Oracle® JRockit

JDK Release Notes

Release R28

E15066-46

October 2018

Oracle JRockit JDK Release Notes, Release R28

E15066-46

Primary Author: Trupthi NT

Contributors: Savija Vijayaraghavan

Contents

Preface

This document contains important release information about Oracle JRockit JDK R28.0.

About this Document

This document includes the following chapters:

- Changes in Supported Configurations in Oracle JRockit JDK R28, which lists the changes in the supported configurations for JRockit JDK R28.0 when compared with R27.6.6.
- New Features and Changes in Oracle JRockit JDK R28, which the new features and changes in JRockit JDK R28.0.
- Issues Resolved in Oracle JRockit JDK R28, which lists issues resolved in JRockit JDK R28.0.
- Known Issues in Oracle JRockit JDK R28, which lists issues known to exist in JRockit JDK R28.0.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc`.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` or visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` if you are hearing impaired.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1 Changes in Supported Configurations in Oracle JRockit JDK R28

This chapter lists the changes in the supported configurations for JRockit JDK R28.x when compared with R27.6.6.
The following are the changes in supported configurations:

- Java Version Updates
- Hardware Must Support Streaming SIMD Extensions (SSE) 2
- J2SE 1.4.2 and JVMPI Not Supported
- Itanium Platforms Not Supported

JRockit JVM R28.x is not supported on Windows 2000, as was the case with R27.

For up to date supported configuration information, see *Oracle Fusion Middleware Supported System Configurations* at: `http://www.oracle.com/technetwork/middleware/ias/downloads/fusion-certification-100350.html`.

1.1 Java Version Updates

The following table lists the Java versions supported by the various Oracle JRockit JDK R28 releases. For information about new features in each release, see the JDK 6 Release Notes.

Table 1-1 Java Versions Supported by the Oracle JRockit JDK R28.x

JRockit JDK R28.x Release	Supported J2SE 5.0 Update	Supported Java SE 6 Update
R28.3.20	-	Update 211
R28.3.19	-	Update 201
R28.3.18	-	Update 191
R28.3.17	-	Update 181
R28.3.16	-	Update 171
R28.3.15	-	Update 161
R28.3.14	-	Update 151
R28.3.13	-	Update 141
R28.3.12	-	Update 131
R28.3.11	-	Update 121
R28.3.10	-	Update 115
R28.3.9	-	Update 111
R28.3.8	-	Update 105

Table 1-1 (Cont.) Java Versions Supported by the Oracle JRockit JDK R28.x

JRockit JDK R28.x Release	Supported J2SE 5.0 Update	Supported Java SE 6 Update
R28.3.7	-	Update 101
R28.3.6	-	Update 95
R28.3.5	Update 81	Update 91
R28.3.4	Update 75	Update 85
R28.3.3	Update 71	Update 81
R28.3.2	Update 65	Update 75
R28.3.1	Update 61	Update 71
R28.2.9	Update 55	Update 65
R28.2.8	Update 51	Update 51
R28.2.7	Update 45	Update 45
R28.2.6	Update 41	Update 43
R28.2.5	Update 38	Update 37
R28.2.4	Update 36	Update 33
R28.2.3	Update 34	Update 31
R28.2.2	Update 32	Update 29
R28.2.1	Update 32	Update 29
R28.2	Update 32	Update 29
R28.1.5	Update 32	Update 29
R28.1.4	Update 30	Update 26
R28.1.3	Update 28	Update 24
R28.1.1	Update 26	Update 22
R28.1.0	Update 24	Update 20
R28.0.2	Update 24	Update 20
R28.0.1	Update 24	Update 20
R28.0.0	Update 22	Update 17

1.2 Hardware Must Support Streaming SIMD Extensions (SSE) 2

Oracle JRockit JDK R28.x does not support x87, the floating point extension to the x86 platform.

Hardware on which you intend to run the Oracle JRockit JVM must support SSE2 (Streaming SIMD Extensions): that is, Intel Pentium 4 or Pentium M; AMD Opteron or Athlon 64; or newer hardware.

1.3 J2SE 1.4.2 and JVMPI Not Supported

Oracle JRockit JDK R28.x does not support J2SE 1.4.2.

As a consequence of this change, JRockit JDK R28.x does not support JVMPI. Most tools partners use JVMTI, which continues to be supported.

1.4 Itanium Platforms Not Supported

Oracle JRockit JDK R28.x does not support the Itanium architecture.

Previous JRockit JDK releases are available and supported for Itanium platforms until all the dependent Oracle products reach end-of-life (EOL).

2

New Features and Changes in Oracle JRockit JDK R28

This chapter describes the new features and changes in Oracle JRockit R28.x releases.
For information about bug fixes and changes in Java SE 6 Updates, see the release notes for JDK at:

`http://www.oracle.com/technetwork/java/javase/overview-156328.html`

> **Note:**
>
> JRockit R28.3.20 is the last Critical Patch Update for JRockit and as noted in Oracle Fusion Middleware Lifetime Support Policy will reach End of Extended Support Life in December 2018. All customers are strongly encouraged to migrate to a later release before that date.

It contains the following topics:

- Changes in R28.0.1
- New Features and Changes in R28.0.0

2.1 Changes in R28.3.20

This section describes the changes in Oracle JRockit JDK R28.3.20:

Note:

JRockit R28.3.20 is the last Critical Patch Update for JRockit and as noted in Oracle Fusion Middleware Lifetime Support Policy will reach End of Extended Support Life in December 2018. All customers are strongly encouraged to migrate to a later release before that date.

- Upgraded to JDK 6u211
- Disabled All DES TLS Cipher Suites
- Removal of Several Symantec Root CAs
- Removal of Baltimore Cybertrust Code Signing CA
- Removal of SECOM Root Certificate
- Improved Validation of Class-Path Attribute in JAR File Manifest
- Improved Cipher Inputs

2.1.1 Upgraded to JDK 6u211

JRockit R28.3.20 is upgraded to JDK 6u211. For information about bug fixes and other changes in this JDK version, see the JDK 6 Release Notes.

2.1.2 Disabled All DES TLS Cipher Suites

security-libs/javax.net.ssl

DES-based TLS cipher suites are considered obsolete and should no longer be used. DES-based cipher suites have been deactivated by default in the SunJSSE implementation by adding the "DES" identifier to the `jdk.tls.disabledAlgorithms` security property. These cipher suites can be reactivated by removing "DES" from the `jdk.tls.disabledAlgorithms` security property in the `java.security` file or by dynamically calling the `Security.setProperty()` method. In both cases re-enabling DES must be followed by adding DES-based cipher suites to the enabled cipher suite list using the `SSLSocket.setEnabledCipherSuites()` or `SSLEngine.setEnabledCipherSuites()` methods.

Note that prior to this change, DES40_CBC (but not all DES) suites were disabled via the `jdk.tls.disabledAlgorithms` security property.

See JDK-8208350

2.1.3 Removal of Several Symantec Root CAs

security-libs/java.security

The following Symantec root certificates are no longer in use and have been removed:

- equifaxsecureca

  DN: OU=Equifax Secure Certificate Authority, O=Equifax, C=US
- equifaxsecureglobalebusinessca1

  DN: CN=Equifax Secure Global eBusiness CA-1, O=Equifax Secure Inc., C=US
- equifaxsecureebusinessca1

  DN: CN=Equifax Secure eBusiness CA-1, O=Equifax Secure Inc., C=US
- verisignclass1g3ca

  DN: CN=VeriSign Class 1 Public Primary Certification Authority - G3, OU="(c) 1999 VeriSign, Inc. - For authorized use only", OU=VeriSign Trust Network, O="VeriSign, Inc.", C=US
- verisignclass2g3ca

  DN: CN=VeriSign Class 2 Public Primary Certification Authority - G3, OU="(c) 1999 VeriSign, Inc. - For authorized use only", OU=VeriSign Trust Network, O="VeriSign, Inc.", C=US
- verisignclass1g2ca

  DN: OU=VeriSign Trust Network, OU="(c) 1998 VeriSign, Inc. - For authorized use only", OU=Class 1 Public Primary Certification Authority - G2, O="VeriSign, Inc.", C=US
- verisignclass1ca

  DN: OU=Class 1 Public Primary Certification Authority, O="VeriSign, Inc.", C=US

See JDK-8191031

2.1.4 Removal of Baltimore Cybertrust Code Signing CA

security-libs/java.security

The following Baltimore CyberTrust Code Signing root certificate is no longer in use and has been removed:

- baltimorecodesigningca

  DN: CN=Baltimore CyberTrust Code Signing Root, OU=CyberTrust, O=Baltimore, C=IE

See JDK-8189949

2.1.5 Removal of SECOM Root Certificate

security-libs/java.security

The following SECOM root certificate is no longer in use and has been removed:

- secomevrootca1

  DN: OU=Security Communication EV RootCA1, O="SECOM Trust Systems CO.,LTD.", C=JP

See JDK-8191844

2.1.6 Improved Validation of Class-Path Attribute in JAR File Manifest

core-libs

The JAR file specification states that URLs in the `Class-Path` manifest attribute must be relative, though this has not been enforced. To better conform to the JAR specification, absolute URLs (those that include a scheme) are now ignored. For JAR files not loaded from the file system, `Class-Path` entries navigating to a parent directory (using "../") are also ignored.

Applications depending on a JAR file loaded from an absolute URL element specified in Class-Path attribute may encounter a `ClassNotFoundException`. The historical behavior can be restored by setting a new system property, `jdk.net.URLClassPath.disableClassPathURLCheck` to `true`. Debugging info for Class-Path entries that are ignored can be printed to stderr by setting `-Djdk.net.URLClassPath.disableClassPathURLCheck=debug`.

2.1.7 Improved Cipher Inputs

security-libs/javax.crypto

The specification of `javax.crypto.CipherInputStream` has been clarified to indicate that this class may catch BadPaddingException and other exceptions thrown by failed integrity checks during decryption. These exceptions are not re-thrown, so the client may not be informed that integrity checks failed. Because of this behavior, this class may not be suitable for use with decryption in an authenticated mode of operation (e.g. GCM). Applications that require authenticated encryption can use the Cipher API directly as an alternative to using this class.

2.2 Changes in R28.3.19

This section describes the changes in Oracle JRockit JDK R28.3.19:

- Upgraded to JDK 6u201

2.2.1 Upgraded to JDK 6u201

JRockit R28.3.19 is upgraded to JDK 6u201. For information about bug fixes and other changes in this JDK version, see the JDK 6 Release Notes.

2.3 Changes in R28.3.18

This section describes the changes in Oracle JRockit JDK R28.3.18:

- Upgraded to JDK 6u191
- TLS Session Hash and Extended Master Secret Extension Support
- Enhanced KeyStore Mechanisms
- 3DES Cipher Suites Disabled
- Server-side HTTP-tunneled RMI Connections Disabled

- CipherOutputStream Usage
- System Property Controls the java.util.logging.FileHandler's MAX_LOCKS Limit

2.3.1 Upgraded to JDK 6u191

JRockit R28.3.18 is upgraded to JDK 6u191. For information about bug fixes and other changes in this JDK version, see the JDK 6 Release Notes.

2.3.2 TLS Session Hash and Extended Master Secret Extension Support

security-libs/javax.net.ssl

Support has been added for the TLS session hash and extended master secret extension (RFC 7627) in JDK JSSE provider. Note that in general, a server certificate change is restricted if endpoint identification is not enabled and the previous handshake is a session-resumption abbreviated initial handshake, unless the identities represented by both certificates can be regarded as the same. However, if the extension is enabled or negotiated, the server certificate changing restriction is not necessary and will be discarded accordingly. In case of compatibility issues, an application may disable negotiation of this extension by setting the System Property `jdk.tls.useExtendedMasterSecret` to `false` in the JDK. By setting the System Property `jdk.tls.allowLegacyResumption` to `false`, an application can reject abbreviated handshaking when the session hash and extended master secret extension are not negotiated. By setting the System Property `jdk.tls.allowLegacyMasterSecret` to `false`, an application can reject connections that do not support the session hash and extended master secret extension.

See JDK-8148421

2.3.3 Enhanced KeyStore Mechanisms

security-libs/javax.crypto

A new security property named `jceks.key.serialFilter` has been introduced. If this filter is configured, the JCEKS KeyStore uses it during the deserialization of the encrypted Key object stored inside a SecretKeyEntry. If it is not configured or if the filter result is UNDECIDED (for example, none of the patterns match), then the filter configured by `jdk.serialFilter` is consulted.

If the system property `jceks.key.serialFilter` is also supplied, it supersedes the security property value defined here.

The filter pattern uses the same format as `jdk.serialFilter`. The default pattern allows `java.lang.Enum, java.security.KeyRep, java.security.KeyRep$Type`, and `javax.crypto.spec.SecretKeySpec` but rejects all the others.

Customers storing a SecretKey that does not serialize to the above types must modify the filter to make the key extractable.

2.3.4 3DES Cipher Suites Disabled

security-libs/javax.net.ssl

To improve the strength of SSL/TLS connections, 3DES cipher suites have been disabled in SSL/TLS connections in the JDK via the `jdk.tls.disabledAlgorithms` Security Property.

2.3.5 Server-side HTTP-tunneled RMI Connections Disabled

core-libs/java.rmi

Server side HTTP-tunneled RMI connections have been disabled by default in this release. This behavior can be reverted by setting the runtime property `sun.rmi.server.disableIncomingHttp` property to `false`. Note, this should not be confused with the `sun.rmi.server.disableHttp` property, which disables HTTP-tunneling on the client side and is false by default.

2.3.6 CipherOutputStream Usage

security-libs/javax.crypto

The specification of `javax.crypto.CipherOutputStream` has been clarified to indicate that this class catches BadPaddingException and other exceptions thrown by failed integrity checks during decryption. These exceptions are not re-thrown, so the client is not informed that integrity checks have failed. Because of this behavior, this class may not be suitable for use with decryption in an authenticated mode of operation (for example, GCM) if the application requires explicit notification when authentication fails. These applications can use the Cipher API directly as an alternative to using this class.

2.3.7 System Property Controls the java.util.logging.FileHandler's MAX_LOCKS Limit

core-libs/java.util.logging

A new JDK implementation specific system property `jdk.internal.FileHandlerLogging.maxLocks` has been introduced to control the `java.util.logging.FileHandler` `MAX_LOCKS` limit. The default value of the current `MAX_LOCKS (100)` is retained if this new system property is not set or an invalid value is provided to the property. Valid values for this property are integers ranging from 1 to Integer `MAX_VALUE-1`.

