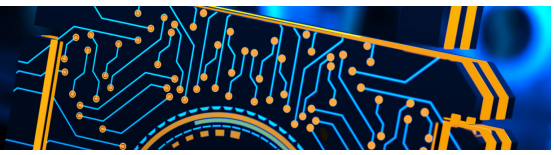
White Paper

Ignition and Java Security: Why and How Ignition Uses Java to Create Secure, Cross-Platform Systems

inductive automation
800.266.7798
www.inductiveautomation.com

Ignition! by inductive automation



Ignition by Inductive Automation is the first SCADA software solution built entirely on Java. Ignition's use of Java makes it totally cross-platform compatible and easily web-deployable, two major reasons for the software's growing community of global Ignition users.

Java is a well-established programming language that has been used and trusted around the world for more than 20 years. Java was released in 1995 and quickly became one of the top programming languages in the world. Since 2001, it has held the number-one or number-two spot on the TIOBE Index.[1]

According to Oracle, which has owned and supported Java since 2009, Java is running on 15 billion devices, is used by 10 million developers worldwide, 5 million students are learning Java, and it's the number-one platform for development in the cloud.[2]

Java is used by thousands of major companies. Consider that IBM uses Java across all of its computing platforms.[3] IBM mainframes are used by 92 of the world's 100 largest banks and 9 out of the world's 10 largest insurance companies; and by 71% of global Fortune 500 companies.[4]

Why is Ignition Built on Java?

At Inductive Automation, our mission is to develop industrial automation software that pushes the boundaries of innovation and empowers our users. We understood that technologies from the IT world had great potential to revolutionize the industrial world. Ignition leverages Java and other IT technologies such as SQL to enable companies in almost every industry to do more with new and existing infrastructures.

Our first and foremost reason for using Java is that it's based on the concept of "write once, run anywhere." Its portability enables Ignition to be a cross-platform solution which users can run on any operating system.[5]

Also, Java is network-centric, which gives Ignition its web-deployed capability. Developers can work with resources across a network and create network-based applications using client/server architecture.

Another reason is that Java is well-supported. The Java community is over 10 million developers strong and continues to grow, so there is a gigantic talent pool of skilled Java developers. Java has an extraordinarily rich set of libraries, so there are a phenomenal number of resources available for developing with Java.

These are all factors that let us remain nimble and make Ignition the product that our customers demand.

What About Security?

We are confident that Java continues to be the right choice for Ignition and delivers the performance our growing community of users is looking for. Its continued, widespread use by major companies and its popularity among programmers are a testament to its security and reliability.

As we'll explain in the next part of this paper, there are many different parts within Java, and the security issues are generally limited to the Java browser plug-in, not the Java language or API. Oracle has met these issues head-on by removing the Java Web Browser Plug-In from future versions. Furthermore, most web browsers have either stopped supporting the Java Web Browser Plug-In or disabled it by default. Ignition clients do not require Java in the web browser in order to run.

If you'd like more detail about these issues, please read the rest of this paper for a deeper look inside Java and some steps you can take to keep your Ignition system safe.

What is Java? Part 1: The Language

Before we get into the details of the recent security issues, it will help to have a solid understanding of Java technology. Java, first and foremost, is a programming language. It is a general-purpose language that a computer programmer can use to write any sort of program. In this respect, it is similar to C, C++, .NET, Python, and many other programming languages.

Programming languages can broadly be divided into two kinds: compiled or interpreted (*see diagram A-1*). Compiled languages use a compiler to turn the source code into an executable binary.

Interpreted languages run source code directly through an interpreter, skipping the compilation step but requiring an interpreter to be present to run the application.

Java uses a slightly different model (*see diagram A-2*). It is a compiled language, but instead of compiling into machine code like traditional compiled languages (think C), it compiles into "bytecode."

Bytecode is like machine code for an imaginary type of machine: not x86, not ARM, not any real CPU that has ever been built. It runs on a virtual machine, which is a special program that translates the bytecode into machine-specific instructions on the fly.

