numecent®
CLOUDPAGING DELIVERS

TECHNOLOGY TOPIC

HOW TO SOLVE COMMON PROBLEMS WITH LEGACY ORACLE® JAVA® VERSIONS IN WINDOWS® 10

Legacy applications programmed in Java are having trouble when introduced to a Windows 10 environment. While Java is widely adopted software programming language with great benefits such as being platform-independent, there are often dilemmas when running the most recent versions of Java within an environment containing legacy software requiring older versions of Java.

Written by
Dan Kobayashi
Lead Software Engineer, Numecent, Inc.

numecent

WINDOWS BACKGROUND

With Windows 7 end-of-life approaching in 2020[1], many corporations have begun to plan their migration strategy to Windows 10. As these corporations are finding, one of the most difficult migration problems is with legacy software, such as Windows XP software which worked on Windows 7, no longer being compatible with Windows 10. One of the main reasons that legacy software is not compatible with Windows 10 is due to dependency conflicts and library incompatibilities.

COMMON DILEMMAS BETWEEN JRE AND WINDOWS 10

Many Windows 10 issues are apparent with applications programmed in Java. Java is widely adopted software programming language with great benefits such as being platform-independent, meaning that program written in one platform can run across many different device-types or operating systems.[2] Not only is Java a programming language, but it is also a software execution platform. As a result, Java Runtime known as JRE (Java Runtime Environment) must be installed on the client device and should be kept up-to-date, which poses its own problems when running legacy applications written many years ago using older versions of Java. Some of the dilemmas between running the latest versions of Java versus older versions of Java are security risk, compatibility issues, and browser supports.

SECURITY

The latest version of Java is typically the recommended version to use as it contains the latest feature updates, addresses most recently known vulnerabilities, and improves on performance over previous versions. While new features and latest performance improvements may not be relevant to

---

[1] Microsoft, "Windows lifecycle fact sheet," 4 2018.
https://support.microsoft.com/en-us/help/13853/windows-lifecycle-fact-sheet
[2] A. Rongala, "Benefits of Java over Other Programming Languages," 7 3 2015.
https://www.invensis.net/blog/it/benefits-of-java-over-other-programming-languages/

the existing systems, using the older version of Java may leave critical security risks in your organization. Between 2016 and 2017, there were 105 published security vulnerabilities found in JRE, out of which 16 were classified as high or critical severity.[3] Most of these high-risk vulnerabilities have been addressed by updated versions of JRE, meaning that running older versions of JRE without the fixes potentially exposes your organization to these risks.

COMPATIBILITY

One of the reasons Java is popular and is widely adopted is in its backwards compatibility.[4] In majority of cases the newer versions of Java are binary compatible with the older versions of Java, meaning that programs written in older versions will run on newer version. For example, between Java 7 and Java 6, there was only one compatibility change in a rarely used function that could cause a program written in Java 6 to completely not run when using Java 7.[5] However, this is far from the real-world scenario.

Typically, legacy applications will run with the latest versions of JRE except for few functionalities, or it may run completely but produce different results. These incompatibilities are known as behavioral incompatibilities and are usually caused by subtle changes in Java libraries and their API (Application Program Interface). While they do not completely breakdown the application, it can make the application unusable due to instability or producing wrong results. Normally, the only way to fix such issues is by updating the application to its latest version which has been verified by the software vendor as compatible with the latest Java version.[6] While upgrading the application is the most straightforward way to resolve compatibility issues,

---

[3] "CVE Details."
https://www.cvedetails.com/vulnerability-list/vendor_id-93/product_id-19117/Oracle-JRE.html
[4] S. Ritter, "4 Reasons Why Java is Still #1," 12 1 2016.
https://www.azul.com/4-reasons-java-still-1/
[5] Oracle, "Java SE 7 and JDK 7 Compatibility."
http://www.oracle.com/technetwork/java/javase/compatibility-417013.html
[6] S. Mostafa, R. Rodriguez , X. Wang, "A Study on Behavioral Backward Incompatibility Bugs in Java Software Libraries," IEEE, 2017.

sometimes it is infeasible due to the application no longer being supported, or the high cost associated with the upgrades.

