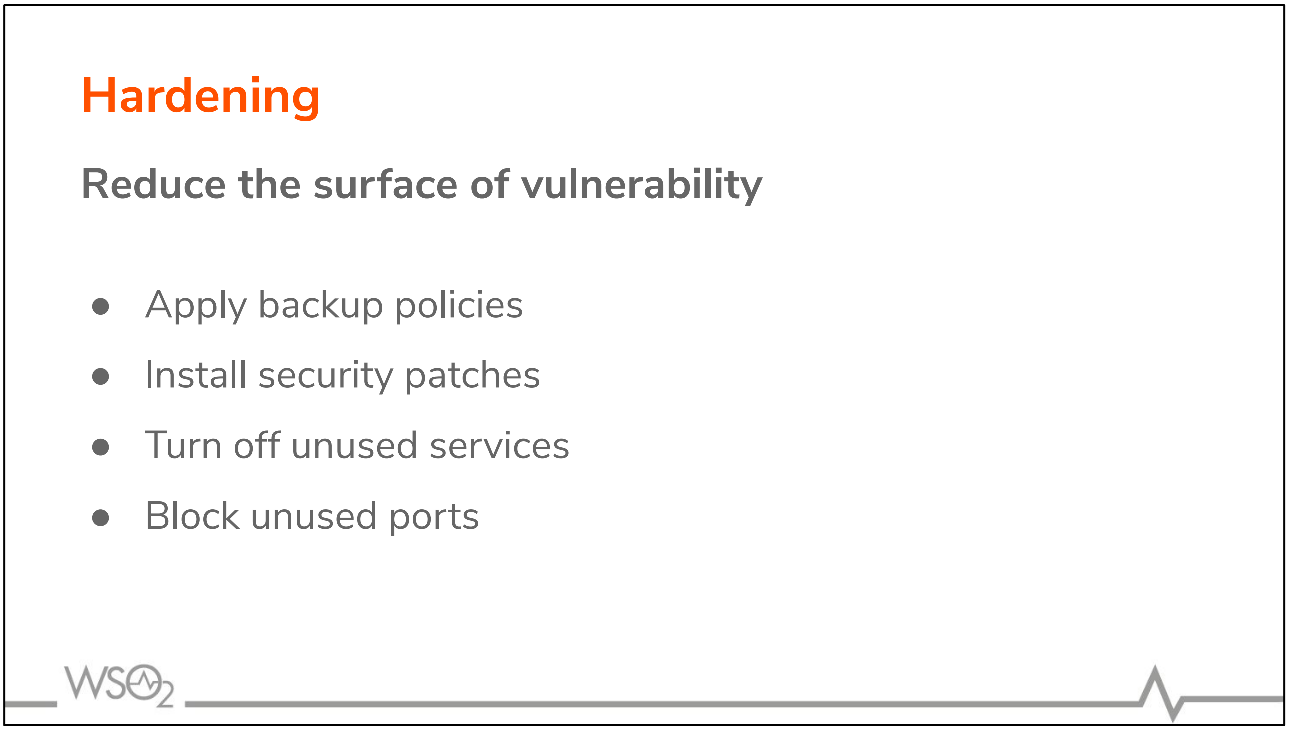


Mitigating Cross Site Scripting (XSS) Attacks

- Inject Malicious payload into web applications
- To gain access to sensitive server side information
- XSS Valve Approach - A filter to verify scripts by validating URL patterns.

Cross Site Scripting (XSS) attacks inject malicious scripts / payload into trusted legitimate web applications using client side scripts

Using this attackers gain access to sensitive page content, session cookies etc of web applications that are maintained by the web browser on behalf of the user.
To mitigate XSS attacks we can use the XSS Valve approach, which uses a filter to differentiate between malicious and legitimate scripts, by validating the URL patterns.



Hardening

Reduce the surface of vulnerability

- Apply backup policies
- Install security patches
- Turn off unused services
- Block unused ports

Hardening involves securing the deployment by reducing the surface of vulnerability.