The main advantage of the traditional model is speed, with the trade-off being portability. You need to recompile your program for each different kind of computer and operating system you want to run on.

The main advantage of the interpreted model is that you typically get fancier language features, and assuming there is a version of "interpreter.exe" available for different kinds of systems, you get portability as well. Java is a blend of these two models.

By using compiled bytecode and a virtual machine, you get speeds that approach or often match the traditional model, and you get portability as well because there is a version of "jvm.exe" for different architectures and operating systems.

What is Java? Part 2: The API

Writing source code and then compiling it or interpreting it or running it through a JVM (Java virtual machine) gives you the ability to execute instructions

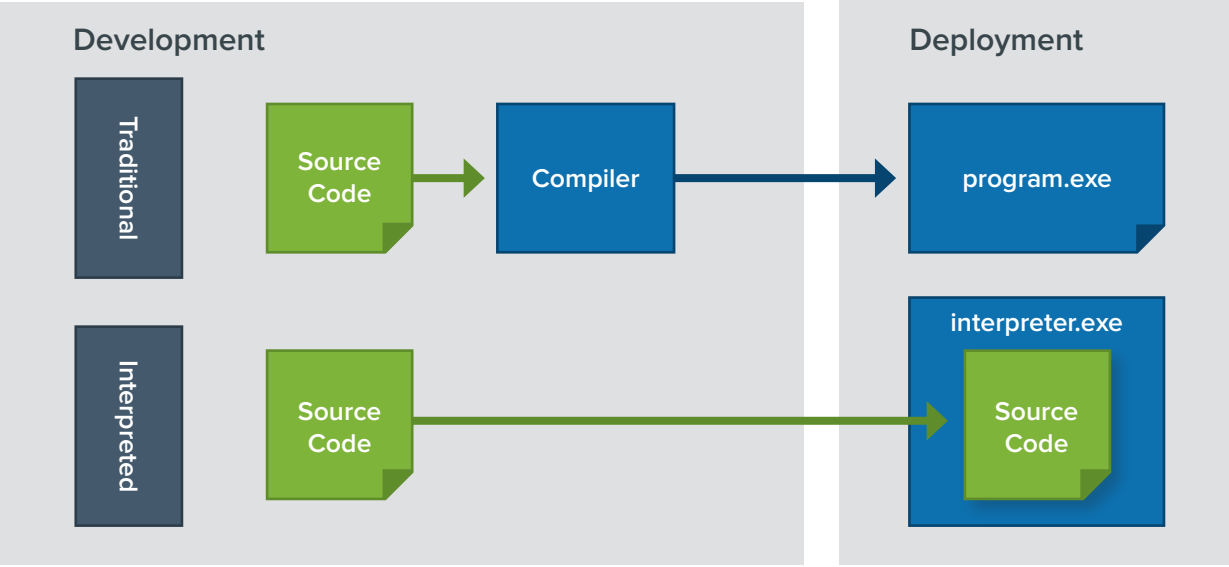


Diagram A-1

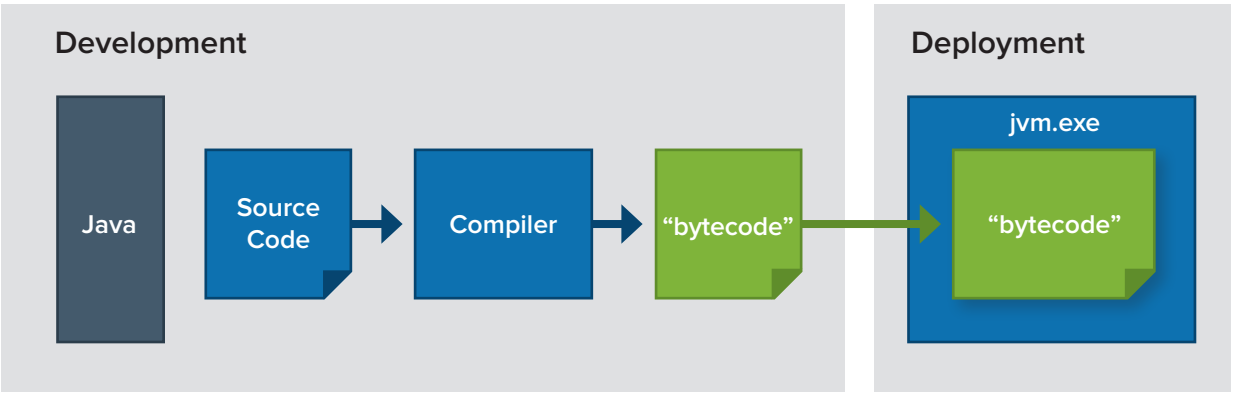


Diagram A-2

against a computer's CPU. By itself this is nearly useless. Without some access to the operating system, your program can't do much of anything.

Therefore, most programming languages include a rather extensive API (application programming interface). This is a built-in library of programming functions that give the programmer access to computer resources such as the filesystem, the network, and the graphical user interface.

It is this API that truly makes Java such a portable programming language. Let's look at an example: Suppose you want to write a program that edits a text file. If you wrote this program in C or C++ on Windows, you'd be programming against the Windows filesystem API. If you wanted to compile a version that was compatible with Linux, not only would you have to re-compile, but you would have to alter your program to be coded against the Linux filesystem API. For a complex program, this "porting" of the code from one OS to another can be extremely costly and complex.

You can see how adding support for additional operating systems and CPU architectures greatly increases the complexity *(refer to diagram B-1)*.

