Supporting the Secure Deployment of OSGi Bundles

Pierre Parrend, Stéphane Frénot

Supporting the Secure Deployment of OSGi Bundles

Pierre Parrend, Stephane Frenot
INRIA Ares/CITI, INSA-Lyon, F-69621 France
Phone: +33 (0) 04 72 43 71 29
E-mail: {pierre.parrend},{stephane.frenot}@insa-lyon.fr

Abstract

The OSGi platform is a lightweight management layer over a Java virtual machine that makes runtime extensibility and multi-application support possible in mobile and constraint environments. This powerfull capability opens a particular attack vector against mobile platforms: the installation of malicious OSGi bundles. The first countermeasure is the digital signature of the bundles. We developed a tool suite that supports the signature, the publication and the validation of the bundles in an OSGi framework. Our tools support the publication of bundles onto a remote bundle repository as well as the validation of the signature according to the OSGi R4 specifications. A comparison of existing validation mechanisms shows that our security layer is the only one that is compliant with the specification.

1 Introduction

The OSGi platform is on the way of becoming the de facto standard componentization middleware for supporting extensible software, through the management layer it provides to control the life-cycle of the so-called 'bundles', *i.e.* the OSGi components. However, the security characteristics of this life-cycle are hardly identified, in particular during the deployment of the bundles which is usually realized over insecure networks. We present here the tools we developed to support the secure deployment of bundles. The first tool, SF-Jarsigner[1], covers the early life-cycle phases of the deployment, namely the signature of the bundles by their issuer and their publication. The second tool, SFelix[2], is actually an implementation of the digital signature validation layer of OSGi Release 4 specifications, and is based on the Felix[3] OSGi implementation. Both are compliant with OSGi Release 4 [10]. The publication process is not defined by these specifications, but relies on the Open Bundle Repository (OBR) format [11], which is supported by several Open Source OSGi implementations: the OBR1 format is supported by Oscar and Knopflerfish, and the OBR2 format is supported by Felix.

The OSGi platform is a lightweight overlay to the Java virtual machine, which supports the runtime installation of Java components, the management of their life-cycle, as well as the proper expression of the dependencies between them. This facility typically allows a user to discover software packages that are available on the Internet or in its near environment, to install them together with their dependencies, and to un-install them when they are no longer required. The runtime extensibility which is thus provided opens a new attack vector, which to the best of our knowledge has been overlooked in the literature: the possibility of seamlessly executing malicious code from the environment. Figure 1 shows the attack tree for executing malicious code on a client system [13]. Two main strategies exists: to install a malicious platform, and to install malicious bundles. The first strategy is prevented by the control of the integrity of the framework when it is installed. The second strategy can be realized through three different approaches: inserting malicious code during the development, inserting malicious bundles onto the bundle repositories, or installing bundles from unauthorized repositories. The bundle issuer is responsible for guaranteeing that the code he provides is sound, which can be achieved through code analysis. The insertion of malicious bundles in the repositories or the installation from unvalid repositories can be prevented by the digital signature of bundles. This paper presents the tools we developed to perform this digital signature. Further information is provided in the related technical report [12].

The main current use cases of the OSGi platform is the deployment of Open Source software. In such environments, security problems are often neglected due to the open structure of the projects. However, the broadening of the applications of the OSGi framework and the ben-

[1]This work is partially founded by MUSE IST FP6 Project #026442.
[1]http://sf-jarsigner.gforge.inria.fr
[2]http://sfelix.gforge.inria.fr/
[3]http://incubator.apache.org/felix/

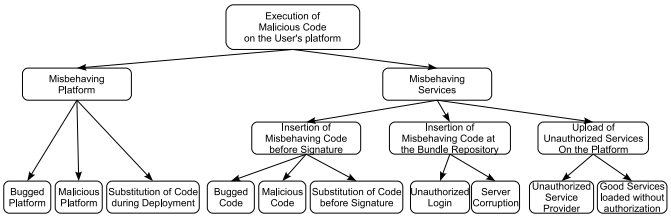

Figure 1. The Attack Tree for malicious Code Execution

efits it may provide to adaptable and extensible business-critical systems are highly likely to require a firm understanding of its security implications, and the availability of effective tools. The security infrastructure, which relies in a great part on the digital signature of bundles, is common to all application domains, such as mobile commerce, wireless remote control, health-care monitoring, and video-surveillance. This infrastructure must be completed with context- and application-specific security policies, which are out of the scope of this paper.