See JDK-8153955

2.4 Changes in R28.3.17

This section describes the changes in Oracle JRockit JDK R28.3.17:

- Upgraded to JDK 6u181
- Support DHE Sizes Up To 8192-bits and DSA Sizes Up To 3072-bits
- Support SHA224withDSA and SHA256withDSA in the SunJSSE provider
- Add Additional IDL Stub Type Checks To org.omg.CORBA.ORBstring_to_object Method

- RSA Public Key Validation
- Restrict Diffie-Hellman Keys Less Than 1024 Bits
- Provider Default Key Size is Updated
- Stricter Key Generation
- Unlimited Cryptography Enabled by Default
- Disable Exportable Cipher Suites
- Disable JARs Signed with DSA Keys Less Than 1024 Bits
- Added wsimport Tool Command Line Option ???disableXmlSecurity
- JMX Connections Need Deserialization Filters

2.4.1 Upgraded to JDK 6u181

JRockit R28.3.17 is upgraded to JDK 6u181. For information about bug fixes and other changes in this JDK version, see the JDK 6 Release Notes.

2.4.2 Support DHE Sizes Up To 8192-bits and DSA Sizes Up To 3072-bits

security-libs/javax.crypto

Enhance the JDK security providers to support 3072-bit DiffieHellman and DSA parameters generation, pre-computed DiffieHellman parameters up to 8192 bits and pre-computed DSA parameters up to 3072 bits.

See JDK-8072452

2.4.3 Support SHA224withDSA and SHA256withDSA in the SunJSSE provider

security-libs/javax.net.ssl

The SHA224withDSA and SHA256withDSA algorithms are now supported in the TLS 1.2 "signature_algorithms" extension in the SunJSSE provider. Note that this extension does not apply to TLS 1.1 and previous versions.

See JDK–8049321

2.4.4 Add Additional IDL Stub Type Checks To org.omg.CORBA.ORBstring_to_object Method

other-libs/corba

Applications that either explicitly or implicitly call `org.omg.CORBA.ORB.string_to_object`, and wish to ensure the integrity of the IDL stub type involved in the `ORB::string_to_object` call flow, should specify additional IDL stub type checking. This is an "opt in" feature and is not enabled by default.

To take advantage of the additional type checking, the list of valid IDL interface class names of IDL stub classes is configured by one of the following:

- Specifying the security property `com.sun.CORBA.ORBIorTypeCheckRegistryFilter` located in the file `conf/security/java.security` in Java SE 9 or in `jre/lib/security/java.security` in Java SE 8 and earlier.
- Specifying the system property `com.sun.CORBA.ORBIorTypeCheckRegistryFilter` with the list of classes. If the system property is set, its value overrides the corresponding property defined in the `java.security` configuration.

If the `com.sun.CORBA.ORBIorTypeCheckRegistryFilter` property is not set, the type checking is only performed against a set of class names of the IDL interface types corresponding to the built-in IDL stub classes.

2.4.5 RSA Public Key Validation

security-libs/javax.crypto

In R28.3.17, the RSA implementation in the SunRsaSign provider will reject any RSA public key that has an exponent that is not in the valid range as defined by PKCS#1 version 2.2. This change will affect JSSE connections as well as applications built on JCE.

2.4.6 Restrict Diffie-Hellman Keys Less Than 1024 Bits

security-libs/javax.net.ssl

Diffie-Hellman keys less than 1024 bits are considered too weak to use in practice and should be restricted by default in SSL/TLS/DTLS connections. Accordingly, Diffie-Hellman keys less than 1024 bits have been disabled by default by adding `DH keySize < 1024` to the `jdk.tls.disabledAlgorithms` security property in the `java.security` file. Although it is not recommended, administrators can update the security property (`jdk.tls.disabledAlgorithms`) and permit smaller key sizes (for example, by setting `DH keySize < 768`).

2.4.7 Provider Default Key Size is Updated

security-libs/javax.crypto

This change updates the JDK providers to use 2048 bits as the default key size for DSA instead of 1024 bits when applications have not explicitly initialized the `java.security.KeyPairGenerator` and `java.security.AlgorithmParameterGenerator` objects with a key size.

If compatibility issues arise, existing applications can set the system property `jdk.security.defaultKeySize` introduced in JDK-8181048 with the algorithm and its desired default key size.

2.4.8 Stricter Key Generation

security-libs/javax.crypto

The `generateSecret(String)` method has been mostly disabled in the `javax.crypto.KeyAgreement` services of the SUN and SunPKCS11 providers. Invoking this method for these providers will result in a `NoSuchAlgorithmException` for most algorithm string arguments. The previous behavior of this method can be re-enabled by setting the value of the `jdk.crypto.KeyAgreement.legacyKDF` system property to `true` (case insensitive). Re-enabling this method by setting this system property is not recommended.

2.4.9 Unlimited Cryptography Enabled by Default

security-libs/javax.crypto

The JDK uses the Java Cryptography Extension (JCE) Jurisdiction Policy files to configure cryptographic algorithm restrictions. Previously, the Policy files in the JDK placed limits on various algorithms. This release ships with both the limited and unlimited jurisdiction policy files, with unlimited being the default. The behavior can be controlled via the new crypto.policy Security property found in the `<java-home>/lib/java.security` file. Refer to that file for more information on this property.

See JDK-8170157

2.4.10 Disable Exportable Cipher Suites

security-libs/javax.net.ssl

To improve the strength of SSL/TLS connections, exportable cipher suites have been disabled in SSL/TLS connections in the JDK by the `jdk.tls.disabledAlgorithms` Security Property.

See JDK-8163237

2.4.11 Disable JARs Signed with DSA Keys Less Than 1024 Bits

security-libs/java.security

DSA keys less than 1024 bits have been added to the `jdk.jar.disabledAlgorithms` Security property in the `java.security` file. This property contains a list of disabled algorithms and key sizes for signed JAR files. If a signed JAR file uses a disabled algorithm or key size less than the minimum length, signature verification operations will ignore the signature and treat the JAR as if it were unsigned. This can potentially occur in the following types of applications that use signed JAR files:

1. Applets or Web Start Applications.
2. Standalone or Server Applications run with a SecurityManager enabled and that are configured with a policy file that grants permissions based on the code signer(s) of the JAR file.

Running `jarsigner -verify -verbose` on a JAR file signed with a weak algorithm or key will print more information about the disabled algorithm or key.

For example, to check a JAR file named `test.jar`, use this command : `jarsigner -verify -verbose test.jar`

If the file in this example was signed with a weak key such as 512 bit DSA, this output would be seen:

```
- Signed by "CN=weak_signer"
    Digest algorithm: SHA1
    Signature algorithm: SHA1withDSA, 512-bit key (weak)
```

To address the issue, the JAR file will need to be re-signed with a stronger key size. Alternatively, the restrictions can be reverted by removing the applicable weak algorithms or key sizes from the jdk.jar.disabledAlgorithms security property; however, this option is not recommended. Before re-signing affected JARs, the existing signature(s) should be removed from the JAR file. This can be done with the zip utility, as follows:

```
 zip -d test.jar 'META-INF/*.SF' 'META-INF/*.RSA' 'META-INF/*.DSA'
```

Periodically check the Oracle JRE and JDK Cryptographic Roadmap at http://java.com/cryptoroadmap for planned restrictions to signed JARs and other security components.

2.4.12 Added wsimport Tool Command Line Option ??? disableXmlSecurity

xml/jax-ws

The `wsimport` tool has been changed to disallow DTDs in Web Service descriptions, specifically:

- DOCTYPE declaration is disallowed in documents
- External general entities are not included by default
- External parameter entities are not included by default
- External DTDs are completely ignored

To restore the previous behavior:

- Set the System property `com.sun.xml.internal.ws.disableXmlSecurity` to `true`
- Use the `wsimport` tool command line option `???disableXmlSecurity`

2.4.13 JMX Connections Need Deserialization Filters

core-svc/javax.management

New public attributes, `RMIConnectorServer.CREDENTIALS_FILTER_PATTERN` and `RMIConnectorServer.SERIAL_FILTER_PATTERN` have been added to `RMIConnectorServer.java`. With these new attributes, users can specify the deserialization filter pattern strings to be used while making a `RMIServer.newClient()` remote call and while sending deserializing parameters over RMI to server respectively.

The user can also provide a filter pattern string to the default agent via `management.properties`. As a result, a new attribute is added to `management.properties`.

Existing attribute `RMIConnectorServer.CREDENTIAL_TYPES` is superseded by `RMIConnectorServer.CREDENTIALS_FILTER_PATTERN` and has been removed.

2.5 Changes in R28.3.16

This section describes the changes in Oracle JRockit JDK R28.3.16:

- Upgraded to JDK 6u171
- Support DHE Sizes Up To 8192-bits and DSA Sizes Up To 3072-bits
- Refactor Existing Providers to Refer to the Same Constants for Default Values for Key Length
- Collections Use Serialization Filter to Limit Array Sizes
- Default Timeouts Have Changed for FTP URL Handler
- New Defaults for DSA Keys in Jarsigner and Keytool

2.5.1 Upgraded to JDK 6u171

JRockit R28.3.16 is upgraded to JDK 6u171. For information about bug fixes and other changes in this JDK version, see the JDK 6 Release Notes.

2.5.2 Support DHE Sizes Up To 8192-bits and DSA Sizes Up To 3072-bits

security-libs/javax.crypto

Enhance the JDK security providers to support 3072-bit DiffieHellman and DSA parameters generation, pre-computed DiffieHellman parameters up to 8192 bits and pre-computed DSA parameters up to 3072 bits.

See JDK-8072452

2.5.3 Refactor Existing Providers to Refer to the Same Constants for Default Values for Key Length

security-libs/java.security

Two important changes have been made for this issue:

- A new system property has been introduced that allows users to configure the default key size used by the JDK provider implementations of `KeyPairGenerator` and `AlgorithmParameterGenerator`. This property is named `"jdk.security.defaultKeySize"` and the value of this property is a list of comma-separated entries. Each entry consists of a case-insensitive algorithm name and the corresponding default key size (in decimal) separated by ":". In addition, white space is ignored.

  By default, this property does not have a value, and JDK providers use their own default values. Entries containing an unrecognized algorithm name will be ignored. If the specified default key size is not a parseable decimal integer, that entry will be ignored as well.

- The `DSA KeyPairGenerator` implementation of the SUN provider no longer implements `java.security.interfaces.DSAKeyPairGenerator`. Applications which cast the SUN provider's `DSA KeyPairGenerator` object to a `java.security.interfaces`. The `DSAKeyPairGenerator` can set the system property `"jdk.security.legacyDSAKeyPairGenerator"`. If the value of this property is "true", the SUN provider will return a `DSA KeyPairGenerator` object which implements the `java.security.interfaces`. The `DSAKeyPairGenerator` interface. This legacy implementation will use the same default value as specified by the javadoc in the interface.

  By default, this property will not have a value, and the SUN provider will return a `DSAKeyPairGenerator` object which does not implement the aforementioned interface and thus can determine its own provider-specific default value as stated in the `java.security.KeyPairGenerator` class or by the `"jdk.security.defaultKeySize"` system property if set.

2.5.4 Collections Use Serialization Filter to Limit Array Sizes

core-libs/java.util:collections

Deserialization of certain collection instances will cause arrays to be allocated. The `ObjectInputFilter.checkInput()` method is now called prior to allocation of these arrays.

- Deserializing instances of `ArrayDeque`, `ArrayList`, `IdentityHashMap`, `PriorityQueue`, `java.util.concurrent.CopyOnWriteArrayList`, and the immutable collections (as returned by List.of, Set.of, and Map.of) will call `checkInput()` with a FilterInfo instance whose serialClass() method returns `Object[].class`.
- Deserializing instances of `HashMap`, `HashSet`, `Hashtable`, and `Properties` will call `checkInput()` with a FilterInfo instance whose serialClass() method returns `Map.Entry[].class`.

In both cases, the `FilterInfo.arrayLength()` method returns the actual length of the array to be allocated. The exact circumstances under which the serialization filter is called, and with what information, is subject to change in future releases.

2.5.5 Default Timeouts Have Changed for FTP URL Handler

core-libs/java.net

Timeouts used by the FTP URL protocol handler have been changed from infinite to 5 minutes. This will result in an IOException from connect and read operations if the FTP server is unresponsive. For example, new URL `("ftp://example.com").openStream().read()`, will fail with `java.net.SocketTimeoutException` in case a connection or reading could not be completed within 5 minutes.

To revert this behaviour to that of previous releases, the following system properties may be used, `sun.net.client.defaultReadTimeout=0`, `sun.net.client.defaultConnectTimeout=0`

2.5.6 New Defaults for DSA Keys in Jarsigner and Keytool

security-libs/java.security

For DSA keys, the default signature algorithm for `keytool` and `jarsigner` has changed from `SHA1withDSA` to `SHA256withDSA` and the default key size for keytool has changed from 1024 bits to 2048 bits.

Users who want to revert to the previous behavior can use the `-sigalg` option of `keytool` and `jarsigner` and specify `SHA1withDSA` and the `-keysize` option of `keytool` and specify 1024.

There are a few potential compatibility risks associated with this change:

- If you have a script that uses the default key size of keytool to generate a DSA keypair but then subsequently specifies a specific signature algorithm.

  For example:

  ```
  keytool -genkeypair -keyalg DSA -keystore keystore -alias mykey ...
  keytool -certreq -sigalg SHA1withDSA -keystore keystore -alias mykey ...
  ```

  will fail with one of the following exceptions, because the new 2048-bit keysize default is too strong for SHA1withDSA:

  ```
  keytool error: java.security.InvalidKeyException: The security strength of SHA-1
  digest algorithm is not sufficient for this key size
  keytool error: java.security.InvalidKeyException: DSA key must be at most 1024
  bits
  ```

  The workaround is to remove the `-sigalg` option and use the stronger `SHA256withDSA` default or, at your own risk, use the `-keysize` option of `keytool` to specify a smaller key size (1024).

- If you use `jarsigner` to sign JARs with the new defaults, previous versions (than this release) of JDK 6 and 7 do not support the stronger defaults and will not be able to verify the JAR. `jarsigner -verify` on an earlier release of JDK 6 or 7 will output the following error:

  ```
  jar is unsigned. (signatures missing or not parsable)
  ```

  If you add `-J-Djava.security.debug=jar` to the `jarsigner` command line, the cause will be output:

  ```
  jar: processEntry caught: java.security.NoSuchAlgorithmException: SHA256withDSA
  Signature not available
  ```

  If compatibility with earlier releases is important, you can, at your own risk, use the `-sigalg` option of `jarsigner` and specify the weaker `SHA1withDSA` algorithm.