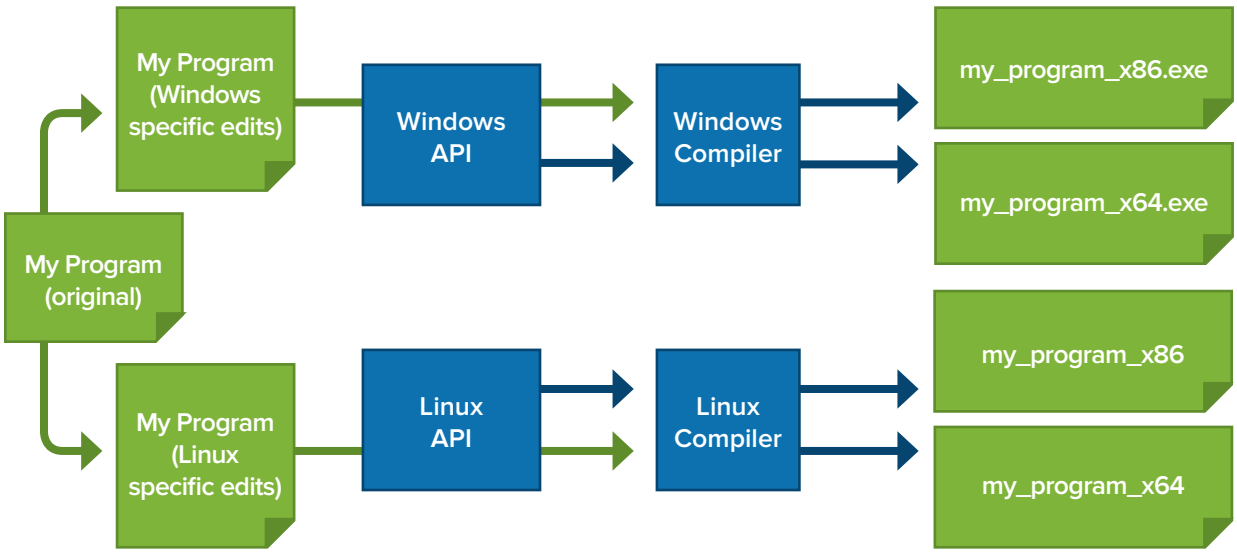


Diagram B-1

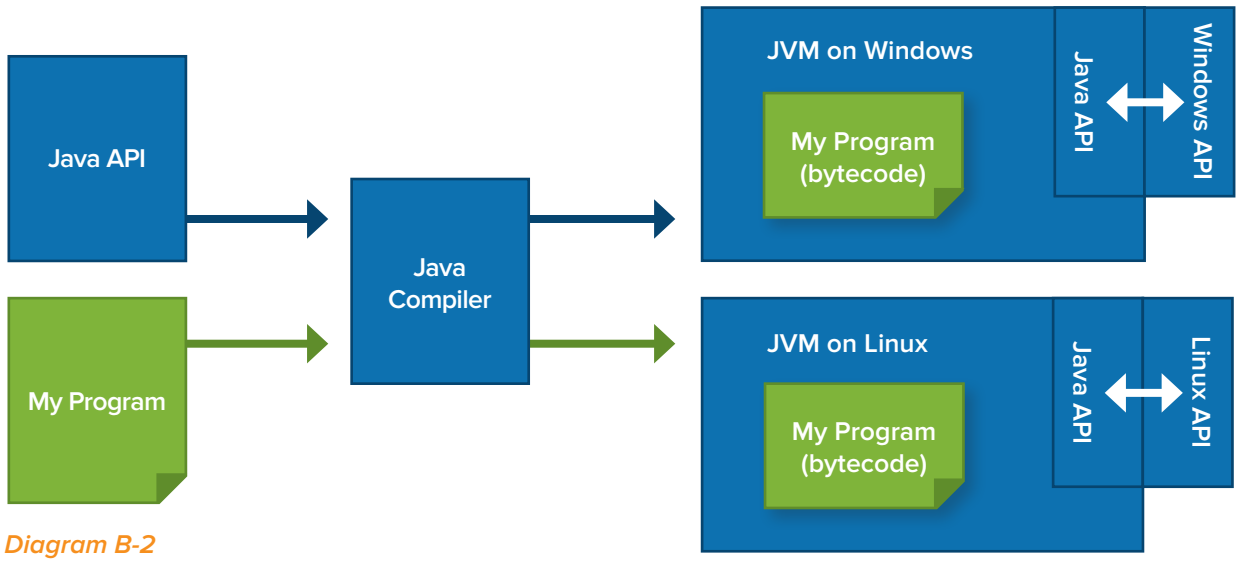


Diagram B-2

The Java API removes the need for this style of manual porting by providing an abstraction layer. Instead of dealing directly with the Windows filesystem API or the Linux filesystem API, a Java programmer only deals with the Java filesystem API. The virtual machine then handles translating between this "virtual" API to the true operating system API (*see diagram B-2*).

What is Java? Part 3: Java Applets and Java Web Start

Beyond the language, compiler, virtual machine and API, Java has a number of ancillary technologies that surround it. Applets and Java Web Start (JWS) are two such technologies used for the same purpose: making it easy to get Java programs deployed and running on your computer.

Applets (*see diagram C-1*), the older of these two, were invented in 1995 as a way to include richer content inside a webpage. Remember that this was way before HTML5 and AJAX, when the capabilities of a basic webpage were very limited. Originally this functionality was built into the browser itself to download Java bytecode referenced by a webpage and run it. Later, it became the responsibility of a browser plug-in provided by Java to load and run the bytecode for these applets.