BROWSER SUPPORT

The future of browser support for Java is grim. Most modern browsers such as Google Chrome, Mozilla Firefox, and Opera no longer support running Java within the browser Window. For Microsoft Edge, it never supported Java from the beginning. The only options left for Java applications that runs on browsers are older browsers such as Internet Explorer or the long-term release of Firefox, besides using third-party extensions which may enable Java on modern browsers with some quirks.[7]
While running older browsers on the newest OS such as Windows 10 is already a challenge, using older browsers will soon not be enough to run these Java applications. Oracle has announced the end of life for Java Web Start, a protocol used by many Java applications that are launched from the browser.[8] Starting with Java 11, which is Oracle's next long-term support version releasing in September 11, 2018, Java Web Start will be deprecated.

INSTALLING MULTIPLE VERSIONS OF JAVA RUNTIME

Today, problems once known as "DLL Hell" are, for the most part, resolved thanks to registration-free COM objects and WinSxS (Windows Side-by-Side) directory.[9] However, system administrators now need to manage multiple versions of Java Runtime and this can quickly get out of hand, especially for managed desktop environments such as VDI. Typically, this type of environment requires interventions by IT staff either through rigorous scripting managing multiple environment variables pointing to various versions of JRE installed in different directory paths. It is important to note

---

[7] C. Neagu, "How to enable Java in all the major web browsers," 05 04 2017.
https://www.digitalcitizen.life/how-enable-java-all-major-web-browsers
[8] Oracle, "Oracle Java SE Support Roadmap," 28 06 2018.
http://www.oracle.com/technetwork/java/javase/eol-135779.html
[9] Wikipedia, "DLL Hell."
https://en.wikipedia.org/wiki/DLL_Hell

that by having multiple JRE installed at the same time, the security risks are potentially magnified due to different versions of JRE having different sets of security vulnerabilities.

The following section describes how Cloudpaging addresses dependency conflicts and library incompatibilities in various software programs.