- If you use a `PKCS11` keystore, the SunPKCS11 provider does not support the `SHA256withDSA` algorithm. `jarsigner` and some `keytool` commands may fail with the following exception if PKCS11 is specified with the `-storetype` option. For example:

  ```
  keytool error: java.security.InvalidKeyException: No installed provider supports
  this key: sun.security.pkcs11.P11Key$P11PrivateKey
  ```

  A similar error may occur if you are using NSS with the SunPKCS11 provider. The workaround is to use the `-sigalg` option of `keytool` and specify `SHA1withDSA`.

See JDK-8057810

2.6 Changes in R28.3.15

This section describes the changes in Oracle JRockit JDK R28.3.15:

- Upgraded to JDK 6u161
- Improved Algorithm Constraints Checking
- JMX Diagnostic Improvements
- Message Digest Algorithm for jarsigner -tsadigestalg Option Now Defaults to SHA-256

2.6.1 Upgraded to JDK 6u161

JRockit R28.3.15 is upgraded to JDK 6u161. For information about bug fixes and other changes in this JDK version, see the JDK 6 Release Notes.

2.6.2 Improved Algorithm Constraints Checking

security-libs/java.security

With the need to restrict weak algorithms usage in situations where they are most vulnerable, additional features have been added when configuring the `jdk.certpath.disabledAlgorithms` and `jdk.jar.disabledAlgorithms` security properties in the `java.security` file.

jdk.certpath.disabledAlgorithms

The `certpath` property has seen the most change. Previously it was limited to two constraint types – either a full disabling of an algorithm by name or a full disabling of an algorithm by the key size when checking certificates, certificate chains, and certificate signatures. This creates configurations that are absolute and lack flexibility in their usage. Three new constraints were added to give more flexibility in allowing and rejecting certificates.

- `jdkCA` examines the certificate chain termination with regard to the `cacerts` file. In the case of `SHA1 jdkCA`. SHA1's usage is checked through the certificate chain, but the chain must terminate at a marked trust anchor in the `cacerts keystore` to be rejected. This is useful for organizations that have their own private CA that trust using SHA1 with their trust anchor, but want to block certificate chains anchored by a public CA from using SHA1.
- `denyAfter` checks if the given date is before the current date or the `PKIXParameter` date. In the case of `SHA1 denyAfter 2018-01-01`, before 2018 a certificate with SHA1 can be used, but after that date, the certificate is rejected. This can be used for a policy across an organization that is phasing out an algorithm with a drop-dead date. For signed JAR files, the date is compared against the TSA timestamp. The date is specified in GMT.
- `usage` examines the specified algorithm for a specified usage. This can be used when disabling an algorithm for all usages is not practical. There are three usages that can be specified:
  - `TLSServer` restricts the algorithm in TLS server certificate chains when server authentication is performed as a client.

- `TLSClient` restricts the algorithm in TLS client certificate chains when client authentication is performed as a server.
- `SignedJAR` restricts the algorithms in certificates in signed JAR files. The usage type follows the keyword and more than one usage type can be specified with a whitespace delimiter. For example, `SHA1 usage TLSServer TLSClient` would disallow SHA1 certificates for TLSServer and TLSClient operations, but SignedJars would be allowed.

All of these constraints can be combined to constrain an algorithm when delimited by `&`. For example, to disable SHA1 certificate chains that terminate at marked trust anchors only for TLSServer operations, the constraint would be `SHA1 jdkCA & usage TLSServer`.

jdk.jar.disabledAlgorithms

One additional constraint was added to this .jar property to restrict JAR manifest algorithms.

`denyAfter` checks algorithm constraints on manifest digest algorithms inside a signed JAR file. The date given in the constraint is compared against the TSA timestamp on the signed JAR file. If there is no timestamp or the timestamp is on or after the specified date, the signed JAR file is treated as unsigned. If the timestamp is before the specified date, the .jar operates as a signed JAR file. The syntax for restricting SHA1 in JAR files signed after January 1st 2018 is: `SHA1 denyAfter 2018-01-01`. The syntax is the same as that for the certpath property, however certificate checking will not be performed by this property.

See JDK-8176536

2.6.3 JMX Diagnostic Improvements

core-svc/java.lang.management

The `com.sun.management.HotSpotDiagnostic::dumpHeap` API is modified to throw `IllegalArgumentException` if the supplied file name does not end with `.hprof` suffix. Existing applications which do not provide a file name ending with the `.hprof` extension will fail with `IllegalArgumentException`. In that case, applications can either choose to handle the exception or restore old behavior by setting system property `jdk.management.heapdump.allowAnyFileSuffix` to true.

2.6.4 Message Digest Algorithm for jarsigner -tsadigestalg Option Now Defaults to SHA-256

security-libs/java.security

If not specified, the message digest algorithm for the `-tsadigestalg` option of jarsigner defaults to SHA-256 (previously it was SHA-1). The `-tsadigestalg` option specifies the message digest algorithm that is used to generate the message imprint to be sent to the TSA server.

See JDK-8177674

2.7 Changes in R28.3.14

This section describes the changes in Oracle JRockit JDK R28.3.14:

- Upgraded to JDK 6u151
- IANA Data 2016j
- MD5 signature verification added to the Security Property jdk.jar.disabled Algorithms
- New system property to control caching for HTTP SPNEGO connection
- New System Property to Control Caching for HTTP NTLM Connection

2.7.1 Upgraded to JDK 6u151

JRockit R28.3.14 is upgraded to JDK 6u151. For information about bug fixes and other changes in this JDK version, see the JDK 6 Release Notes.

2.7.2 IANA Data 2016j

JDK R28.3.14 contains IANA time zone data version 2016j. For more information, refer to Timezone Data Versions in the JRE Software.

See JDK-8170316

2.7.3 MD5 signature verification added to the Security Property jdk.jar.disabled Algorithms

security-libs/java.security

This JDK release introduces a new restriction on how MD5 signed JAR files are verified. If the signed JAR file uses MD5, signature verification operations ignore the signature and treat the JAR as if it were unsigned. This can potentially occur in the following types of applications that use signed JAR files:

- Applets or Web Start Applications
- Standalone or Server Applications that are run with a Security Manager enabled and are configured with a policy file that grants permissions based on the code signer of the JAR.

The list of disabled algorithms is controlled using the security property, `jdk.jar.disabled Algorithms`, in the java.security file. This property contains a list of disabled algorithms and key sizes for cryptographically signed JAR files.

To check if a weak algorithm or key was used to sign a JAR file, one can use the `jarsigner` binary that ships with this JDK. Running `jarsigner -verify` on a JAR file signed with a weak algorithm or key prints more information about the disabled algorithm or key.

For example, to check a JAR file named test.jar, use the following command :`jarsigner -verify test.jar`

If the file in this example was signed with a weak signature algorithm like MD5withRSA, this output would be displayed:

"The jar is treated as unsigned, because it is signed with a weak algorithm that is now disabled. Re-run jarsigner with the -verbose option for more details."

More details can be seen with the verbose option: `jarsigner -verify -verbose test.jar`

The following output would be displayed:

```
- Signed by "CN=weak_signer"
    Digest algorithm: MD5 (weak)
    Signature algorithm: MD5withRSA (weak), 512-bit key (weak)
  Timestamped by "CN=strong_tsa" on Mon Sep 26 08:59:39 CST 2016
    Timestamp digest algorithm: SHA-256
    Timestamp signature algorithm: SHA256withRSA, 2048-bit key
```

To address the issue, the JAR file must be re-signed with a stronger algorithm or key size. Alternatively, the restrictions can be reverted by removing the applicable weak algorithms or key sizes from the `jdk.jar.disabled Algorithms` security property. However, this option is not recommended. Before re-signing affected JARs, the existing signature must be removed from the JAR file. This can be done with the `zip` utility, as follows:

```
zip -d test.jar 'META-INF/.SF' 'META-INF/.RSA' 'META-INF/*.DSA'
```

Oracle recommends that you periodically check the Oracle JRE and JDK Cryptographic Roadmap at http://java.com/cryptoroadmap for planned restrictions to signed JARs and other security components.

JDK-8171121

2.7.4 New system property to control caching for HTTP SPNEGO connection

core-libs/java.net

A new JDK implementation-specific system property to control caching for HTTP SPNEGO (Negotiate/Kerberos) connections is introduced. Caching for HTTP SPNEGO connections is enabled by default. There is no behavior change if the property is not explicitly specified.

When connecting to an HTTP server which uses SPNEGO to negotiate authentication, and when connection and authentication with the server is successful, the authentication information will then be cached and reused for further connections to the same server. In addition, connecting to an HTTP server using SPNEGO usually involves keeping the underlying connection alive and reusing it for further requests to the same server. In some applications, it is recommended to disable all caching for the HTTP SPNEGO (Negotiate/Kerberos) protocol to force requesting new authentication with each new requests to the server.

With this fix, a new system property that allows control of the caching policy for HTTP SPNEGO connections is now provided. If `jdk.spnego.cache` is defined and evaluates to false, then all caching is disabled for HTTP SPNEGO connections. Setting this system property to false may however result in undesirable side effects:

- Performance of HTTP SPNEGO connections may be severely impacted as the connection must be re-authenticated with each new request, requiring several communication exchanges with the server.
- Credentials will need to be obtained again for each new requests, which, depending on whether transparent authentication is available or not, and depending on the global Authenticator implementation, may result in a popup asking the user for credentials for every new request.

JDK-8170814

2.7.5 New System Property to Control Caching for HTTP NTLM Connection

core-libs/java.net

A new JDK implementation-specific system property to control caching for HTTP NTLM connection is introduced. Caching for HTTP NTLM connection remains enabled by default. So, if the property is not explicitly specified, there is no behavior change.

On some platforms, the HTTP NTLM implementation in the JDK can support transparent authentication, where the system user credentials are used at the system level. When transparent authentication is not available or unsuccessful, the JDK only supports getting credentials from a global authenticator. If connection to the server is successful, the authentication information is cached and reused for further connections to the same server. In addition, connecting to an HTTP NTLM server usually involves keeping the underlying connection alive and reusing it for further requests to the same server. In some applications, it is essential to disable all caching for the HTTP NTLM protocol to force requesting new authentication with each new request to the server.

With this fix, a new system property that allows control of the caching policy for HTTP NTLM connections is provided. If `jdk.ntlm.cache` is defined and evaluates to false, then all caching is disabled for HTTP NTLM connections. Setting this system property to false, however, results in undesirable side effects:

- Performance of HTTP NTLM connections is severely impacted as the connection is re-authenticated with each new request, requiring several communication exchanges with the server.
- Credentials must be obtained again for each new request, which, depending on whether transparent authentication is available or not, and depending on the global Authenticator implementation, results in a popup asking for credentials for every new request.

JDK-8163520

2.8 Changes in R28.3.13

This section describes the changes in Oracle JRockit JDK R28.3.13:

- Upgraded to JDK 6u141
- IANA Data 2016i
- Improved protection for JNDI remote class loading
- jarsigner -verbose -verify should print the algorithms used to sign the jar
- Added security property to configure XML Signature secure validation mode

- Serialization Filter Configuration
- RMI Better constraint checking
- Add mechanism to allow non default root CAs to not be subject to algorithm restrictions
- New --allow-script-in-comments option for javadoc
- Increase the minimum key length to 1024 for XML Signatures
- Make 3DES a legacy algorithm in the JSSE provider
- Improve the default strength of elliptic curve cryptography in JDK
- Restrict certificates with DSA keys less than 1024 bits
- Add TLS v1.1 and v1.2 to the client list of default-enabled protocols
- More checks added to DER encoding parsing code
- Additional access restrictions for URLClassLoader.newInstance

2.8.1 Upgraded to JDK 6u141

JRockit R28.3.13 is upgraded to JDK 6u141. For information about bug fixes and other changes in this JDK version, see the JDK 6 Release Notes.

2.8.2 IANA Data 2016i

JDK R28.3.13 contains IANA time zone data version 2016i. For more information, refer to Timezone Data Versions in the JRE Software

2.8.3 Improved protection for JNDI remote class loading

core-libs/javax.naming

Remote class loading via JNDI object factories stored in naming and directory services, is disabled by default. To enable remote class loading by the RMI Registry or COS Naming service provider, set the following system property to the string true", as appropriate:

```
com.sun.jndi.rmi.object.trustURLCodebase
com.sun.jndi.cosnaming.object.trustURLCodebase
```

2.8.4 jarsigner -verbose -verify should print the algorithms used to sign the jar

security-libs/java.security

The jarsigner tool has been enhanced to show details of the algorithms and keys used to generate a signed JAR file and will also provide an indication if any of them are considered weak.

Specifically, when `jarsigner -verify -verbose filename.jar` is called, a separate section is printed out showing information of the signature and timestamp (if it exists) inside the signed JAR file, even if it is treated as unsigned for various reasons. If any algorithm or key used is considered weak, as specified in the Security property `jdk.jar.disabled` algorithms, it will be labeled with "(weak)".

For example:

```
- Signed by "CN=weak_signer"
    Digest algorithm: MD2 (weak)
    Signature algorithm: MD2withRSA (weak), 512-bit key (weak)
  Timestamped by "CN=strong_tsa" on Mon Sep 26 08:59:39 CST 2016
    Timestamp digest algorithm: SHA-256
    Timestamp signature algorithm: SHA256withRSA, 2048-bit key
``
```

See JDK-8163304

2.8.5 Added security property to configure XML Signature secure validation mode

security-libs/javax.xml.crypto

A new security property named `jdk.xml.dsig.secureValidationPolicy` has been added that allows you to configure the individual restrictions that are enforced when the secure validation mode of XML Signature is enabled. The default value for this property in the `java.security` configuration file is:

```
jdk.xml.dsig.secureValidationPolicy=\
    disallowAlg http://www.w3.org/TR/1999/REC-xslt-19991116,\
    disallowAlg http://www.w3.org/2001/04/xmldsig-more#rsa-md5,\
    disallowAlg http://www.w3.org/2001/04/xmldsig-more#hmac-md5,\
    disallowAlg http://www.w3.org/2001/04/xmldsig-more#md5,\
    maxTransforms 5,\
    maxReferences 30,\
    disallowReferenceUriSchemes file http https,\
    noDuplicateIds,\
    noRetrievalMethodLoops
```

Refer to the definition of the property in the `java.security` file for more information.

2.8.6 Serialization Filter Configuration

core-libs/java.io:serialization

Serialization Filtering introduces a new mechanism which allows incoming streams of object-serialization data to be filtered in order to improve both security and robustness. Every ObjectInputStream applies a filter, if configured, to the stream contents during deserialization. Filters are set using either a system property or a configured security property. The value of the `jdk.serialFilter` patterns are described in JEP 290 Serialization Filtering and in `<JRE>/lib/security/java.security`. Filter actions are logged to the `java.io.serialization` logger, if enabled.