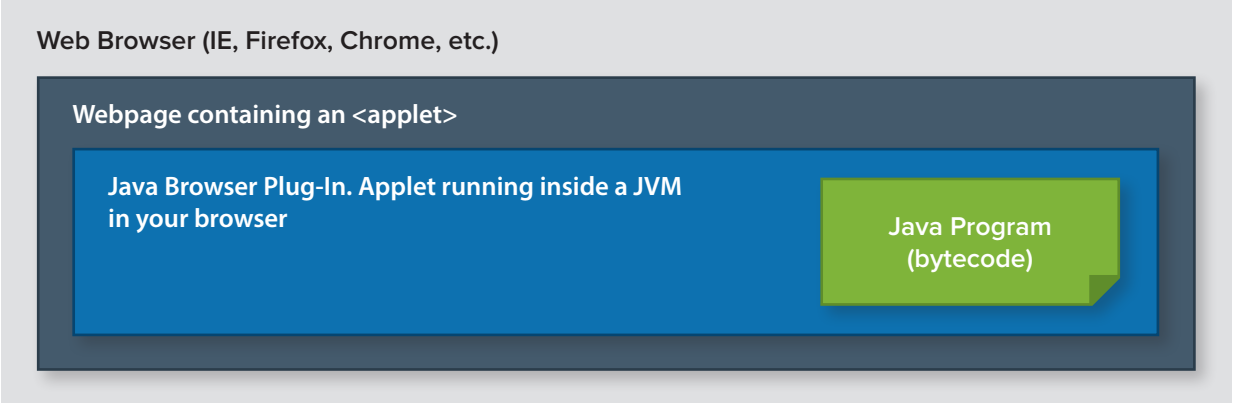


Diagram C-1

Java Web Start, introduced in 2001, is a way to get a full-blown Java program installed and running on a client computer without requiring an installer. From the user's perspective, they click on a link, and the application downloads itself and runs. This was preferred over applets for more complex applications that were better not stuck inside a web browser.

Java Web Start is a remarkably simple idea. Instead of relying on a plug-in to the web browser, it simply registers a file association in the operating system for the filetype "JNLP" (Java Network Launch Protocol).

As an example, this mechanism is similar to how Adobe registers an association with the filetype "PDF" so that when you download a PDF, it opens using Acrobat Reader.

JWS opens the JNLP file when you click on a Web Start launch link, which describes where to find the binaries (bytecode) that makes up the Java program you want to launch. Then it downloads the binaries and starts up a JVM to run the program.

Note that no browser plug-in is required for this functionality (*see diagram C-2*).

Security Concerns Explained

Java itself, as a general-purpose programming language, can't really be considered "secure" or "insecure." Consider that most viruses are written in C, but nobody considers C itself to be "insecure"; it's just a tool. This isn't to belittle the security concerns, but rather to underscore exactly where the security issues lie. Java programs themselves are actually quite secure, in that they are immune to one of the most common types of attacks (buffer overruns).

Java is also just a tool: You could just as easily write a virus in Java as in C. The problem, therefore, is not with Java itself, but rather in the way Java can be deployed.

If someone you did not know emailed you a file named "run_me.exe", would you execute that program? Obviously not. We use an array of virus checkers, filtering routers, and common sense, all for a common purpose: making sure that untrusted programs don't get a chance to run on our computers.

What about Java applets? Here's a technology whose entire purpose is to run a program on your computer when you visit a webpage! Unsurprisingly, it is this deployment scheme that has been the cause of the recent security breaches involving Java.

When applets and JWS (Java Web Start) were designed, the inherent security risk was not overlooked. To address this issue, Java used the idea of a "sandbox." A sandbox means that an untrusted Java program being deployed as an applet or through JWS would be allowed to run, but only with access to a restricted set of the Java API. Notably, the filesystem and the network would be off-limits, unless the applet was trusted. Simply stated, the recent security breaches have all involved running untrusted Java programs designed to exploit flaws in order to break out of the sandbox.