The following of the paper is organized as follows. Section 2 presents the related works. Section 3 gives the detail relative to our tool suite. Section 4 provides a comparison of the various available tools and frameworks dedicated to signing and validating the digital signature of bundles. Lastly, we conclude this work and give further perspectives.

2 Related Works

So as to define a secure process for deploying OSGi bundles, it is necessary both to control the overall deployment process, as well as the security mechanisms that can be used to protect it. We therefore present other research works related to the deployment process, and to the security aspects of deployment.

2.1 Deployment of Components

The various steps of the life-cycle of software components and of OSGi bundles in particular are the following [1]: the publication of the components, the discovery of the components, the resolution of dependencies between the components, the download, and the configuration at the initialization step.

The publication step requires that both the components and the information necessary for the client to execute them properly are made available: the component meta-data, dependencies between components, the source location and the scope of the component release are published in 'release databases' [16]. The binding between the publishers and the clients is done either through the publication of static meta-data [11], or through a publish/subscribe mechanism [17]. In both cases, a third party broker is required. The discovery phase is made of the identification of the available bundles by the clients, and the resolution of the dependencies. The Eureka framework is an example of such a discovery mechanism over the OSGi platform [14]. Specific languages are defined to support this process of dependency resolution [7]. The download phase is typically done from a centralized repository [11]. However, the distribution of the resources over peer-to-peer overlay networks can be used with benefit, so as to increase the robustness of the infrastructure, the availability of the components[3].The last phase of the deployment is the configuration of the components on the client. It builds the finalization phase of the deployment of complex distributed systems, and is often overlooked when deploying single applications [6].

The tools we propose are based on the Bundle Repository Metadata to support the extraction of metadata during the publication, and their resolution by the clients. The download is handled by the 'bundle repository' bundle at the client's, which is part of the Felix framework. We integrate these existing facilities together with a publication management tool we developed. This latter deals with the update of the meta-data of the bundle repository and the upload of the bundles. To the best of our knowledge, such a tool is available neither in the frame of the Felix project, nor in the Equinox platform.

2.2 Secure Deployment

The cardinal security properties are the integrity of the data, the authentication of their emitter, the confidentiality of the communications and the non-repudiation of the actions. Diverse approaches have been defined so as to enforce these properties on the context of component deployment.

The current specification for securing the deployment of bundles is based on the Java Archive specification [15], and defined with precision in the OSGi Release 4 specifications [10]. It consists in the integration of the integrity control for each resource through a hash value and a digital signature in the bundle itself. The algorithms used are SHA-1 for hashing and DSA for the signature. The necessary meta-data is encapsulated in a file compliant with the 'Cryptographic Message Syntax' [8]. This approach provides the authentication of the bundle issuer and the guarantee of the integrity of the code.

An alternative to signature based on asymmetric cryptography is the use of Message Authentication Code (MAC) based on symmetric key [9]. This provides a more lightweight cryptographic mechanism, but is often considered as being less secure than asymmetric cryptography, be-

cause the secret key is shared among several entities. Moreover, no performance indication is given, which means that no factual comparison can be done with current specifications. The MAC signature process can take benefit of the XML signature encapsulation for easier management [9].

A more robust mechanism for guaranteeing the deployment of component is the S-CODEP (SECure COmponent DEPloyment) protocol [5]. It is based on Kerberos, and provides an anti-replay mechanism. It assumes that the underlying platform can not be compromised. The use of Kerberos make this approach a heavyweight one. It is adapted for sensitive systems such as enterprise information systems or telecommunication environments.