2.8.7 RMI Better constraint checking

core-libs/java.rmi

RMI Registry and Distributed Garbage Collection use the mechanisms of JEP 290 Serialization Filtering to improve service robustness. RMI Registry and DGC implement built-in white-list filters for the typical classes expected to be used with each service. Additional filter patterns can be configured using either a system property or a security property. The `sun.rmi.registry.registryFilter` and `sun.rmi.transport.dgcFilter` property pattern syntax is described in `JEP 290` and in `<JRE>/lib/security/java.security`.

2.8.8 Add mechanism to allow non default root CAs to not be subject to algorithm restrictions

security-libs

In the `java.security` file, an additional constraint named `jdkCA` is added to the `jdk.certpath.disabledAlgorithms` property. This constraint prohibits the specified algorithm only if the algorithm is used in a certificate chain that terminates at a marked trust anchor in the `lib/security/cacerts keystore`. If the `jdkCA` constraint is not set, then all chains using the specified algorithm are restricted. `jdkCA` may only be used once in a `DisabledAlgorithm` expression.

Example: To apply this constraint to SHA-1 certificates, include the following:

```
SHA1 jdkCA
```

2.8.9 New --allow-script-in-comments option for javadoc

tools/javadoc(tool)

The javadoc tool will now reject any occurrences of JavaScript code in the javadoc documentation comments and command-line options, unless the command-line option, `--allow-script-in-comments` is specified.

With the `--allow-script-in-comments` option, the javadoc tool will preserve JavaScript code in documentation comments and command-line options. An error will be given by the javadoc tool if JavaScript code is found and the command-line option is not set.

2.8.10 Increase the minimum key length to 1024 for XML Signatures

security-libs/javax.xml.crypto

The secure validation mode of the XML Signature implementation has been enhanced to restrict RSA and DSA keys less than 1024 bits by default as they are no longer secure enough for digital signatures. Additionally, a new security property named `jdk.xml.dsig.SecureValidationPolicy` has been added to the `java.security` file and can be used to control the different restrictions enforced when the secure validation mode is enabled.

The secure validation mode is enabled either by setting the xml signature property `org.jcp.xml.dsig.secureValidation` to true with the `javax.xml.crypto.XMLCryptoContext.setProperty` method, or by running the code with a `SecurityManager`.

If an XML Signature is generated or validated with a weak RSA or DSA key, an XMLSignatureException will be thrown with the message RSA keys less than 1024 bits are forbidden when secure validation is enabled" or "DSA keys less than 1024 bits are forbidden when secure validation is enabled"

2.8.11 Make 3DES a legacy algorithm in the JSSE provider

security-libs/javax.net.ssl

For SSL/TLS/DTLS protocols, the security strength of 3DES cipher suites is not sufficient for persistent connections. By adding `3DES_EDE_CBC` to the `jdk.tls.legacyAlgorithms` security property by default in JDK, 3DES cipher suites will not be negotiated unless there are no other candidates during the establishing of SSL/TLS/DTLS connections.

At their own risk, applications can update this restriction in the security property (`jdk.tls.legacyAlgorithms`) if 3DES cipher suites are really preferred.

2.8.12 Improve the default strength of elliptic curve cryptography in JDK

security-libs/javax.net.ssl

To improve the default strength of elliptic curve cryptography, elliptic curve keys less than 224 bits have been deactivated in certification path processing (via the `jdk.certpath.disabledAlgorithms`, Security Property) and SSL/TLS/DTLS connections (via the `jdk.tls.disabledAlgorithms` Security Property) in JDK. Applications can update this restriction in the Security Properties and permit smaller key sizes if really needed (for example, EC keySize < 192).

Elliptic curves less than 256 bits are removed from the SSL/TLS/DTLS implementation in JDK. The new System Property, `jdk.tls.namedGroups`, defines a list of enabled named curves for EC cipher suites in order of preference. If an application needs to customize the default enabled EC curves or the curves preference, update the System Property accordingly. For example:

```
jdk.tls.namedGroups="secp256r1, secp384r1, secp521r1
```

Note that the default enabled or customized EC curves follow the algorithm constraints. For example, the customized EC curves cannot re-activate the disabled EC keys defined by the Java Security Properties

2.8.13 Restrict certificates with DSA keys less than 1024 bits

security-libs/java.security

DSA keys less than 1024 bits are not strong enough and should be restricted in certification path building and validation. Accordingly, DSA keys less than 1024 bits have been deactivated by default by adding “DSA keySize < 1024" to the `jdk.certpath.disabledAlgorithms` security property. Applications can update this restriction in the security property (`jdk.certpath.disabledAlgorithms`) and permit smaller key sizes if really needed (for example, "DSA keySize < 768")

2.8.14 Add TLS v1.1 and v1.2 to the client list of default-enabled protocols

security-libs/javax.net.ssl

TLSv1.2 and TLSv1.1 are now enabled by default on the TLS client end-points. This is similar behavior to what already happens in JDK 8 releases.

See details from crypto roadmap for more details.

2.8.15 More checks added to DER encoding parsing code

security-libs

More checks are added to the DER encoding parsing code to catch various encoding errors. In addition, signatures which contain constructed indefinite length encoding will now lead to IOException during parsing. Note that signatures generated using JDK default providers are not affected by this change.

2.8.16 Additional access restrictions for URLClassLoader.newInstance

core-libs/java.net

Class loaders created by the `java.net.URLClassLoader.newInstance` methods can be used to load classes from a list of given URLs. If the calling code does not have access to one or more of the URLs, and the URL artifacts that can be accessed do not contain the required class, then a ClassNotFoundException, or similar, will be thrown. Previously, a SecurityException would have been thrown when access to a URL was denied. If required to revert to the old behavior, this change can be disabled by setting the `jdk.net.URLClassPath.disableRestrictedPermissions` system property.

2.9 Changes in R28.3.12

This section describes the changes in Oracle JRockit JDK R28.3.12:

- Upgraded to JDK 6u131

2.9.1 Upgraded to JDK 6u131

JRockit R28.3.12 is upgraded to JDK 6u131. For information about bug fixes and other changes in this JDK version, see the JDK 6 Release Notes.

2.10 Changes in R28.3.11

This section describes the changes in Oracle JRockit JDK R28.3.11:

- Upgraded to JDK 6u121
- Support for TLS v1.2

2.10.1 Upgraded to JDK 6u121

JRockit R28.3.11 is upgraded to JDK 6u121. For more information about bug fixes and other changes in this JDK version, see the JDK 6 Release Notes.

2.10.2 Support for TLS v1.2

TLS v1.2 is now a TLS protocol option available with the release of JDK 6u121 and JRockit R28.3.11. By default, TLS v1.0 will remain as the default enabled protocol on both client and server sides. For more information about enabling TLS v1.2, see the release notes for JDK 6u121 at:

`http://www.oracle.com/technetwork/java/javase/overview-156328.html`

2.11 Changes in R28.3.10

This section describes the changes in Oracle JRockit JDK R28.3.10:

- Upgraded to JDK 6u115

2.11.1 Upgraded to JDK 6u115

JRockit R28.3.10 is upgraded to JDK 6u115. For information about bug fixes and other changes in this JDK version, see the JDK 6 Release Notes.

2.12 Changes in R28.3.9

This section describes the changes in Oracle JRockit JDK R28.3.9:

- Upgraded to JDK 6u111
- Support for TLS v1.1
- New Diagnostic Command to Generate Core File

2.12.1 Upgraded to JDK 6u111

JRockit R28.3.9 is upgraded to JDK 6u111. For information about bug fixes and other changes in this JDK version, see the JDK 6 Release Notes.

2.12.2 Support for TLS v1.1

TLS v1.1 is now a TLS protocol option available with the release of JDK 6u111 and JRockit R28.3.9. By default, TLS v1.0 will remain as the default enabled protocol on both client and server sides. For more information about enabling TLS v1.1, see the release notes for JDK 6u111 at:

`http://www.oracle.com/technetwork/java/javase/overview-156328.html`.

2.12.3 New Diagnostic Command to Generate Core File

A new diagnostic command, `fork_and_abort`, has been added to generate core files from a JRockit process in environments where external tools such as `gdb` and `gcore` cannot be used.

For more information about the `fork_and_abort` command, see "Diagnostics Commands" in *JRockit Command-Line Reference*.

2.13 Changes in R28.3.8

This section describes the changes in Oracle JRockit JDK R28.3.8:

- Upgraded to JDK 6u105
- New Command-Line Options for Generating Core Dump Files on Exception

2.13.1 Upgraded to JDK 6u105

JRockit R28.3.8 is upgraded to JDK 6u105. For information about bug fixes and other changes in this JDK version, see the JDK 6 Release Notes.

2.13.2 New Command-Line Options for Generating Core Dump Files on Exception

In this release, two new command-line options have been added for JVM diagnostics, `-XX:AbortVMOnException` and `-XX:AbortVMOnExceptionMessage`. These options are used to dump the JRockit text dump and a core file when an exception specified by `-XX:AbortVMOnException` and having an exception message specified by `-XX:AbortVMOnExceptionMessage` occurs.

Example:

```
java -XX:+UnlockDiagnosticVMOptions -XX:AbortVMOnException=InvalidClassException
-XX:AbortVMOnExceptionMessage="class invalid for deserialization" -
Xverbose:exceptions DeserializeTest
```

The above command dumps JRockit text dump and a core file when the specified `InvalidClassException` exception having the message "class invalid for deserialization" occurs. The `-Xverbose` output will be as follows:

```
[INFO ][excepti][00004] java/io/InvalidClassException: A; class invalid for
deserialization
[ERROR] JRockit Fatal Error: Non-continuable exception (60)
[ERROR] Saw java/io/InvalidClassException, aborting
[JRockit] JVM State dumped to /export/test/jrockit.2091.dump.
Aborted (core dumped)
```

2.14 Changes in R28.3.2

This section describes the changes in Oracle JRockit JDK R28.3.2:

- New Default Value for the -XX:+CheckStacks Command-Line Option

- New Command-Line Option to Disable Garbage Collection of Constant Pool
- New Verbose Option for Shutdown Report

2.14.1 New Default Value for the -XX:+CheckStacks Command-Line Option

In JRockit versions prior to R28.3.2, the `-XX:+|-CheckStacks` option was disabled by default, meaning that JRockit did not explicitly check for stack overflows on a JNI method entry.

The -XX:+|-CheckStacks option in JRockit R28.3.2 and later versions will be enabled by default. In very rare cases, the additional overhead of stack overflow detection may result in low performance. This overhead can be avoided by explicitly disabling the stack checking by using the `-XX:-CheckStacks` option.

It is also possible, in very rare cases, that the system starts throwing `StackOverflowErrors` after enabling -XX:+|-CheckStacks. This happens only if the thread was within one page of memory from overflowing the stack. In this case, the recommended resolution is to increase the stack size by a small amount using the `-Xss` option, not by disabling -XX:+|-CheckStacks. For more information about the `-Xss` option, see *JRockit Command-Line Reference*.

2.14.2 New Command-Line Option to Disable Garbage Collection of Constant Pool

Applications that use a lot of reflection or serialization would suffer from the performance overhead of garbage collection activity that is required to help prune the runtime shared constant pool. This overhead can be eliminated with the new command line option `-XX:-UseCPoolGC`. Use of this option may result in native memory leaks.

For more information about this option, see `-XX:-UseCPoolGC` in *JRockit Command-Line Reference*.

2.14.3 New Verbose Option for Shutdown Report

A new parameter `shutdown` has been added to the `-Xverbose` option. When you set this parameter, JRockit provides information about any event that has triggered a normal shutdown of the JVM.

For more information, see the description of `-Xverbose` in *JRockit Command-Line Reference*.

2.15 Changes in R28.2.3

This section describes the changes in Oracle JRockit JDK R28.2.3.

2.15.1 New Default Value for the -XX:MaxLargePageSize Command-Line Option

In earlier versions of JRockit R28, the default value of the `-XX:MaxLargePageSize` option was zero, which means there was no limit for the maximum size for the large pages and the value was specified by the operating system.

The default value for this option in JRockit R28.2.3 and later versions will be 256 MB.

2.16 Changes in R28.2.2

This section describes the changes in Oracle JRockit JDK R28.2.2.

2.16.1 Fixed Issues in Finalization

Earlier versions of JRockit R28 suffered from an issue where some finalizers may never be executed. This issue was resolved in JRockit R28.2.2. Because JRockit uses finalizers internally to manage class constant pool data, a side effect of fixing this issue is that applications that continuously load and access the constant pool data of classes using sun.reflect.ConstantPool, may experience an increase in the number of finalizable objects stored on the Java heap after upgrading to R28.2.2 or later. In very rare cases, this additional pressure on the memory subsystem may result in performance issues (such as higher heap consumption or more GC activity) or even OutOfMemoryErrors. It is almost always possible to resolve such performance issues by modifying the application to eliminate redundant class loading.

2.17 Changes in R28.2.0

This section describes the changes in Oracle JRockit JDK R28.2. These changes are:

- Improved JRockit Flight Recorder Heap Statistics Events
- Command-Line Options to Filter Exception Logging and Events

2.17.1 Improved JRockit Flight Recorder Heap Statistics Events

In earlier versions of JRockit R28, long garbage collections resulted in application pauses during profiling recording.

To avoid the long application pauses, the JRockit Flight Recorder Heap Statistics events have been improved.

2.17.2 Command-Line Options to Filter Exception Logging and Events

The new command-line option `-XX:ExceptionTraceFilter` and the new diagnostic command `exception_trace_filter` filter JVM exception logging and JRockit Flight Recorder exception events based on the exception type specified.

For more information about these commands, see *Oracle JRockit Command-Line Reference*.

2.18 Changes in R28.1.5

This section lists the changes in Oracle JRockit JDK R28.1.5.

2.18.1 JRockit Mission Control Samples are No Longer Installed by Default

The installer for Oracle JRockit JDK R28.1.5 with Oracle JRockit Mission Control 4.0.1 will not install JRockit Mission Control samples by default. You must select the optional component **Demos and Samples** to install JRockit Mission Control samples.

For more information about installing the product, see *JRockit Installation and Upgrade Guide*.

2.19 Changes in R28.1.0

This section lists the changes in JRockit JDK R28.1.0. These changes are:

- Improved Garbage Collection
- Command-Line Option to Specify the Receive Buffer Size
- Enabling JVM Crash When an Out-of-Memory Error Occurs
- Collecting and Packaging Flight Recording Data from Disk Buffers

2.19.1 Improved Garbage Collection

In the `genpar` garbage collection mode, when the nursery runs out of memory in the old generation, objects that are identified for promotion to the old space are promoted within the nursery and this resulted in fragmentation of the nursery. This situation is known as promotion failure.