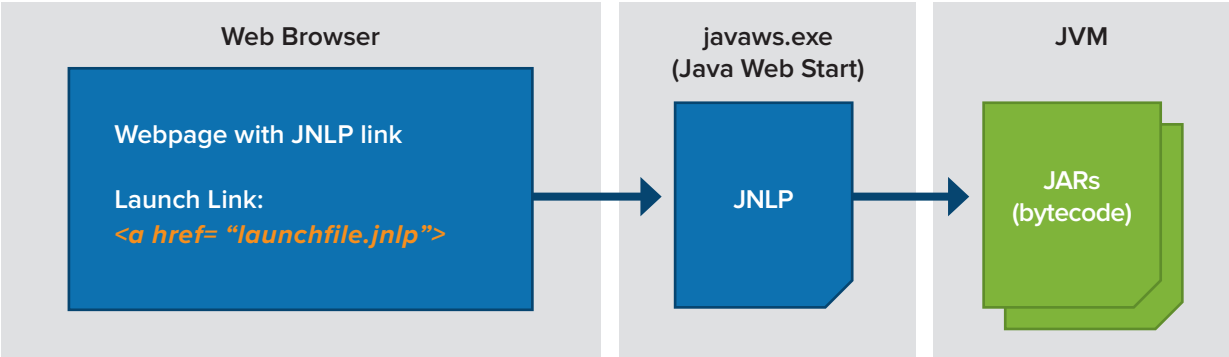
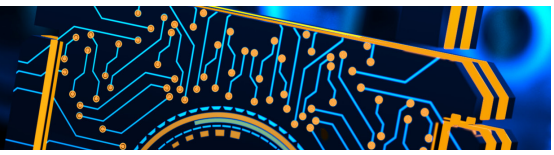


Diagram C-2



The end result is an untrusted program running on your computer, without your consent. Long story short: the sandbox system cannot be trusted.

If the sandbox doesn't work as advertised, how can you protect yourself from this issue? One option would be to disable Java applets and JWS entirely. This "nuclear option" is effective, but unfortunately will compromise your ability to conveniently use Ignition, since the Ignition Designer and Ignition clients use JWS to launch.

Fortunately, your other option is quite good. Simply set Java's security settings to not run untrusted applications at all. This is the default setting for Java security. Instead of running untrusted applications in a sandbox, you can simply opt to not allow them to run at all. Ignition, like any reputable Java application, uses digital signatures to prove that it is a trusted application.

You may also choose to disable the Java plug-in within your browser (refer to instructions below). This will prevent applets from running at all, completely sidestepping the entire Java security issue. If you do this, launching an Ignition client as an applet will not work, but very few customers choose this launch option anyhow.

The standard launch options (Windowed and FullScreen) which use JWS will work fine without the Java browser plug-in.

Trust and Digital Signatures

Keeping Java Web Start enabled for only trusted applications is Oracle's recommendation for Java, as well as ours. How is this trust established? We'll take a moment to explain how digital signatures work, and why they can be used to establish trust.

Java applications can be "signed" with a digital signature. When they are signed, they can request extra permissions: to be elevated out of the sandbox and run as a trusted application, with access to the filesystem, network, etc. Java first confirms with the

How to Disable the Java Plug-In in Your Browser

For additional protection, you may disable the Java browser plug-in from within your browser. This will prevent websites from loading Java applets within web pages, but still allow Java Web Start to function. The directions for disabling the Java browser plug-in are different for each browser:

Chrome
Type "chrome://plugins" into your address bar. Find the "Java" item on the list and click below it where it says "Disable" in blue. Restart your browser.

Firefox
Choose the "Add-ons" item under the "Tools" menu. Select "Plug-ins" on the left side. Find the item in the list called "Java (TM) Platform ..." Click the "Disable" button on the right. Restart your browser.

Internet Explorer 8, 9, 10
Choose the "Manage Add-ons" item under the "Tools" menu. On the left of the popup under "Show," select "All Add-ons." Find the entries in the list under the group "Oracle America, Inc." Select each item (if more than one) and disable with the disable button in the bottom right-hand corner. Restart your browser.