Several security infrastructures for specific component platforms have been defined. For instance, the Cingal model is a component model very similar to the OSGi one, and provides digital signature using the same principle of digital signature of the bundles [2]. A similar mechanism has been defined to support the deployment of Web Services [4]. The .Net framework also proposes a similar mechanism to sign assemblies. Its main limitation is that the installed assemblies can not be removed, contrary to OSGi bundles, which makes it unproper for mobile and resource-constraint platforms.

These research works related to the problem of securely deploying OSGi bundles and other software components suffer from one obvious drawback. There is currently to the best of our knowledge no tools that supports the deployment process as it is specified by the OSGi Alliance. This matter of fact makes any comparison impossible - and greatly limits the use of the OSGi platform in environments where security is required. We therefore developed those tools, and present them in the subsequent sections.

3 The SFelix Tools suite

So as to support the process of secure deployment of OSGi bundles, we developed two complementary tools. The first is the SF-Jarsigner that performs the signature and the publication of the bundles by their issuer. The second is SFelix, which is an extension of the Felix implementation of OSGi. Our main contribution in SFelix is the implementation of the OSGi R4 security layer, which is the sole project we know that supports bundle signature according to the OSGi Specifications [10]. For more precision, please refer to our technical report [12].

3.1 Secure Deployment

The process of secure deployment is shown in figure 2. The deployment of OSGi bundles is compound of issuance phase, and a client-side phase.

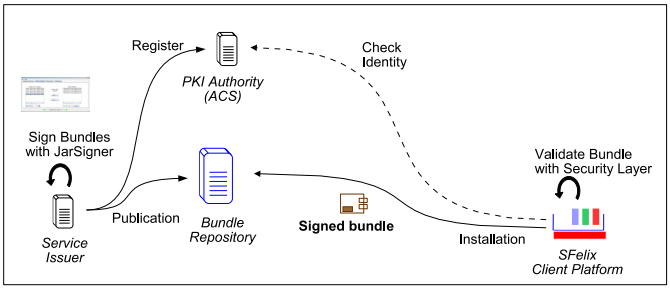

Figure 2. Secure Deployment of OSGi Bundles

The issuance phase is performed by the bundle issuer. The first step is to sign the bundles, or to check that the existing signature is valid. The second step is the extraction of the meta-data of the signed bundles, and the publication onto a third party repository. The format of the meta-data is defined by the OBR2 Request for Comments of the OSGi Alliance [11].

The client-side phase of the deployment is made of the discovery of the bundles, the dependency resolution, the download of the code archives, the validation of the digital signature, and the start of the bundles. The discovery, the dependency resolution and the download steps are dealt with by the bundle-repository facility, which is available in the Felix distribution. The validation step is performed when the bundle is stored locally, to avoid a TOCTOU (time-of-check, time-of-use) substitution, and before it is installed, so that not authenticated bundles can not be executed. It must be compliant with OSGi R4 specifications, but also compatible with the behavior of the virtual machine. The Sun tools are considered as the reference. When the bundle signature is valid, and the signer is identified by the platform as being a trusted one, the bundle can be started.

3.2 SF-Jarsigner: a Tool for secure Publication of Bundles

The SF-Jarsigner tool aims at providing a convenient graphical user interface for signing and publishing OSGi bundles. Since the signature generation mechanism is compatible with the Sun Jarsigner tool, other types of Jar files or data archives can also be signed.

The SF-Jarsigner tool is compound of four graphical panels. The first panel let the archive publisher select a 'keystore' file, where its private/public key pair (respectively a public key) is stored for signing (respectively checking) bundles. The second panel is the one shown on figure 3, it supports the signature or verification of bundles that are stored on the local system. The valid bundles are shown

on the list on the right, and made available for publication. The third panel allows to set and store the informations relative to the remote file server where the bundles are to be published. Currently, only the FTP protocol is supported. The last panel lets the publisher choose among the available signed bundles which one are to be published. The operation can be repeated on several Bundle Repositories.