In R28.1, the JRockit JVM prevents promotion failure by triggering an early old collection for those young collections that are running out of memory.

2.19.2 Command-Line Option to Specify the Receive Buffer Size

When reading from network sockets, the size of the receive buffer can be limited by using the new command-line option, `-XX:MaxRecvBufferSize`.

For more information about this option, see `-XX:MaxRecvBufferSize` in the *JRockit Command-Line Reference*.

2.19.3 Enabling JVM Crash When an Out-of-Memory Error Occurs

Oracle JRockit R28.1 introduces the command-line option `-XX:[+|-]CrashOnOutOfMemoryError`. If this option is enabled, when an out-of-memory error occurs, the JRockit JVM crashes and produces crash files. The state of the JVM before a crash is saved to a core dump file for off-line analysis.

For more information about this option, see `-XX:+|-CrashOnOutOfMemoryError` in the *JRockit Command-Line Reference*.

2.19.4 Collecting and Packaging Flight Recording Data from Disk Buffers

This release of Oracle JRockit introduces the command-line tool `oracle.jrockit.jfr.tools.ConCatRepository`, that allows you to extract JRockit Flight Recorder data that has been written to disk, but not handled and packaged as a flight recording, and then create a flight recording from it. This feature is useful when you have flight recording buffers on disk and the JVM terminates in such a way that `.jfr` files are not assembled to a complete flight recording file.

For more information, see the *JRockit Flight Recorder Run Time Guide*.

2.20 Changes in R28.0.1

This section lists the changes in JRockit JDK R28.0.1.

2.20.1 Default MaxCodeMemory on Linux IA32 with Large Pages Increased to 64 MB

The default maximum code memory on Linux IA32 with large pages was 32 MB in R28.0.0.

In R28.0.1, the default value has been changed to 64 MB.

For more information, see `-XX:MaxCodeMemory` in the *JRockit Command-Line Reference*.

2.21 New Features and Changes in R28.0.0

The following are the new features and changes in JRockit JDK R28.0.0. These changes are:

- Change in Thread Suspension Mechanism
- Ability to Generate HPROF-Formatted Heap Dumps
- Improved Logging for Code Generation and Optimization
- Better Control Over Code Optimization Through Directives
- Garbage Collection Strategy Does Not Change at Run Time
- Large Objects Are Allocated in the Nursery
- Single Command-Line Option to Specify Compaction Behavior
- Changes in the JMX Agent
- Compressed References for Larger Heaps
- Changes in Heap Sizing
- Change in Class and Code Garbage Collection
- New Command-Line Options in R28.0
- Command-Line Options Deprecated in R28.0
- Command-Line Options Changed to the HotSpot Format in R28.0

2.21.1 Change in Thread Suspension Mechanism

The mechanism to stop threads in the JVM has been changed. In previous releases of the JRockit JVM, threads were suspended (for performing garbage collection, for example) by sending signals. In JRockit JVM R28.0, the threads check periodically whether they should self-suspend.

This change does not result in visible behavioral changes, but it makes the JRockit JVM easier to maintain and less error-prone.

2.21.2 Ability to Generate HPROF-Formatted Heap Dumps

Heap dumps can now be generated in the HPROF binary format, which can be parsed using heap analysis tools. You can use the `-XX:+HeapDumpOnOutOfMemoryError` or `-XX:+HeapDumpOnCtrlBreak` command-line options to generate Java heap dumps in HPROF binary format on `OutOfMemory` errors. You can also generate heap dumps in HPROF format by using the `hprof` diagnostic command and through JRockit Mission Control.

For more information, see "Generating Java Heap Dumps in the HPROF Binary Format" in the *JRockit Diagnostics and Troubleshooting Guide*.

2.21.3 Improved Logging for Code Generation and Optimization

The granularity of logging for code generation and optimization has been increased, to enable faster diagnostics and troubleshooting. The information in the outputs of the `-Xverbose:opt` and `-Xverbose:codegen` options is now more detailed. For more information, see the *JRockit Command-Line Reference*.

2.21.4 Better Control Over Code Optimization Through Directives

In previous releases of the JRockit JVM, you could control code optimization by specifying directives in an optfile and then using the `-Djrockit.optfile` property to indicate the name and location of the optfile.

In JRockit JVM R28.0, the format for specifying compiler-control directives in the optfile has been extended and improved to enable control over code optimization at a more detailed level. A new **diagnostic** command-line option, `-XX:OptFile`, is available for specifying the name and path of the optfile.

For more information, see "Specifying Optimization Directives" in the *JRockit Diagnostics and Troubleshooting Guide*.

2.21.5 Garbage Collection Strategy Does Not Change at Run Time

In JRockit JVM R28.0, when you specify the `throughput` or `pausetime` garbage collection mode by using the `-Xgc` command-line option, the strategy associated with the specified mode— by default, `genpar` and `gencon` respectively—is used **throughout** the run time. The garbage collector does not change between generational and nongenerational garbage collectors during the run time.

This change has been made to reduce the extent of underterministic garbage collection behavior due to strategy changes during run time.

Note that the `-XgcPrio` option continues to work in R28.0. Oracle recommends that you use the `-Xgc` option instead of using the `-XgcPrio` option.

For more information about `-Xgc`, see the *JRockit Command-Line Reference*.

2.21.6 Large Objects Are Allocated in the Nursery

Oracle JRockit JVM R28.0 allocates large objects in the nursery if the size of the object is within the limit specified by the `-XXtlaSize:wasteLimit` command-line option.

This change improves the utilization of the nursery and reduces the frequency of old-space garbage collections.

For more information about `-XXtlaSize:wasteLimit`, see the *JRockit Command-Line Reference*.

2.21.7 Single Command-Line Option to Specify Compaction Behavior

Oracle JRockit R28.0 supports a new command-line option that enables you to specify compaction-related behavior: `-XXcompaction`. This option accepts all the parameters for compaction: compaction percentage, maximum number of references, and so on.

All the other compaction-related options are deprecated in R28.0.

For more information about `-XXcompaction`, see the *JRockit Command-Line Reference*.

2.21.8 Changes in the JMX Agent

In JRockit JVM R28.0, after the local management service starts, it remains active until the JVM is terminated.

In previous releases, the performance counter memory used to leak, and new addresses of the JMX connector were written during a memory leakage. Therefore, the JMX client was reading the wrong address of the JMX connector. This issue has been fixed in R28.0.

To allow RMI communication between the JRockit JVM server and a client through a firewall, two ports (RMI Registry and RMI Server) are required to configure the firewall. In previous releases, the RMI Server port number was generated randomly on the JRockit JVM server; so it was not possible to configure the firewall in advance. In JRockit JVM R28.0, the JMX agent enables you to select the same port number for the RMI Registry and the RMI Server. Therefore, you can use the default JMX agent for RMI communication through a firewall.

You can set the JMX agent properties by using the system properties or by using the `-Xmanagement` command-line option. For more information about the JMX agent system properties, see Appendix B "JMX Agent-Related –D Options" in the *JRockit Command-Line Reference*.

2.21.9 Compressed References for Larger Heaps

Oracle JRockit JVM R28.0 supports up to 64 GB compressed references for various heap sizes. You can define the compressed reference size during the JVM startup by using the `-XXcompressedRefs` command-line option.

For more information about `-XXcompressedRefs`, see the *JRockit Command-Line Reference*.

2.21.10 Changes in Heap Sizing

In JRockit JVM R28.0, the heap grows faster than before. The JVM also ensures that the heap size grows up to the maximum Java heap size (`-Xmx`) before an OutofMemory error is thrown. In addition, the default value of the `-Xmx` option is changed from 1 GB to 3 GB on 64-bit platforms.

The JRockit JVM shrinks the heap if it is unused or if other applications require more physical memory.

In the previous releases, the JVM used to crash when the heap size reduced from a very large size to a small size. This issue has been fixed in R28.0.

2.21.11 Change in Class and Code Garbage Collection

The pause time during the old collection of code and class garbage collections has been removed in Oracle JRockit JVM R28.0. With this change, the code and class garbage collections are mostly concurrent in R28.0.

2.21.12 New Command-Line Options in R28.0

The Oracle JRockit JVM R28.0 supports several new command-line options.

For more information, see Appendix A, "Changes in Command-Line Options" in the *JRockit Command-Line Reference*.

2.21.13 Command-Line Options Deprecated in R28.0

Some command-line options have been deprecated in Oracle JRockit JVM R28.0.

For more information, see Appendix A, "Changes in Command-Line Options" in the *JRockit Command-Line Reference*.

2.21.14 Command-Line Options Changed to the HotSpot Format in R28.0

In Oracle JRockit JVM R28.0, the format of several command-line options has been changed to the HotSpot format: `-XX:+|-`*`option`* (for example, `-XX:+UseClassGC`).

For a list of the command-line options that have been changed to the HotSpot format, see Appendix A, "Changes in Command-Line Options" in the *JRockit Command-Line Reference*.

3

Issues Resolved in Oracle JRockit JDK R28

This chapter lists the issues resolved in Oracle JRockit JDK R28.

Note:

For information about bug fixes in the JDK, see the JDK 6 Release Notes.

It contains the following sections:

3.1 Issues Resolved in R28.3.20

For information about bug fixes in the JDK, see the JDK 6 Release Notes.

3.2 Issues Resolved in R28.3.19

The following issues have been fixed in Oracle JRockit JDK R28.3.19. For more information about bug fixes and other changes in the JDK, see the JDK 6 Release Notes.

3.2.1 JVM Hang During Startup on Processors with Large Number of Logical Processors

If an x86 or x86_64 Processor reports (via CPUID) more than 128 logical processors `per package`, earlier versions of JRockit would hang during startup.

3.2.2 Immediate Crash During Startup with Linux on Recent x86 / x86_64 Processors

When using Linux on some newer x86 or x86_64 processors, the additional stack space required to save the state of newly added registers may cause earlier versions of JRockit to run out of stack space and crash during signal handling. In environments where this issue happens, JRockit will crash immediately during startup and will not output a text crash file (jrockit.<pid>.dump). Currently the only processors known to trigger this issue all support various versions of the AVX-512 instruction set extension.

3.3 Issues Resolved in R28.3.18

The following issues have been fixed in Oracle JRockit JDK R28.3.18. For more information about bug fixes and other changes in the JDK, see the JDK 6 Release Notes.

- MissingResourceException Thrown While Trying to Load ResourceBundle via Reflection

3.3.1 MissingResourceException Thrown While Trying to Load ResourceBundle via Reflection

A regression was introduced in JRockit R28.3.17 that may prevent JRockit from correctly loading a `java.util.ResourceBundle` via the reflection API. One common symptom of this issue is a `java.util.MissingResourceException` being thrown while running the WebLogic Scripting Tool (WLST).

3.4 Issues Resolved in R28.3.17

For information about bug fixes in the JDK, see the JDK 6 Release Notes.

3.5 Issues Resolved in R28.3.16

For information about bug fixes in the JDK, see the JDK 6 Release Notes.

3.6 Issues Resolved in R28.3.15

For information about bug fixes in the JDK, see the JDK 6 Release Notes.

3.7 Issues Resolved in R28.3.14

The following issues have been fixed in Oracle JRockit JDK R28.3.14. For more information about bug fixes and other changes in the JDK, see the JDK 6 Release Notes.

- Correction of IllegalArgumentException from TLS handshake

3.7.1 Correction of IllegalArgumentException from TLS handshake

security-libs/javax.net.ssl

A recent issue from the JDK-8148516 fix can cause issue for some TLS servers. The problem originates from an IllegalArgumentException thrown by the TLS handshaker code.

```
java.lang.IllegalArgumentException: System property
jdk.tls.namedGroups(null) contains no supported elliptic curves
```

The issue can arise when the server doesn't have elliptic curve cryptography support to handle an elliptic curve name extension field (if present). Users are advised to upgrade to this release. By default, JDK 7 Updates and later JDK families ship with the SunEC security provider which provides elliptic curve cryptography support. Those releases should not be impacted unless security providers are modified.

See JDK-8173783

3.8 Issues Resolved in R28.3.13

For information about bug fixes in the JDK, see the JDK 6 Release Notes.

3.9 Issues Resolved in R28.3.12

The following issues have been fixed in Oracle JRockit JDK R28.3.12. For more information about bug fixes and other changes in the JDK, see the JDK 6 Release Notes.

- Hashtable Deserialization Reconstitutes Table with Wrong Capacity

3.9.1 Hashtable Deserialization Reconstitutes Table with Wrong Capacity

The class library is normally kept in sync between HotSpot-based and JRockit-based JDKs, but the fix for JDK bug 8068427 was not included in the previous JRockit release, R28.3.11 (JDK 6u121). This fix is included in this release.

3.10 Issues Resolved in R28.3.11

The following issues have been fixed in Oracle JRockit JDK R28.3.11:

- Corrupted Heap Data Resulting in Stability Issues
- Incorrect Value for Dark Matter Reported by Heap Diagnostics
- Incorrect Heap Statistics When Instances of a Class Consume Over 2 GB of Heap Space
- Hardware Support for Square Root on SPARC T2
- Default Number of Garbage Collection Worker Threads on Certain Solaris Systems
- JVM Crashes when Using Application Data Integrity Features
- Unexpected Behavior when Copying Arrays

3.10.1 Corrupted Heap Data Resulting in Stability Issues

A regression was introduced in JRockit R28.3.10 that may cause corruption of data stored on the Java heap resulting in crashes and other unexpected behavior.

This issue has been fixed.

3.10.2 Incorrect Value for Dark Matter Reported by Heap Diagnostics

Under certain circumstances, the amount of dark matter (fragmentation) reported by heap diagnostics (`heap_diagnostics` and `HeapDiagnosticsOnOutOfMemoryError`) would be incorrect. Often this would manifest as an impossibly large value (larger than the total heap size) being reported.

This issue has been resolved.

3.10.3 Incorrect Heap Statistics When Instances of a Class Consume Over 2 GB of Heap Space

Heap diagnostics (as output by the `HeapDiagnosticsOnOutOfMemoryError` command line flag or the `heap_diagnostics` diagnostic command) could contain errors when instances of a single class collectively consume 2 GB or more of heap space. Specifically, such classes may be displayed with an incorrect delta value (the calculated difference with the previous heap diagnostic output) and may appear in an incorrect position (out of sort order) within the list of classes. Trend analysis sort order in the Memory Leak Detector may also be impacted.

This issue has been resolved.