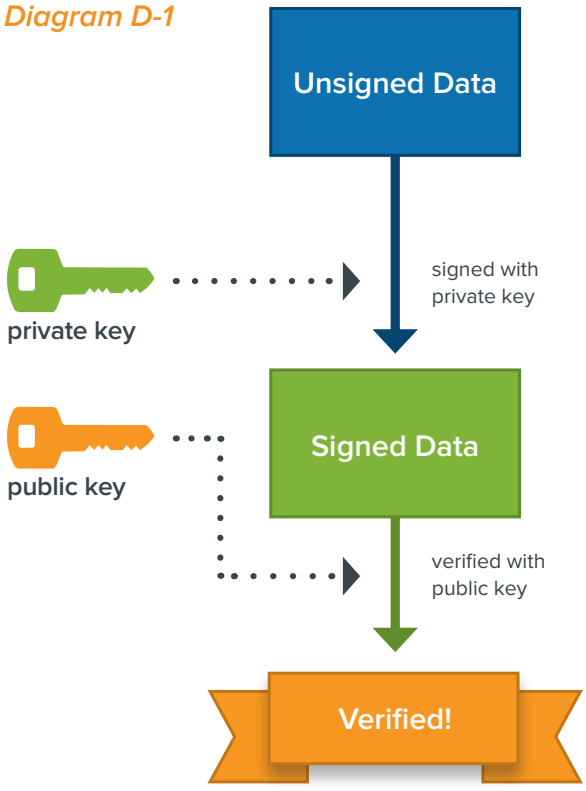
Safari
Choose "Safari" and then "Preferences" on the taskbar. Click "Security" on the top of the window that appears. Uncheck the box "Enabled Java." Restart your browser.

user that they trust the author of the application. It is the identity of the author that confers the trust. The digital signature ensures that this identity cannot be forged.

To understand how this works, you must understand something called PKI (Public Key Infrastructure) and its underlying technology: asymmetric encryption (*see diagram D-1).*

Imagine you have two keys. These keys are special, in that if you lock something with one key, it can only be unlocked with the other. Now, you keep one key private, but freely make copies of the other key and distribute them to your friends. This is called your public key. If you want to send a message to a friend, you can lock it with your private key. Anyone with your public key can read the message, but nobody can lock the box with a different message inside. The point is that if your friend receives the locked box, and your public key opens that box, they know that you locked the box. The identity of who locked the box is known, and cannot be forged. This is asymmetric encryption.

Diagram D-1



How does this work to prove the identity of a Java program? (*See diagram D-2.*) We create a keypair with a public and private key. We then send our public key to a certificate authority. The certificate authority's job is to ensure that we are who we say we are. They check us out, make sure we're a real company, talk to us on the phone, etc. When they are satisfied, they issue us a certificate, which they sign with their private key so the authenticity of the certificate can be trusted.

There are only a few certificate authorities in the world, and your computer comes pre-installed with their public keys. Examples are Verisign, Thawte, GeoTrust, etc. When we sign our program with our private key, and include the certificate, we become a trusted identity.

Therefore the end user can trust that 1) we are who we say we are and 2) we signed the program (nobody has altered the program).

Ignition Security

There are other simple steps users can take to secure their Ignition system. Ignition offers security features such as client authentication that give organizations the ability to assign user roles and permissions. Users can further create security zones to allow access to Ignition based on determined criteria. Ignition can leverage Microsoft Active Directory and easily integrate with existing security schemes. Furthermore, administrators can use auditing to track behavior and changes made to a project.

Ignition includes support for SSL encryption, commonly used in the financial industry, to help safeguard communications between clients and servers. This becomes especially useful for remote or wide-area installations with multiple locations.