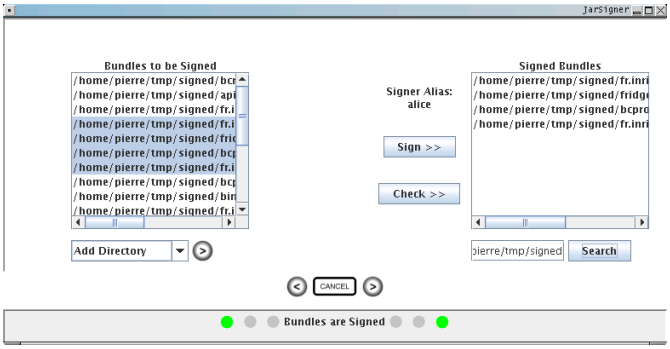

Figure 3. The Interface of the SF-Jarsigner Tool

The SF-Jarsigner tool is provided as OSGi bundles. It can be executed over the Felix, SFelix and Equinox platforms. With a minor configuration effort, it can also be executed over Knopflerfish. It uses several third party open source libraries: the Bouncycastle cryptographic libraries bcprov and bcmail, the XML libraries xml-commons-resolver and Xerxes from Apache, the FTP library Edtftp, and the OBR2 metadata extractor Bindex. The validation functionalities are provided by the 'jarvalidation' bundle, and the graphical interface by the 'Jar Signer Gui' bundle. Its total size is 3,657 kbytes. SF-Jarsigner is available through the SFelix Bundle Repository[4].

3.3 SFelix: a hardened OSGi platform

The SFelix platform is an extension of the Felix OSGi implementation, which performs a bundle signature check before the bundles are installed. If the signature of a given bundle is unvalid, the bundle is rejected and the installation is aborted.

The SFelix security layer is provided as a Java library, and therefore not seen as actual bundles in the framework. The verification is performed as a single call to the 'jarvalidation' library we provide, and which is the same that the bundle used in the SF-Jarsigner tool. The required libraries are the Bouncycastle libraries bcprov and bcmail. The total size of the Security Layer is 1.932 kbytes, because of the numerous cryptographic primitives that are embedded in the Bouncycastle library. In an environment with restricted resources, it would be necessary to extract the required classes from these library, so as not to overwhelm the available memory with unused code.

[4]http://www.sfelix.free.fr/repository/repository-ppd.xml

4 Comparison with other Archive Validation Processes

So as to confirm the usefulness of the tools we propose, we compare them to the solutions that are currently available. Two aspects need to be considered: the signature step, and the validation step.

4.1 Signing Archives

Two tools are available for signing Jar files or bundles. The first is the Sun Jarsigner, with is part of the Java SDK. The second is the SF-Jarsigner tool we provide. The Sun Jarsigner is available as a command line utility. The SF-Jarsigner is provided as a set of OSGi bundles, which support both a convenient use through a graphical user interface, or a programmatic use through services published inside the OSGi framework.

The specifications of digital signature according to the Jar file specifications [15] are relatively vague. They are therefore completed in the OSGi R4 specifications [10]. However, if the validity criteria for archive signature are more strict in the frame of OSGi systems (see section 4.2), the default signature process by the Sun Jarsigner generates OSGi compatible archives.

The main difference between the Sun JVM and the OSGi specifications is that, in the case of the Sun signature, all entries of the manifest file must be hashed and stored in the so-called 'Signature File' (see [12] for the detailed structure of a signed bundle). This is not required by the OSGi specifications, which consider that no new resource can be added to a signed archive. However, to provide compatibility with Sun tools, the SF-Jarsigner adds these additional meta-data.

Consequently, it is possible to use any of those tools for performing the digital signature of OSGi bundles - or of other Java archives. The benefit of SF-Jarsigner lies 1) in the convenient user interface and 2) in the support for bundle publication.

4.2 Signature Validation

Whereas the actual behavior of the considered signing tools is identical, the criteria for verifying the validity of signed archive greatly varies between the various available checkers and platforms. This has the direct consequence that an OSGi bundle that has been modified will not be considered as unvalid in all tools but ours.