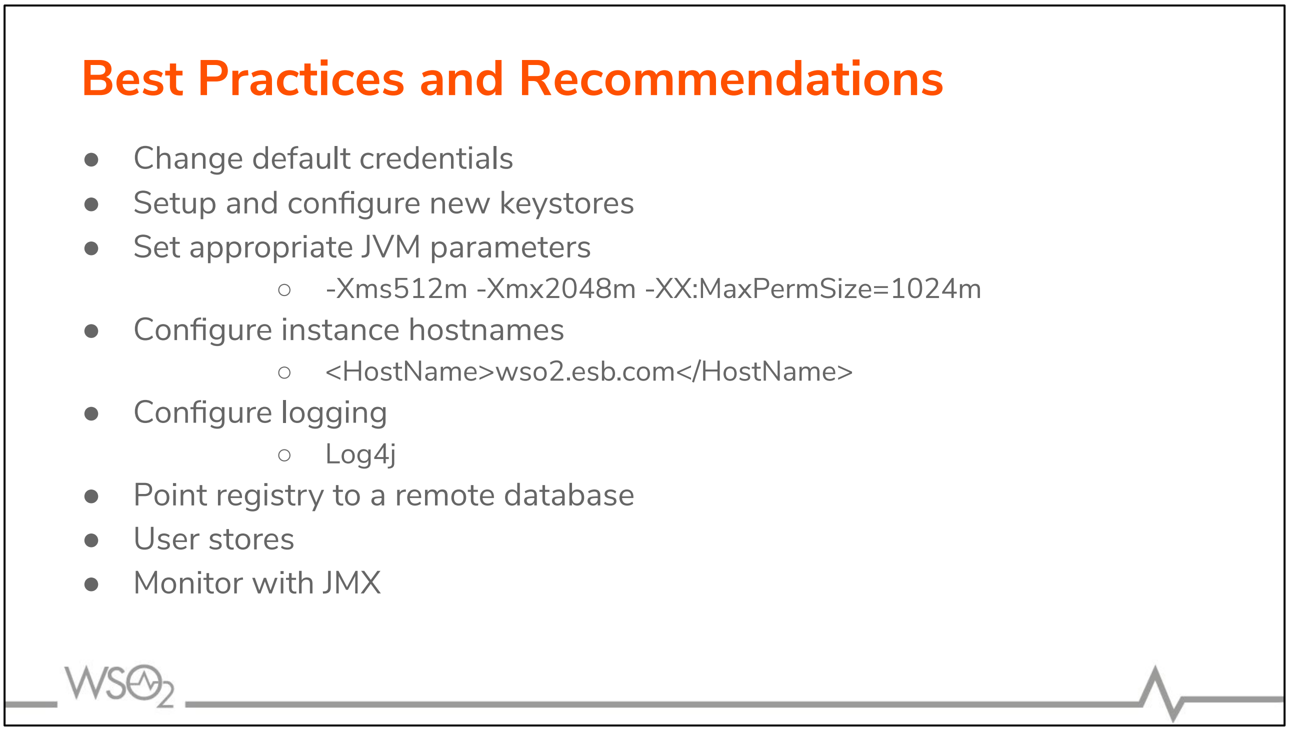
For this we recommend to apply the organization's backup policies on the entire deployment.
In addition, all security patches provided by the relevant OS vendor should be applied on each machine running instances of the deployment.
Turn off all services that are not utilized and add firewall rules to close all ports other than the ports used by WSO2 products.

In addition you can follow securing and hardening techniques depending on the OS.

Eg: For linux based servers

http://www.puschitz.com/SecuringLinux.shtml

http://www.tecmint.com/linux-server-hardening-security-tips/



Best Practices and Recommendations

- Change default credentials
- Setup and configure new keystores
- Set appropriate JVM parameters
  - -Xms512m -Xmx2048m -XX:MaxPermSize=1024m
- Configure instance hostnames
  - <HostName>wso2.esb.com</HostName>
- Configure logging
  - Log4j
- Point registry to a remote database
- User stores
- Monitor with JMX

There is a set of recommended best practices to be followed when setting up deployments to ensure security and maintainability.

Changing default credentials is a must as all servers use an administrator use with common credentials.

All WSO2 products use a default keystore which comes with a self signed SSL key. These keys are public, therefore the recommendation is to configure a new keystore.

Appropriate Heap and Permgen values for JVM should be configured based on the deployment scenarios.
These can be configured in the '<PRODUCT_HOME>/bin/wso2server.sh' file.
The recommended values for a typical deployment are as follows.
-Xms512m -Xmx2048m -XX:MaxPermSize=1024m
Also keep in mind that in order to run the JVM with 2 GB (-Xmx2048m), you should ideally have about 4GB of memory on the physical machine.

WSO2 products identify the current machine's hostname via the Java API by

default. But this sometimes causes errors in certain environment. Therefore it is recommended to configure a hostname explicitly for the server instance by setting the HostName parameters in
<Product_Home>/repository/conf/carbon.xml