If you have questions or would like more information about Java security or Ignition security, please contact Inductive Automation Support at: 800-266-7798 (US) or 916-456-1045 (international), or visit: support.inductiveautomation.com

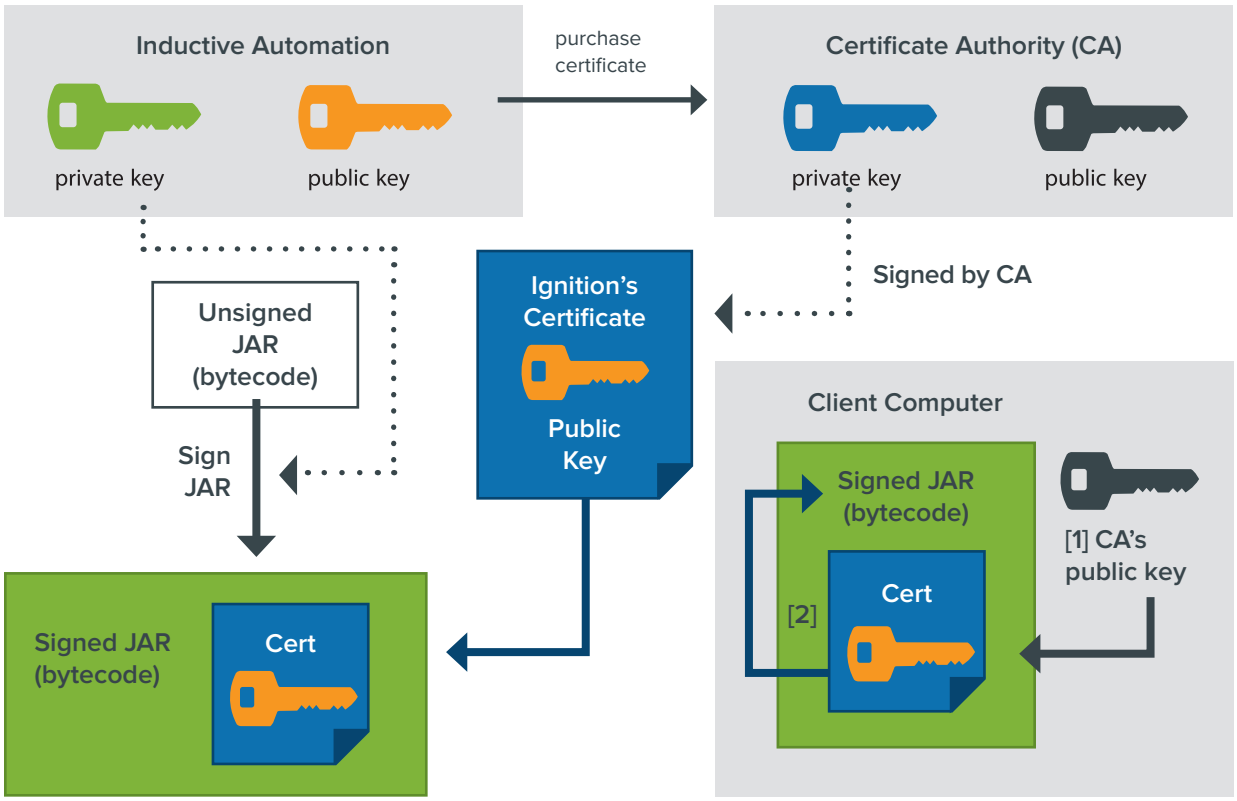


Diagram D-2

[1] verify certificate using CA public key
[2] verify bytecode using Ignition's public key

Footnotes:

[1] Krill, Paul. "Java, C Remain Entrenched as Most Popular Languages." InfoWorld. Nov. 7, 2016.

[2] "Go Java." https://go.java/index.html

[3] "Java on z/OS." IBM Knowledge Center. IBM.com

[4] O'Malley, Chris. "Executive Viewpoint 2017: Compuware – Mainstreaming the Mainframe." Virtual Strategy Magazine. Feb. 3, 2017.

[5] Supported operating systems for Ignition v7.9: Windows Server 2008/2012; Windows Vista, 7, 8, and 10; Ubuntu Linux 12.04 or later; and other Java SE-enabled OSes.