Table 1. Behavior of several tools and frameworks in the presence of unvalid archives

Error Type	Sun Jarsigner	Java with Security Manager	Felix	SFelix
Unsigned Archive	W	A	R	R
Unknown Signer	A	A	R	R
Addition of Resource	A	A	A	R
Removal of Resource	A	A	A	R
Modification of Resource	R	R	W	R
Unvalid Order of Resources	A	A	A	R
Signature of Embedded Archive Unvalid	R	R	W	R
Time Of Check	Test	Exec	Exec	Install

A: Accept; R: Reject; W: Warning;

The criteria of validation of the signature of a bundle are shown in the table 4.2. All potential errors are listed according to the OSGi R4 specification [10], and the behavior of the tested tools and platforms in the presence of such an error is given. The considered tools and platforms are the following: the Sun Jarsigner, the Sun Java virtual machine with a Security Manager, the Felix platform with security enabled, and the SFelix platform. All data are directly drawn from experience, but the Felix behavior. Since no minimal permission policy is made available, and can not be easily deduced from the behavior of the platform, it can be assumed that 1) the certificate control, which is done explicitly in the platform, is performed correctly, and 2) that the integrity control, which is done by the virtual machine, is done in the same way than the 'Java with Security Manager' case. The behavior of the SF-Jarsigner tool, which can also be used to check signatures, is the same as the one of SFelix, since it relies on the same 'JarValidation' library. The other open source implementations of OSGi are not considered here. Knopflerfish[5] does not provide support for bundle signature. Neither does Equinox[6]. The integrity control during the deployment of Eclipse plugins, which is based on the Equinox framework, is performed during the Eclipse plugin deployment through a specific mechanism.

[5]http://www.knopflerfish.org/

[6]http://www.eclipse.org/equinox/

The results of the experiments are the following. The Sun Jarsigner tool identifies files that are not signed, that have been tampered with, or for which the public key certificate is outdated. It does not take into account the fact that the signer is trusted or not, though it has access to the 'keystore' which contains such information. The Java virtual machine has the same behavior than the Jarsigner, but does not take the warnings into accounts. Consequently, no difference is done between a signed and an unsigned archive if the valid signer are not explicitly indicated in the permission policy before the virtual machine is launched. Therefore, no modification of the list of trusted signers can be done at runtime, which can be restrictive. The Felix framework is expected to reject the unsigned archives and the archives with unvalid signers. It issues warnings when archives that have been tampered with are installed, but seems to install them anyhow. SFelix has been developed specifically to be conform to OSGi R4, so it has a proper behavior in all tested error cases. Its current limitation is that it does not handle certificates chains, used when a valid signer delegates its right to another entity.

The time of integrity control is different in each tool. The Sun Jarsigner tool, as a command line utility, prints the test results immediately. The Java virtual machine checks the integrity of the files when they are loaded to be executed, and so does the Felix platform. This is not consistent with the requirements that the whole archive is sound which is expressed by the OSGi specifications. This explains why Felix is not R4 compliant, at least what concerns the control of the integrity of the bundles. The SFelix platform performs the archive signature check at install time. This implies a slight performance overhead, but is necessary to guarantee that only valid archives are installed. This approach prevents the sudden unavailability of services that are installed, but for which some unfrequently executed classes are tampered with.

5 Conclusions and Perspectives

We present in this paper the tool suite we developed to support the secure deployment of OSGi bundles. Our contribution is twofold. First, we integrate the available publication mechanisms defined by the Bundle Repository format in a convenient tool that makes possible to publish a set of bundles so that they are made available to the client platforms. Secondly, we provide a library that support the generation and the validation of signature according to the OSGi R4 specifications. The signature generation functionality is integrated in the publication tool SF-Jarsigner. The signature validation functionality is integrated as an extension of the Felix OSGi implementation, name SFelix (for Secure Felix). No other tool for bundle publication is currently available, to the best of our knowledge. Moreover,

all other existing validation mechanisms for OSGi bundle signature do not follow the OSGi specifications. To provide firm arguments of the benefit of our tools, we performed a comparison between the various archive signature validation mechanisms.