3.10.4 Hardware Support for Square Root on SPARC T2

On SPARC T2-based systems, JRockit did not utilise direct floating point hardware support for square root calculations (that is, `Math.sqrt()`). Instead, such calculations were done in software using simpler instructions.

This issue has been resolved, and now JRockit will utilise the `fsqrtd` instruction for such calculations. As a result, applications which call `Math.sqrt()` frequently may experience a performance improvement.

3.10.5 Default Number of Garbage Collection Worker Threads on Certain Solaris Systems

On SPARC-based systems, sometimes the number of CPU cores could not be correctly identified.

This issue has now been resolved. On SPARC T1, T2, and T3 systems only, the number of CPU cores detected is used to calculate the default number of GC worker threads. (There is no other impact from this change, regardless of CPU).

In the rare event that this change negatively impacts the performance of a given application, the number GC worker threads can be specified with the `-XXgcThreads` command-line option. The number of each type of GC worker thread used by previous versions of JRockit in a given environment can be confirmed by examining `-Xverbose` GC debug output. For example:

```
$ <OLD_JROCKIT_HOME>/bin/java -Xverbose:gc=debug -version
.
< . . . >
[DEBUG][memory ] Initial and maximum number of gc threads: 8, of which 8
parallel threads, 4 concurrent threads, and 8 yc threads.
< . . . >
.
$ <NEW_JROCKIT_HOME>/bin/java -XXgcThreads:yc=8,con=4,par=8 MyJavaApplication
```

3.10.6 JVM Crashes when Using Application Data Integrity Features

On SPARC systems equipped with a CPU that supports Realtime Silicon Secured Memory, a stale memory reference would be detected during JVM startup. This would result in a crash when using `libadimalloc`, the Application Data Integrity aware memory allocation library, preventing its use with JRockit.

This issue has been resolved.

3.10.7 Unexpected Behavior when Copying Arrays

The JRockit compiler, when optimizing code that calls the `java.lang.System.arraycopy()` method, may generate incorrect code resulting in erroneous computation results. This issue often manifests as an unexpected `ArrayIndexOutOfBoundsException`.

This issue has been resolved.

3.11 Issues Resolved in R28.3.10

The following issues have been fixed in Oracle JRockit JDK R28.3.10:

- Crash while Running Finalizer for the ConstPoolWrapper Object
- Crash due to Incorrectly Compiled (JIT) checkcast Operation
- Unexpected Behavior when Inlining a Method

3.11.1 Crash while Running Finalizer for the ConstPoolWrapper Object

When a Java heap exhaustion (a type of OutOfMemoryError condition) happens, JRockit would sometimes crash while executing the `constpoolwrapper_finalize` function. This issue has been resolved.

However, for any type of OutOfMemoryError condition, Oracle recommends that you identify and resolve the root cause of the memory exhaustion regardless of this change.

3.11.2 Crash due to Incorrectly Compiled (JIT) checkcast Operation

A race on VM startup between class loading and JIT compilation may result in incorrectly generated code. The behavior of such code is undefined, but has been known to manifest as a JVM crash, often accompanied by a `SIGTRAP` signal or a 'No exception handler found [56]' error message.

This issue has been resolved.

3.11.3 Unexpected Behavior when Inlining a Method

The JRockit compiler, when inlining a method which has no normal return paths (that is, it only throws exceptions), may generate incorrect code resulting in erroneous computation results. The JVM may hang, crash, or exhibit other undefined behavior.

This issue has been resolved.

3.12 Issues Resolved in R28.3.9

The following issues have been fixed in Oracle JRockit JDK R28.3.9:

- Issues with the ObjectStreamClass.lookup Method
- Process Hangs after NewStringUTF Invocation

3.12.1 Issues with the ObjectStreamClass.lookup Method

In certain circumstances, `ObjectStreamClass.lookup()` could return a non-null result even when a non-serializable class is passed. This could result in various serialization issues and other incorrect behavior.

This issue has been resolved.

3.12.2 Process Hangs after NewStringUTF Invocation

Under certain circumstances, the JRockit process may hang soon after a NULL pointer is passed (in place of a char array) as the second argument to the JNI `NewStringUTF` function. As native code within the Java runtime itself may trigger this issue, even systems that do not use third party JNI libraries may be susceptible.

This issue has been resolved.

3.13 Issues Resolved in R28.3.8

The following issues have been fixed in Oracle JRockit JDK R28.3.8:

- Incorrect CPU Consumption Values on Linux
- Unexpected NoSuchMethodError in JRockit JVM

3.13.1 Incorrect CPU Consumption Values on Linux

On Linux kernels 2.6 and later, JRockit would include time spent waiting for IO completion as "CPU usage". During periods of heavy IO activity, this could result in misleadingly high values reported as CPU consumption in various tools like Flight Recorder, performance counters, and the `cpuload` diagnostic command.

This issue has been resolved.

3.13.2 Unexpected NoSuchMethodError in JRockit JVM

In extreme cases, the reference count on a JVM internal string could overflow, which could result in internal string comparisons failing. This internal failure could result in a `NoSuchMethodError` and possibly other errors. This issue has been resolved.

3.14 Issues Resolved in R28.3.6

The following issues have been fixed in Oracle JRockit JDK R28.3.6:

- JRockit JVM Crashes while Debugging a Java Program Compiled with javac

- JVM Crashes while Using an Agent

3.14.1 JRockit JVM Crashes while Debugging a Java Program Compiled with javac

Bug 12943958

Running Java applications compiled with `javac` in debug mode (using the "`-g`" option) might result in a JVM crash. This happens when you have specified `-Xdebug` or `-XX:+JavaDebug` option at JVM startup. The incorrect debugging information in some class files generated by Oracle `javac` caused this issue.

This issue has been fixed in Oracle JRockit JDK R28.3.6. After upgrading, recompile the classes using the fixed version of `javac`.

3.14.2 JVM Crashes while Using an Agent

Previous versions of JRockit may crash or experience other stability issues after redefining a class at runtime by using the JVM Tool Interface (JVM TI) or by using the `java.lang.instrument` interface (`-javaagent` startup parameter).

This issue has been resolved.

3.15 Issues Resolved in R28.3.5

The following issues have been fixed in Oracle JRockit JDK R28.3.5:

- Issue with Profiling Methods
- JVM Crashes with Illegal Memory Access Error Due to an Optimization Issue

3.15.1 Issue with Profiling Methods

Previous versions of JRockit R28 were unable to correctly instrument Java methods that used generics. This resulted in the inability to profile such methods with the JMXMAPI profiling API or JRockit Mission Control Console's profiler.

This issue has been resolved.

3.15.2 JVM Crashes with Illegal Memory Access Error Due to an Optimization Issue

Fixed an issue where JRockit would crash due to bad code optimization when using StringBuffer/StringBuilder in a loop. This issue only happened when running with compressed references.

3.16 Issues Resolved in R28.3.4

The following issues have been fixed in Oracle JRockit JDK R28.3.4:

- Reduced Memory Footprint of Command-line Tools
- JRockit Crashes while Calling jrockit.vm.ArrayCopy.copy Methods

3.16.1 Reduced Memory Footprint of Command-line Tools

The default options that are passed to the JRockit JVM during invocation of various command-line tools such as `jps`, `jrcmd`, and `jstat` have been changed to reduce the amount of native memory required.

3.16.2 JRockit Crashes while Calling jrockit.vm.ArrayCopy.copy Methods

When copying huge arrays, the array size (or offsets) measured in elements could be converted to an array size in bytes for some copies and this could overflow the size of a 32-bit integer, leading to a copy that does not terminate correctly and rarely causing a JVM crash.

This issue has been resolved.

3.17 Issues Resolved in R28.3.2

The following issues have been fixed in Oracle JRockit JDK R28.3.2:

- Issue with Flight Recording During Startup
- Check Stacks Option on SPARC Platform
- Unexpected NullPointerException Thrown from Methods After Code Optimization

3.17.1 Issue with Flight Recording During Startup

In previous versions of JRockit R28, if the default recording and/or buffering to disk is enabled by using the `-XX:FlightRecorderOptions` command-line option and when you start a new recording using the `-XX:StartFlightRecording` option, JRockit would sometimes hang indefinitely during startup. For more information about the command-line options and parameters, see *JRockit Command-Line Reference*.

This issue has been resolved.

3.17.2 Check Stacks Option on SPARC Platform

In previous versions of JRockit, the value of -XX:+|-CheckStacks option was ignored on SPARC platform.

This issue has been resolved. In JRockit R28.3.2, the -XX:+|-CheckStacks option is implemented correctly.

3.17.3 Unexpected NullPointerException Thrown from Methods After Code Optimization

In certain cases, when a method invoked implicit or explicit boxing operations, a NullPointerException was thrown from the method. An issue existed in the code optimization that caused this exception.

This issue has been resolved.

3.18 Issues Resolved in R28.3.1

The following issue has been fixed in Oracle JRockit JDK R28.3.1.

3.18.1 FileNotFoundException Thrown while Opening Zip Archives

JRockit did not recognize multibyte characters and hence a FileNotFoundException was thrown while opening a zip archive that has a UTF8 encoded filename.

This issue has been resolved.

3.19 Issues Resolved in R28.2.9

The following issues have been fixed in Oracle JRockit JDK R28.2.9:

- Heap Dumps Not Generated on Out Of Memory Error
- Issue with the Out of Memory Error Message

3.19.1 Heap Dumps Not Generated on Out Of Memory Error

The JVM could deadlock when creating an hprof dump on OutOfMemoryError if you are using the JVM flag `-XX:+HeapDumpOnOutOfMemoryError`.

This issue has been resolved.

3.19.2 Issue with the Out of Memory Error Message

In previous versions of JRockit R28, a native OutOfMemoryError message may indicate an incorrect value for the failed allocation size.

This issue has been resolved.

3.20 Issues Resolved in R28.2.8

The following issues have been fixed in Oracle JRockit JDK R28.2.8:

- NullPointerExceptions from Package.getPackages Calls
- NullPointerExceptions from Class.isAssignable
- JRockit Crashes while Code Optimization in cgGetColorForVarInBlock

3.20.1 NullPointerExceptions from Package.getPackages Calls

In JRockit releases R28.2.6 and R28.2.7, a known issue exists wherein, under rare circumstances, a call to `Package.getPackages` may result in a NullPointerException in `Package.defineSystemPackage`. This issue has been resolved.

3.20.2 NullPointerExceptions from Class.isAssignable

A race on VM startup between class loading and serialization of classes could result in sporadic NullPointerExceptions from `Class.isAssignable` in `java.lang` package. This issue has been resolved.

3.20.3 JRockit Crashes while Code Optimization in cgGetColorForVarInBlock

In certain cases, JRockit could crash during code optimization in the `cgGetColorForVarInBlock` method. This issue has been resolved.

3.21 Issues Resolved in R28.2.6

The following issues have been fixed in Oracle JRockit JDK R28.2.6:

- JRockit Fight Recorder Repository Growing Indefinitely
- Unexpected Errors from Applications with Dynamically-created Classes
- JMXMAPI Profiling API Can Now Profile All Versions of a Class

3.21.1 JRockit Fight Recorder Repository Growing Indefinitely

When using certain settings in a custom .jfs file, the Oracle JRockit Flight Recorder repository could, under some circumstances, continue growing indefinitely, even when limited by `maxsize` settings. This has been fixed.

3.21.2 Unexpected Errors from Applications with Dynamically-created Classes

Applications that dynamically created classes (for example, by using JAXB) could run out of native memory, crash, or exhibit other unexpected behaviors due to an internal reference counting issue that could eventually corrupt memory or allow unused classname strings to accumulate after their referring classes were unloaded. This has been fixed.

3.21.3 JMXMAPI Profiling API Can Now Profile All Versions of a Class

Previous versions of the JMXMAPI profiling API could only profile a single instance of a class with the same name. If multiple class loaders loaded the same class, only one version of the class would be instrumented. Now all versions of a class (with the same fully qualified class name) will be instrumented. This change also impacts JRockit Mission Control Console's profiling functionality (there is no impact to JFR profiling).

3.22 Issues Resolved in R28.2.5

The following issues have been fixed in Oracle JRockit JDK R28.2.5:

- JRockit Crashes when Interned Strings are Allocated

- JRockit Crashes while Running with an optfile
- FileNotFoundException Thrown while Reading Files from FileInputStream
- Issue while Closing a NIO Socket
- NIO Operations Fail on Windows with a Security Exception
- Wrong Exception Thrown when Flight Recorder is Disabled
- JRockit Crashes while Invoking a com.sun.management Method

3.22.1 JRockit Crashes when Interned Strings are Allocated

Bug 14271750

Under rare circumstances, JRockit would crash manipulating shared resources from multiple threads, mostly involving interning Java Strings concurrently from multiple threads. This issue has been fixed.

3.22.2 JRockit Crashes while Running with an optfile

Bug 14268514

In rare conditions, the JRockit JVM would crash if compiler directives are used in an optfile. This issue has been fixed.

3.22.3 FileNotFoundException Thrown while Reading Files from FileInputStream

Bug 14093205

In previous JRockit versions, from R28.0.0 to R28.2.4, if the standard input stream was a pipe and if it was closed by the application, subsequent attempts to read other FileInputStreams could incorrectly result in a `FileNotFoundException` error. This issue has been fixed.

3.22.4 Issue while Closing a NIO Socket

Bug 14047648

Under certain circumstances, JRockit would hang while closing a NIO Socket. This issue has been fixed.

3.22.5 NIO Operations Fail on Windows with a Security Exception

Bug 13925641

When a SecurityManager is used on Windows, certain NIO operations used to fail with an `AccessControlException`. This issue has been fixed.

3.22.6 Wrong Exception Thrown when Flight Recorder is Disabled

Bug 13350796

In previous versions of JRockit, when an application tried to use the Flight Recorder API while running on a JRockit instance where Flight Recorder is disabled, a `java.lang.Error` would be thrown. This behavior has been changed so that `java.lang.IllegalStateException` is thrown instead. New explanatory text has also been added to the exception.

3.22.7 JRockit Crashes while Invoking a com.sun.management Method

Bug 14266485

JRockit crashes with illegal memory access in `long [] com.sun.management.getThreadCpuTime(long[] ids)` method due to incomplete implementation of `com.sun.management` extensions.

The array version of `ThreadMXBean.getThreadCpuTime()` has been implemented in Oracle JRockit R28.2.5. The issue has been resolved.

3.23 Issues Resolved in R28.2.4

The following issues have been fixed in Oracle JRockit JDK R28.2.4:

- Issue with the jrcmd Command File Parsing
- Failure to Start on Solaris While Using a Large Page Size
- Issue with print_memusage Diagnostic Command

3.23.1 Issue with the jrcmd Command File Parsing

A regression was introduced in JRockit R28.2.3 where the "stop" statement was ignored by the jrcmd command. This issue has been fixed.