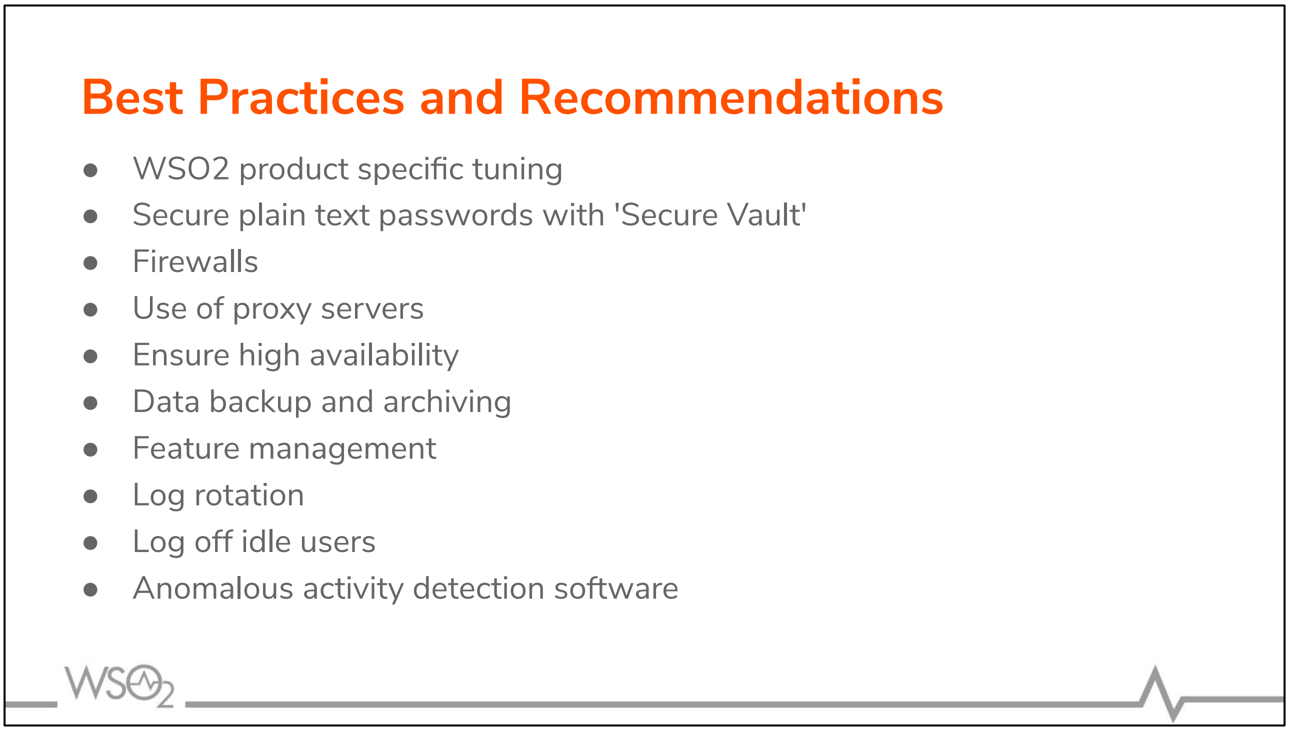
```
   <HostName>wso2.esb.com</HostName>
```

A properly configure logging system is a must for a production-grade server in order identify errors. security threads and usage patterns. WSO2 products use lo4j logging and this can be configured via the management console or <Porduct_Home>/repository/conf/log4j.properties file

The default database used by the registry as the persistent storage of WSO2 products is the embedded H2. This is less reliable and performant compared to standard databases like MySQL / Oracle when there is a large number of deployed artifacts. Therefore it is recommended to point the registry to a database like MySQL / Oracle/ MSSQL and also follow periodic backup procedures. When the registry is pointed to a remote database in this manner, multiple instances of products can boot up and run against the same registry configurations.

When there is a large user base, it is recommended to plug a user store. WSO2 products support the use of LDAP, Active directory and JDBC use stores.

WSO2 products support JMX monitoring by default and JMX monitoring helps you keep track of memory and CPU usages , Threads etc.



Best Practices and Recommendations

- WSO2 product specific tuning
- Secure plain text passwords with 'Secure Vault'
- Firewalls
- Use of proxy servers
- Ensure high availability
- Data backup and archiving
- Feature management
- Log rotation
- Log off idle users
- Anomalous activity detection software

WSO2 product documentation includes performance tuning parameters for each product. Tuning the servers based on these parameters is recommended for any production deployment.

You can use secure vault implementation of WSO2 products to encrypt plaintext passwords included in configuration files.

You can also enable firewalls and block the ports which are not used by server instances within the deployment.

Use of proxy servers to front deployments is another security measure take to prevent exposing the deployment directly to external entities.

Clustering is recommended for WSO2 product deployments in order to achieve high availability and failover.

Periodic database artifacts and configuration backups is a must for production deployments.

Compatible features can be installed on WSO2 products using the WSO2 Carbon Feature Manager when required. For example, WSO2 Identity server can be used as a Key Management Server by installing the Key Manager features.

Log rotation can be configured in the '<PRODUCT_HOME>/repository/conf/log4j.properties' file in order to manage log growth or the server.

Idle users impose security threats. Therefore idle timeouts can be specified to

log off users from the system.

Use programs like logcheck to detect security violations and anomalous

activities and notify periodically.

THANK YOU

wso2.com