The limitations of our tools are the following. Related to the publication phase, only the OBR2 protocol is supported. However, Knopflerfish, and Oscar (the predecessor of Felix) still use the OBR1 format, which is not compatible with it. The extension of the current facility would make our tool useful in systems that use those platforms, in particular Knopflerfish. Related to the signature process, the main current limitation is that the possibility of signature delegation through signature path is not supported at the verifier's, which must have a reference to each actual signers. Moreover, the signature generation must be done to maintain the compatibility with the Java virtual machine and Sun tools. Therefore additional meta-data are inserted in the hash files of the signed bundles that would not be necessary.

This work provides the community with convenient tools for securely publishing OSGi bundles, or other software components. It also provides an implementation of the OSGi security layer which is necessary for comparing the current specifications and a modified security architecture. The next requirement is the development of an infrastructure to manage the identity of the bundle issuers. The current proposed solution is based on Public Key Infrastructure, which proves to be difficult to put into use. We therefore plan to study the possibility of using an alternative signature scheme. The Identity-based signature mechanism, for instance, could be a valuable alternative to provide a comparable level of security, while greatly simplifying the key management process.

References

[1] A. Carzaniga, A. Fuggetta, R. S. Hall, A. van der Hoek, D. Heimbigner, and A. L. Wolf. Characterization framework for software deployment technologies. Technical Report CU-CS-857-98, Dept. of Computer Science, University of Colorado, 1998.

[2] A. Dearle, G. Kirby, A. McCarthy, and J. C. D. y Carballo. A flexible and secure deployment framework for distributed applications. In *Second International Working Conference on Component Deployment*, number 3083 in LNCS. 2004.

[3] S. Frenot and Y. Royon. Component deployment using a peer-to-peer overlay. In *Third International Working Conference on Component Deployment, Grenoble, France*, volume 3798 of *LNCS*, November 2005.

[4] M. Gaedke, J. Meinecke, and M. Nussbaumer. Supporting secure deployment of portal components. In *Internation Conference on Web Engineering*, number 3140 in LNCS, 2004.

[5] M. Grechanik and D. E. Perry. Secure deployment of components. In *Second International Working Conference on Component Deployment*, number 3083 in LNCS, 2004.

[6] R. Hall, D. Heimbigner, A. van der Hoek, and A. Wolf. An architecture for post-development configuration management in a wide-area network. In *International Confeence on Distributed Computing Systems*, May 1997.

[7] D. Heimbigner, R. Hall, and A. Wolf. A framework for analyzing configurations of deployable software systems. In *Proc. of the 5 IEEE Int'l Conf. on Engineering of Complex Computer Systems*, October 1999.

[8] R. Housley. Cryptographic message syntax (cms). IETF RFC 3852, July 2004.

[9] Y.-G. Kim, C.-J. Moon, D.-H. Park, and D.-K. Baik. A mac-based service bundle authentication mechanism in the osgi service platform. In *Database Systems for Advanced Applications*, volume 2973 of *LNCS*, 2004.

[10] OSGi Alliance. Osgi service platform, core specification release 4. Draft, July 2005.

[11] OSGi Alliance and R. S. Hall. Bundle repository. OSGi Alliance RFC 112, 2006.

[12] P. Parrend and S. Frenot. Secure component deployment in the osgi(tm) release 4 platform. Technical Report RT-0323, INRIA, June 2006.

[13] P. Parrend and S. Frenot. A security analysis for home gateway architectures. In *International Conference on Cryptography, Coding & Information Security, Venice, Italy*, November 2006.

[14] K. Pauls and R. S. Hall. Eureka - a resource discovery service for component deployment. In *Second International Working Conference on Component Deployment*, volume 3083 of *LNCS*, pages 159–174, 2004.

[15] Sun Microsystems, Inc. Jar file specification. Sun Java Specifications, 2003.

[16] A. van der Hoek and A. L. Wolf. Software release management for component-based software. *Software: Practice and Experience*, 33:77–98, 2003.

[17] C. Wang, A. Carzaniga, D. Evans, and A. L. Wolf. Security issues and requirements for internet-scale publish-subscribe systems. In *Hawaii International Conference on System Sciences*, 2002.