3.23.2 Failure to Start on Solaris While Using a Large Page Size

In previous versions, JRockit could select the wrong compressed reference size and fail to start on Solaris while using a page size over 1 GB. This issue has been fixed.

3.23.3 Issue with print_memusage Diagnostic Command

An invalid value for the `level` argument of the `print_memusage` command could cause the target JVM to exit with an assertion error.

This issue has been fixed in R28.2.4. An error message is displayed and there is no impact to the target JVM.

3.24 Issues Resolved in R28.2.3

The following issues have been fixed in Oracle JRockit JDK R28.2.3:

- Redirecting Ouput of the jrcmd Command to a Specified File
- Issue with the Limited File Size for the jrcmd Script File
- Issue while Reserving VMSpace

- Improved Stack Overflow Handling
- Issue while Optimizing a Method
- Issue with JRockit after Removing JRockit Flight Recorder

3.24.1 Redirecting Ouput of the jrcmd Command to a Specified File

In earlier versions of JRockit, the `jrcmd` command was not redirecting the output to the file specified by the `set_filename` diagnostic command. In JRockit R28.2.3, if an output file is specified by the `set_filename` command, output from diagnostic commands invoked by `jrcmd` will be redirected to the file in the same manner as the output is redirected by the diagnostic commands that you run by the control break handler (`SIGQUIT`).

By default, the output from diagnostic commands invoked by `jrcmd` is sent to the `STDOUT` output stream of the `jrcmd` process. To reset the default behaviour of the `set_filename` command, run the command without specifying a value for the `filename` argument. For more information, see "Diagnostic Commands" in the *JRockit Command-Line Reference*.

3.24.2 Issue with the Limited File Size for the jrcmd Script File

In earlier releases of JRockit, the `jrcmd` command limited the size of a file passed using the `-f` option to a maximum of 256 bytes. From JRockit 28.2.3 onwards, the file size is unlimited, but each line in the file is limited to 256 bytes.

3.24.3 Issue while Reserving VMSpace

There was a regression introduced in JRockit R28.2.0 that could cause JRockit to crash during the `vmsiReserve` method on specific platforms. This issue has been fixed.

3.24.4 Improved Stack Overflow Handling

JRockit used to crash during a stack overflow under some conditions, not having enough space to run its own signal handler, causing the process to abort rather than throwing a StackOverflowError. This issue has been fixed.

3.24.5 Issue while Optimizing a Method

Under rare circumstances, JRockit would crash when optimizing a method, often a method in the `java/util/regex` package. This issue has been fixed.

3.24.6 Issue with JRockit after Removing JRockit Flight Recorder

If the Flight Recorder repository was deleted from the hard drive while JRockit was running, the process would hang indefinitely. This issue has been fixed.

3.25 Issues Resolved in R28.2.2

The following issues have been fixed in Oracle JRockit JDK R28.2.2:

- Exceptions are Thrown while Establishing SSL Connections that use Cipher Suite
- Issue with Code Optimization
- Missing Finalizers

3.25.1 Exceptions are Thrown while Establishing SSL Connections that use Cipher Suite

A regression in Java SE 6 Update 29 caused SSL connection failure while using the `TLS_DH_anon_WITH_AES_128_CBC_SHA` cipher suite. This issue has been fixed.

3.25.2 Issue with Code Optimization

A bug in string append optimization could lead to a crash in JRockit JVM, often during garbage collection. This issue has been fixed.

3.25.3 Missing Finalizers

In JRockit versions R28.0.0 through R28.2.1, some finalizers may never be called. Because JRockit uses finalizers internally to manage class constant pool data, this could cause data corruption resulting in JVM crashes and other unspecified behavior. This issue has been fixed.

3.26 Issues Resolved in R28.1.5

The following issues have been fixed in Oracle JRockit JDK R28.1.5:

- Unable to Reserve Memory in the Low Address Space of the Java Heap
- Thread Starvation while Using the Default Number of Garbage Collection Threads in Multi-Core Machines
- Error while Setting SUID or SGID on JRockit JVM

3.26.1 Unable to Reserve Memory in the Low Address Space of the Java Heap

Bug 12599685

On a 64-bit platform, JRockit could run out of space in the low 4-GB address space of the Java heap and cause `OutofMemory` error during class allocation.

To avoid this error, you can reserve memory in the low 4-GB heap during the JVM startup by using the `-XX:InitialClassBlockMemory` option as follows:

```
-XX:+UnlockDiagnosticVMOptions -XX:InitialClassBlockMemory=100M
```

3.26.2 Thread Starvation while Using the Default Number of Garbage Collection Threads in Multi-Core Machines

Bug 12620601

The default value of garbage collection threads specified by the `-XXgcThreads` option was based on the number of cores and hardware threads on the machine.

The heuristics for garbage collection has been improved to select dynamic number of garbage collection threads. The number of garbage collection threads dynamically selected is now more conservative on large multi-core machines in order to reduce overhead.

3.26.3 Error while Setting SUID or SGID on JRockit JVM

Bug 12339700

The effective user was not same as the real user if you had set SUID or SGID on the Java library. The JRockit JVM failed to start.

This issue has been fixed.

3.27 Issues Resolved in R28.1.4

The following issues have been fixed in Oracle JRockit JDK R28.1.4:

- Warnings Print When Launching Java Involving Symbolic Links on Windows
- Corrupt HPROF File

3.27.1 Warnings Print When Launching Java Involving Symbolic Links on Windows

Bug 12355103

When launching Java involving symbolic links on Windows, Oracle JRockit would sometimes print a warning, such as:

```
[WARN ][osal   ] Could not add counter \Virtual Bytes for query
[WARN ][osal   ] Failed to init virtual size counter.
```

This has been fixed.

3.27.2 Corrupt HPROF File

Bug 11730737

When Oracle JRockit was configured to dump an HPROF on an OutOfMemoryError and was receiving multiple simultaneous OutOfMemoryErrors in multiple threads, the resulting HPROF file might have been corrupt. This has been fixed.

3.28 Issues Resolved in R28.1.3

The following issues have been fixed in Oracle JRockit JDK R28.1.3:

- Deadlock Occurring in the ClassLoader (Sun Bug 7001933)
- "Peer Not Authenticated" Exception Unexpectedly Thrown (Sun Bug 6924489)
- Problem Setting SO_RCVBUF/SO_SNDBUF (Sun Bug 6984182)

3.28.1 Deadlock Occurring in the ClassLoader (Sun Bug 7001933)

Bug 11769358

Occasionally, if a custom file protocol handler was in place, a deadlock would occur in the ClassLoader. This has been fixed; this fix resolves Sun Bug 7001933.

3.28.2 "Peer Not Authenticated" Exception Unexpectedly Thrown (Sun Bug 6924489)

Bug 11769385

The exception `javax.net.ssl.SSLPeerUnverifiedException: peer not authenticated` was unexpectedly being thrown. This has been fixed; this fix resolves Sun Bug 6924489.

3.28.3 Problem Setting SO_RCVBUF/SO_SNDBUF (Sun Bug 6984182)

Bug 11769415

Setting `SO_RCVBUF/SO_SNDBUF` to a value larger than `tcp_max_buf` failed on Solaris 11 if the kernel parameters changed. This has been fixed; this fix resolves Sun Bug 6984182.

3.28.4 Passing Read-Only Bytebuffer to Channel Write Method Throwing Exception

Bug 11709391

Passing a read-only bytebuffer to a channel write method could throw a `java.nio.ReadOnlyByteBufferException`. This has been fixed.

3.28.5 Specific JNI API Routines Did Not Correctly Set isCopy Parameter

Bug 10415204

JNI API `Get<PrimitiveType>ArrayElements` routines did not correctly set `isCopy` parameter to `JNI_TRUE` when returning copies. This has been fixed.

3.28.6 Incorrectly Optimized Methods Forcing Long Values to Become Very Large

Bug 10360591

In rare circumstances, on instances of 32-bit JRockit, a method could be incorrectly optimized forcing long values that should be 0 to become very large. This has been fixed.

3.29 Issues Resolved in R28.1.1

The following issues have been fixed in Oracle JRockit JDK R28.1.1:

- Crashes During Concurrent Sweep JNI Object Allocation
- Silent Exit When Command-Line Options are Misspelled
- Erroneous Optimization of an arraycopy
- JDK Read Fixed Number of Bytes When Calling SecureRandom.generateSeed
- instanceof Check Failing

3.29.1 Crashes During Concurrent Sweep JNI Object Allocation

Bug 10164002

Previously, while it was conducting a concurrent sweep without a nursery, Oracle JRockit might crash if it tried to allocate an object from JNI that was the exact size of the minimum thread local area size. This has been fixed.

3.29.2 Silent Exit When Command-Line Options are Misspelled

Bug 10295969

R28.0.0 and later silently exit if a command-line option was misspelled; for example, "`-X:MaximumNurseryPercentage=80`" (note the single `X` where `XX` is required). This has been fixed in R28.1.1.

3.29.3 Erroneous Optimization of an arraycopy

Bug 10296987

In rare cases, Oracle JRockit could erroneously optimize an `arraycopy` to reuse the source array as the target array. This could lead to the wrong values being read from the source array after the `arraycopy`. This has been fixed.

3.29.4 JDK Read Fixed Number of Bytes When Calling SecureRandom.generateSeed

Bug 10301830

When calling `SecureRandom.generateSeed`, the JDK would always read 8192 bytes from the entropy pool. The JDK has been changed to only read the number of bytes requested. For more information, see:

`http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=6998583`

3.29.5 instanceof Check Failing

Bug 10366647

Occasionally, the optimizer would erroneously change the behavior of an `instanceof opcode`, causing the `instanceof` check to fail. This has been fixed.

3.30 Issues Resolved in R28.1.0

The following issues have been fixed in Oracle JRockit JDK R28.1.0:

- Oracle JRockit Hangs when used with Application Management Solutions
- Memory Leakage in the JMX Implementation
- Oracle JRockit Exits when Aborting an Optimization
- Oracle JRockit Heap Dumps Do Not Open in Eclipse Memory Analyzer
- Exceptions Thrown Without InvocationTargetException Wrapping

3.30.1 Oracle JRockit Hangs when used with Application Management Solutions

Oracle JRockit would hang during startup when Oracle WebLogic Server was started with an application management solution such as CA Wily Introscope.

This issue has been fixed in JRockit R28.1.0.

3.30.2 Memory Leakage in the JMX Implementation

A memory leakage used to occur in the JMX implementation of JDK 6. This issue has been fixed by Sun in JDK 6 Update 22.

The fix is included in JRockit R28.1.0.

3.30.3 Oracle JRockit Exits when Aborting an Optimization

If the `-XX:+|-ExitOnOutOfMemoryError` option was enabled, JRockit would exit when aborting an optimization due to the compiler memory limit.

This issue has been fixed in JRockit R28.1.0.

3.30.4 Oracle JRockit Heap Dumps Do Not Open in Eclipse Memory Analyzer

When the heap size was 2 GB or higher, JRockit would write segmented heap dumps that Eclipse Memory Analyzer (MAT) could not parse.

This issue has been fixed in JRockit R28.1.0.

3.30.5 Exceptions Thrown Without InvocationTargetException Wrapping

When you invoke methods using the class reflection feature, JRockit would sometimes throw an exception type that is different from the signature of the `Method.invoke()` method.

This issue has been fixed in JRockit R28.1.0. Invoking methods using reflection now throws the correct `InvocationTargetException` type.

3.31 Issues Resolved in R28.0.2

The following issues have been fixed in Oracle JRockit JDK R28.0.2:

- Oracle JRockit Starts Slowly on Some Solaris Machines
- IO Exceptions in Epoll Socket Muxer Would Throw NoClassDefFoundErrors
- Oracle JRockit Crashing While Pruning References to Obsoleted Code
- Oracle JRockit Could Not Open JAR or ZIP Files Larger Than 2GB
- Xalan and Xerces Versions Updated

3.31.1 Oracle JRockit Starts Slowly on Some Solaris Machines

Bug 9714564

On some Solaris machines, Oracle JRockit would start slowly, printing warnings; for example:

```
[WARN ][osal   ] Failed to initialize kstat for CPU 0, ignoring
```

This issue has been fixed in release R28.0.2.

3.31.2 IO Exceptions in Epoll Socket Muxer Would Throw NoClassDefFoundErrors

Bug 9728938

IO exceptions originating from the epoll socket muxer would occasionally throw NoClassDefFoundErrors when trying to find the `java/lang/IOException` class. This issue has been fixed in release R28.0.2.

3.31.3 Oracle JRockit Crashing While Pruning References to Obsoleted Code

Bug 9763391

Occasionally, while pruning references to obsoleted code, Oracle JRockit would crash in `Code_and_classgc_background_task`. This issue has been fixed in release R28.0.2.

3.31.4 Oracle JRockit Could Not Open JAR or ZIP Files Larger Than 2GB

Bug 9795028

In previous Oracle JRockit releases, the JVM could not open JAR or ZIP files larger than 2GB. This issue has been fixed in release R28.0.2.

3.31.5 Xalan and Xerces Versions Updated

Bug 9829074

The Xalan and Xerces versions were updated to fix a functional regression found in JDK 1.6.0_18. This fix is included in the Oracle JRockit R28.0.2 JVM.

3.32 Issues Resolved in R28.0.1

The following issues have been fixed in Oracle JRockit JDK R28.0.1.

- JVM Crashes on Encountering Non-UTF8 Characters in Compiler Directives
- Null-Check Incorrectly Optimized or Proved as Always Failing
- Linux Systems Crash at Startup when libjsig.so is Set to be Preloaded
- NIO Selector Functionality Failure
- Deprecated Flag -XXExternalCompactRatio Gives Incorrect Warning
- ZipEntry Initialization Error
- Crash in ZLIB Code While Running Finalizer
- Undeterministic Behavior on x86_64 Machines
- JVM Spins Forever When Compiling JavaFX Classes
- Descriptions Not Intuitive for Compaction JFR Events
- WLS NIOSocketMuxer Occasionally Loses Sockets On Windows

3.32.1 JVM Crashes on Encountering Non-UTF8 Characters in Compiler Directives

Bug 9475801

The JVM crashes when it encounters non-UTF8 characters in a compiler control directives file.

This issue has been fixed in R28.0.1.

3.32.2 Null-Check Incorrectly Optimized or Proved as Always Failing

Bug 9343546

In certain cases involving try-catch clauses, the JVM incorrectly optimizes or proves a null-check as always failing.