HOW CLOUDPAGING ADDRESSES DEPENDENCY CONFLICTS AND LIBRARY INCOMPATIBILITIES

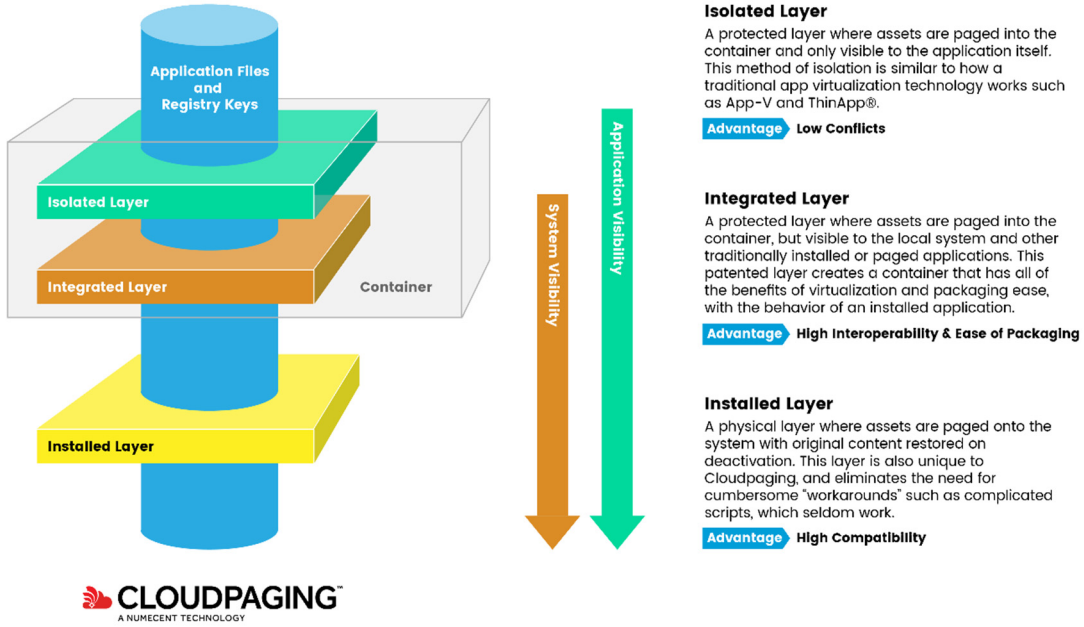

Figure A. Cloudpaging's configuration solves the application compatibility problems many companies face. During the application packaging process, Cloudpaging creates unique dispositions for isolating or integrating software and the end user's operating system. This results in maximum application compatibility and functionality, whether the end users' desktops are virtual or physical.

Cloudpaging technology is capable of isolating troublesome or legacy components, using its patented Virtual Disposition Layers, as illustrated in **Figure A.** This technology allows Cloudpaging to deliver legacy Java Runtime

such as JRE 6 in an isolated environment, such that it does not conflict with the latest version of Java Runtime installed on the system. Cloudpaging can allow or disallow certain applications (or the system itself) from seeing the isolated JRE, therefore achieving a solution where multiple applications are each seeing a different version of JRE without conflicts. By limiting and restricting the use of older JRE you can reduce the risk of security vulnerabilities and with minimum interventions from IT staff.

Following are examples of the ways in which Cloudpaging addresses dependencies and conflicts between JRE and Windows 10.

ISOLATION THROUGH APPLICATION PACKAGING

The simplest way to isolate JRE is to package it alongside the application that requires it. The JRE is packaged into Cloudpaged container, and it is virtualized on to the client desktop in an isolated layer making it visible only to the application inside the container. Multiple packages can be deployed to the same desktop and each of the application will only see the JRE packaged with it. Legacy browser can be packaged this way with Java to overcome the issues of limited Java support in modern browsers.

CUSTOMIZED VIRTUAL CONTAINERS

Native application can be launched as a virtual process in order to allow accessing the contents inside the virtual container. In this case, the package only contains the JRE and the rules restricting which processes can see inside the container. The allowed processes can then run as virtual which will allow it to see the legacy JRE inside the virtual container instead of the latest JRE installed on the system. By leveraging this technology, IT administrator can allow natively installed Internet Explorer to use the latest version of Java for their daily work, and force IE to use the legacy version of Java only when accessing specific applications that require older JRE.

numecent

CONCLUSION

Running legacy applications on the latest OS can be a challenging task particularly when the application requires Java Runtime Environment. There are workarounds, but they typically require cumbersome scripting and manual interventions. The chosen solution can also put the system in a less secure state by installing JRE with known security vulnerabilities.

Using Cloudpaging, customers can create an environment where multiple versions of JRE are available on the same system and being used for different purposes to run modern applications and legacy applications that would normally be in conflict. The system integrity can be maintained by restricting access to potentially vulnerable versions of JRE.

To learn more about Java packaging technique using Cloudpaging, contact Numecent.

ABOUT THE AUTHOR

Dan Kobayashi is a Lead Software Engineer for Numecent. He has worked with various iterations of Numecent's technology for over 10 years. Dan has extensive experience with agile software development and reusable software design as well as cloud app hosting, management, and deployment. Dan graduated from University of California, Irvine and received a B.S. Magna Cum Laude in Information and Computer Science.

numecent

ABOUT NUMECENT

Numecent is a pioneer and technology leader in the rapid, secure, and friction-free provisioning of native software applications from the cloud or on-premises. Working across a range of sectors, Numecent's ground-breaking Cloudpaging technology offers a new paradigm for application delivery. It delivers native applications from the cloud between 20 – 100 times faster when compared to a linear digital download, while it can execute on the client's platform without the need for installation, ensuring absolute efficiency and ease of use. Numecent's primary customers include Cloud Service Providers (CSPs), Independent Software Vendors (ISVs), Managed Service Providers (MSPs), and Enterprises.

Numecent introduced its proprietary Cloudpaging platform and emerged into the market in 2012. The company's headquarters are located in Irvine, California with partners located throughout the world.

For more information, please visit www.numecent.com.