This issue has been fixed in R28.0.1.

3.32.3 Linux Systems Crash at Startup when libjsig.so is Set to be Preloaded

Bug 9466275

On Linux systems, the JVM crashes at startup if the user sets `libjsig.so` (the signal chaining library) to be preloaded through the `LD_PRELOAD=libjsig.so` environment variable. This prevents some third-party JNI libraries from being used with Oracle JRockit R28. To download a patch for this issue, see patch ID 9586671 for JDK 6 and patch ID 9672120 for JDK 5.

This issue has been fixed in R28.0.1.

3.32.4 NIO Selector Functionality Failure

Bug 9485661

In certain cases, the NIO selector functionality fails unless `net.dll` is loaded before `nio.dll`. This happens only when using `JAVA_HOME\bin\java` as opposed to `JAVA_HOME\jre\bin\java`. To download a patch for this issue, see patch ID 9586671 for JDK 6 and patch ID 9672120 for JDK 5.

This issue has been fixed in R28.0.1.

3.32.5 Deprecated Flag -XXExternalCompactRatio Gives Incorrect Warning

Bug 9631915

When the deprecated command-line option `-XXexternalCompactRatio` is used, the following incorrect warning is displayed.

```
[WARN ] -XXexternalCompactRatio is a deprecated option. Please use -
XXcompaction:internalPercentage instead.
```

This issue has been fixed in R28.0.1.

3.32.6 ZipEntry Initialization Error

Bug 9671985

When the `java.util.zip.ZipEntry` created for an uncompressed entry (method STORED) is initialized, the uncompressed and compressed fields are not initialized with the same value. This sometimes causes a `java.util.zip.ZipException`.

This issue has been fixed in R28.0.1.

3.32.7 Crash in ZLIB Code While Running Finalizer

Bug 9672130

Sometimes, calling `java.lang.util.zip.Deflater.deflateBytes()` after calling `java.lang.util.zip.Deflater.end()` causes the JVM to crash.

This issue has been fixed in R28.0.1 to throw a `NullPointerException`.

3.32.8 Undeterministic Behavior on x86_64 Machines

Bug 9459003

Sometimes, a method invocation with more than eleven arguments introduces undeterministic behavior in Java applications on x86_64 machines.

This issue has been fixed in R28.0.1.

3.32.9 JVM Spins Forever When Compiling JavaFX Classes

Bug 9651960

The JRockit JVM spins forever when compiling JavaFX classes that contain endless loops.

This issue has been fixed in R28.0.1.

3.32.10 Descriptions Not Intuitive for Compaction JFR Events

Bug 9616739

Some descriptions for the `GcCompaction` JFR event are not intuitive.

This issue has been fixed in R28.0.1.

3.32.11 WLS NIOSocketMuxer Occasionally Loses Sockets On Windows

Bug 9582716

At times, sockets disappear when the `NIOSocketMuxer` is used with WebLogic Server running on Windows.

This issue has been fixed in R28.0.1.

3.33 Issues Resolved in R28.0.0

The following issues have been fixed in Oracle JRockit JDK R28.0.0.

- ACopyRemoval Breaks Explicit Typechecks
- Deadlocks On the Windows Platform When Threads Block on I/O Operations
- Issues with Nondefault Flag with -XXcallProfiling in Oracle JRockit R27.x
- Performance Issues with Windows Computers Running Many Processes
- Optimizing Compiler Producing Erroneous Results
- Broken Java Launcher Removed from Product
- JVMTI_EVENT_COMPILED_METHOD_UNLOAD Event Not Being Posted

3.33.1 ACopyRemoval Breaks Explicit Typechecks

Bug 8816217

Inner type checks on arrays used to show the wrong type in optimized code. This issue has been resolved.

3.33.2 Deadlocks On the Windows Platform When Threads Block on I/O Operations

This release adds a workaround for deadlocks on the Windows platform when thread(s) block on I/O operations on any standard stream (`stdin`, `stdout stderr`/ `System.in`, `System.out`, `System.err`) while a shared library (DLL) is loading. The deadlock occurs if a process is launched such that the stream on which a call is blocked is a redirected pipe, typically the result of either a spawned process through `Process.exec` or similar; or launched through a shell with data piped to or from itself. The bug happens because any I/O operation on such a pipe holds a kernel lock during the whole call, a lock which is also required by the WinAPI function `GetFileType()`. This function in turn is called from the Microsoft CRT startup code whenever either a new CRT DLL, or a DLL with statically linked CRT functions, is loaded.

The bug only occurs on Windows releases prior to Windows Vista.

Workaround

Detect whenever the JVM process is started with its `stdin` redirected to a pipe. Then intercept all `FileInputStream.read()` calls to it and prevent them from blocking, essentially doing polling I/O. The workaround prevents a thread reading from `System.in` from blocking any `System.loadLibrary` calls. This workaround might add some latency to reading from `System.in`, but should be invisible for most applications.

This workaround can be controlled by using these diagnostic options (unlock using `-XX:+UnlockDiagnosticVMOptions`):

- `-XX:+|-UseStdinPipeReadWorkaround`, which enables the workaround (default is on).
- `-XX:StdinPipeReadWorkaroundPollPeriod=<`*`millis`*`>`, which polls the period for reads from `stdin` (default 1)

Notes:

- This fix does not handle NIO reads from `System.in`. Using these might still cause deadlocks of the JVM.
- This fix is only activated if redirection of standard in is detected.

3.33.3 Issues with Nondefault Flag with -XXcallProfiling in Oracle JRockit R27.x

This release resolves the following issues, which arose when using the nondefault flag, `-XXcallProfiling` in Oracle JRockit R27.x.

- Deadlock between compiler and code garbage collection.
- Memory leak of call profiling data from invalidated methods.

3.33.4 Performance Issues with Windows Computers Running Many Processes

This release fixes the issue of slow performance on Windows machines running with many (more than 40) processes with the same name whenever access was needed to the process PDH header or slow startup when running with the JRockit Flight Recorder.

3.33.5 Optimizing Compiler Producing Erroneous Results

The Oracle JRockit optimizing compiler was, in rare cases, producing erroneous results from computations that included a narrowing of primitive types. This issue has been fixed.

3.33.6 Broken Java Launcher Removed from Product

The broken Java launcher java-rmi.exe in JRockit 6 on Windows is no longer shipped with the product. See also Sun Bug 6512052 at:`http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=6512052`

3.33.7 JVMTI_EVENT_COMPILED_METHOD_UNLOAD Event Not Being Posted

Oracle JRockit R27.6.0 was not posting the `JVMTI_EVENT_COMPILED_METHOD_UNLOAD` event when unloading obsolete method code. This issue has been fixed.

4

Known Issues in Oracle JRockit JDK R28

This chapter describes issues known to exist in the Oracle JRockit JDK R28.

4.1 Issue with Object Initialization in JRockit

The JRockit JVM 1.6.0 registers finalizeable objects when an object is allocated.

The Java Language Specification (JLS) necessitates objects to be registered only after the `<init>` method of the `java.lang.Object` class has been run and completed successfully.

4.2 Issues while Using 64-Bit Compressed References on SPARC

The JRockit JVM might crash if you use 64-bit compressed references on SPARC platform.

Workaround

Specify a lower size for the compressed references by using the option `-XXcompressedRefs:size=4GB`. If the problem persists, disable the compressed references by specifying the option `-XXcompressedRefs:enable=false`.

Alternatively, you can change the value of `-Xmx` option to a value less than 3 GB.

For more information about the `-XXcompressedRefs` and `-Xmx` options, see *Oracle JRockit Command-Line Reference*.

4.3 Limited Amount of Active Monitors

If a Java program uses too many (4,194,304) active monitors (that is, by doing wait/notify or contended synchronization on too many objects) the JVM's internal monitor index can overflow. This is more likely to happen when using large heaps and few garbage collections occur. If this happens, the JVM will crash with an error saying "The number of active Object monitors has overflowed."

4.4 Error While Using print_utf8pool Command on Windows

The jrcmd command `print_utf8pool`, that prints all UTF8 strings, fails to handle unicode characters correctly on Windows platform.

4.5 HPROF Heap Dump Might be Corrupt When Multiple OOMs Thrown

If Oracle JRockit encounters several Java Out of Memory exceptions while writing an HPROF heap dump, the contents of the heap dump might be corrupt.

4.6 java.math.BigDecimal Objects Cannot be Serialized Over IIOP Between Releases

Serialization of `java.math.BigDecimal` objects over IIOP between the JRockit JVM and other JVMs throws an IOException. This incompatibility has been fixed; the JRockit JVM R28 is now compatible with other JVMs but not with older R27 releases. As a consequence, `java.math.BigDecimal` objects cannot be serialized over IIOP between the R27 and R28 releases of the JRockit JVM.

4.7 Timing Stability Issue When "Fast Time" Is Enabled on Intel Systems

Timing stability issues might occur on modern x86 systems (for instance Nehalem-EX) with more than two CPU sockets.

Disabling fast time (by using the `-XX:-UseFastTime` command-line option) could solve the issue.

4.8 JMAPI Method Changed to Throw an UnapplicableMethodException

In R28.0 the JMAPI method `com.bea.jvm.ProfilingSystem.newConstructorProfileEntry()` was changed to throw an `UnapplicableMethodException`. This exception is never thrown in practice but the addition causes compilation errors for old code. Removing the exception declaration will also cause problems compilation, thus the exception will remain in future versions.

4.9 Error Message for CPU Load Counters for JRockit JVM Running on Windows

At times, the following message might be displayed when running the JRockit JVM on Windows.

```
[WARN ][osal   ] could not enumerate processors (1) error=-1073738824
[WARN ][osal   ] Failed to init system load counters
```

This issue occurs when the Windows processor performance counter (`PerfOS`) is disabled.

The message also indicates that CPU load events are not recorded in JRockit Flight Recorder and not shown in the JRockit Mission Control console.

Workaround: Enable the `PerfOS` counter in Windows by using the Microsoft Extensible Counter List tool (`exctrlst.exe`). You can download the tool from `http://download.microsoft.com/download/win2000platform/exctrlst/1.00.0.1/nt5/en-us/exctrlst_setup.exe`.

4.10 Oracle JRockit Hangs On OEL/OVM Combination

When Oracle JRockit is running on OEL on OVM, a fix is required in OEL. Listed below are the minimum requirements for running JRockit on OEL on OVM:

- OVM 2.1.2
- OEL 4.7 ia32

  Patch required. The version of the para-virtualized kernel for OEL must be 2.6.9-78.0.13.0.1.1.ELxenU or later.
- OEL 4.7 x64

  GA bits works fine
- OEL 5.3 ia32 and x64

  GA bits works fine

Oracle JRockit supports both hardware and para-virtualized versions and both OEL 4 and OEL 5.

4.11 Triggering Young Collections if the Nursery is Too Small

If the nursery is too small, Oracle JRockit might begin triggering young collections, “back to back", without promoting anything. This appears in the `-Xverbose:memdbg` output as repeated young collections where the number of promoted objects is zero. It can also be seen as very short times between the young collections (close to 0 ms).

Workaround:

Increase the nursery size. If nursery size has been set automatically by `-Xgcprio:throughput`, it can be overridden by manually setting `-Xns` to a higher value.

4.12 SSE2 Registers Might Not be Restored Correctly After Return from Signal Handler

Due to a Linux kernel bug, certain SSE2 registers might not be restored correctly after returning from a signal handler. This issue manifests itself as such undefined behavior as erroneous floating values in Java code and crashes.

Workaround:

This problem is fixed in mainline kernel version 2.6.xx. You can obtain patches for OEL 4 and OEL 5 as RPM'S from the Unbreakable Linux Network. Follow normal kernel upgrade procedure to obtain the fix.

For older kernels, use the command-line option `-XX:+UseMembarForTransitions`.

4.13 System Crashing when Stack Expansion Uses Randomized Address Spaces

On some newer Linux systems (for example, SLES 11) you might experience crashes related to stack expansion when using randomized address spaces.

Workaround:

In Linux, you might be able to eliminate these crashes by using the kernel configuration command `sysctl -w kernel.randomize_va_space=0`.

In Oracle JRockit, you can eliminate these crashes by using the JVM command-line option `-XX:+TrustPThreadStackInfo`. The flag defaults to false.

4.14 Large Pages on Solaris Might Cause Long Pauses

Using large pages on Solaris might occasionally cause long pauses. These pauses happen when a page is accessed for the first time.

Workaround:

Disable large pages by using the command-line option `-XX:-UseLargePagesForHeap`.

4.15 Calculation-Intensive Applications Returning Corrupt Register Values

Floating point calculation-intensive programs run on top of the Oracle JRockit JVM might result in bogus results. This happens because a bug in the Linux kernel does not preserve some CPU registers when switching between tasks. Oracle makes a patch available for OEL 4 and 5. For OEL 4 you need OEL 4.8 with updated kernel (2.6.9-89.0.18.0.1.EL or later). For OEL 5 you need an updated kernel (2.6.18-164.9.1.0.1.el5 or later). The fix is included in OEL 5.5. Novell also makes a fix available (BugZilla number 573478). The fix is available for SLES 9 SP4; SLES 10 SP2 and SP3; and SLES 11. The issue has also been reported to RedHat (BugZilla number 547893). Contact RedHat support for access to this fix.

4.16 R28 Not Supported On Windows 2008 With More Than 64 Processors

Oracle JRockit does not support more than 64 logical processors on Windows Server 2008 and Windows Server 2008 R2.

4.17 Out of Memory Error Occurs When Classblock Memory Runs Low

On Solaris/SPARC, due to the way classblock memory is reserved, Oracle JRockit might occasionally run out of memory when a large number of classes are loaded (in the order of 100000).

Workaround:

Use the following command-line options:

```
-XX:+UnlockDiagnosticVMOptions  -XX:MaxClassBlockMemory=xxM
```

The default value of `-XX:MaxClassBlockMemory` is 50 MB, and a reasonable value is around 75 MB.

4.18 IllegalArgumentException from TLS handshake

security-libs/javax.net.ssl

A recent issue from the JDK-8148516 fix can cause issue to some TLS servers. The problem originates from an `IllegalArgumentException` thrown by the TLS handshaker code.

```
java.lang.IllegalArgumentException: System property
jdk.tls.namedGroups(null) contains no supported elliptic curves
```

The issue can arise when the server does not have elliptic curve cryptography support to handle an elliptic curve name extension field (if present). Users are advised to upgrade to this release. By default, JDK 7 Updates and later JDK families ship with the

SunEC security provider which provides elliptic curve cryptography support. Those releases should not be impacted unless security providers are modified.

See JDK-8173783

See JDK-8148516