Security of the OSGi platform *

Anton Philippov, Olga Gadyatskaya, and Fabio Massacci

DISI, University of Trento, Italy
{name.surname} @unitn.it

Abstract. In the last few years we have seen how increasing computational power of electronic devices triggers the functionality growth of the software that runs on them. The natural consequence is that modern software is no longer single-pieced, it becomes, instead, the composition of autonomous components that run on the shared platform. The examples of such platforms are web browsers (such as Google Chrome), smartphone and smart card operating systems (e.g., Android and Java Card), intelligent vehicle systems or smart homes (usually implemented on OSGi). On one hand, these platforms protect components by isolation, but at the same time, provide methods to share and exchange services. If the components can come from different stakeholders, how do we make sure that one's services would only be invoked by one's authorized siblings? In this PhD proposal we illustrate the problems on the example of OSGi platform. We propose to use the security-by-contract methodology (S×C) for loading time security verification to separate the security from the business logic while controlling access to applications.

1 Introduction

With the ever-increasing popularity of smart devices and rapid development of Internet, the single application model is more and more often being replaced with the service platforms. These include Java Card platform for multi-applicational smart cards, Android for smartphones, OSGi for component-based Java applications and smart homes, Google Chrome platform for browser plugins. In general, we say that *service platform* is a platform, on which applications are isolated, but can share selected parts of their functionality with other applications on the platform. Furthermore, such shared functionality we will call a *service*. In the current proposal we will concentrate on Java-based service platforms and, in particular, OSGi platform.

The Open Services Gateway Initiative (OSGi) framework [1] is one of the most flexible solutions for the deployment of pervasive services in home, office, or automobile environments [5, 7, 9]. The OSGi services are also the basic building blocks for service mash-ups extending the classical "smart homes" scenarios to richer settings [8]. In a nutshell, the OSGi framework redefines the modular system of Java by introducing *bundles*: JAR files enhanced with specific

* This work is partially supported by EU-funded project FP7-257930 ANIKETOS and EU-funded project FP7- 256980 NESSOS.

metadata. The *services* layer connects bundles in a dynamic way with a publish-find-bind model for Java objects. As a result, an OSGi platform is expected to be highly dynamic. All pervasive and mash-up applications expect that bundles can be installed, updated or removed at any time. From a security perspective, the possibility of bundle interactions is a threat for bundle owners. Since bundles can contain sensitive data or activate sensitive operations (such as locking doors and windows of somebody's house), it is important to ensure that the security policy of each bundle owner is respected by other bundles. However, such aspects have been only partially investigated.

How do we make sure that one's services are invoked by one's authorized siblings? A simple solution is to rely on service-to-service authentication to identify the services and then interleave functional and security logic into bundles, for example, by using aspect-oriented programming [9]. However, this decreases the benefits of common platform for service deployment and significantly hinders evolution and dynamicity: any change to the security policy would require redeployment of the bundle (even if its functionalities are unchanged). Vice versa, any changes in the bundle's code would require redeployment of security as well.

Our solution is to use the security-by-contract methodology (S×C) [2, 3] for loading time security verification in order to separate security and the business logic while achieving a sufficient protection of applications among themselves.

In the next section we illustrate the problem by introducing a concrete case study for home gateways (§2) and discuss the security issues that the plain OSGi model cannot solve without ad-hoc security codes within each bundle. We then introduce the solution (§3) and conclude in §4 with an overview of the paper.

2 Problem statement

Further we concentrate on OSGi platform, identify the flows of its security mechanisms. However, due to the similarities in the architecture, the problems can be applied to other service platforms (e.g., Android) as well. Due to the paper length constraints we skip the technical description of the OSGi platform and assume the reader has at least a basic familiarity with the service platform architecture.

The Scenario We consider as a case study an OSGi platform deployed as a service gateway in a smart home. Let us consider Alice, the smart home resident, and a telecom provider, the owner of the platform. Alice can download bundles for entertainment (news RSS feeds, media bundles from TV providers) or even bundles with traditional Internet content (like Facebook or Twitter), as nowadays new TV sets can be used for all these purposes. The interested reader can refer to [6] for more details on the news feed scenario. In this example we have used fictional names, but they give an idea of realistic bundle interactions and possible policies regarding these interactions.

Alice, a beginner stock market player, downloads and installs bundle A from provider $FSM.com$ that can provide her with an interface of the stock market

operations. This bundle includes service S_A that retrieves updates about the stock prices. However, Alice later finds and installs another stock market bundle B from $BH.fr$ provider, that also provides service for prices information retrieval S_B and service S_{fr} that allows Alice to transfer money from her stock market account (registered on $BH.fr$) to her Happy Farm account on $FB.com$. Thus Alice also installs Happy Farm bundle F.

The bundle providers want to ensure that their security policies related to bundles and services usage are enforced on the Alice's platform. Their requirements are as follows:
FSM.com: *Access to S_A service is allowed only for bundles signed by $FSM.com$.*
BH.fr: *Access to S_B service is allowed only for bundles signed by $BH.fr$. Only bundles signed by $BH.fr$ can import the package containing S_B. Access to S_{fr} service can be granted only for bundles signed by $FB.com$ or by $BH.fr$.*

The OSGi platform at Alice's smart home has to ensure that the requirements of each provider are respected. We will next discuss how the OSGi platform can enforce these requirements and why this approach is not satisfactory. We will also demonstrate that there can exist similar requirements of bundle providers that cannot be enforced by the OSGi platform at all.

Security Challenges Let us first briefly present the relevant OSGi platform details [1]. An OSGi bundle is a JAR file that includes the manifest.mf file containing the necessary OSGi metadata including dependencies and the provided libraries. Dependencies are expressed as *requirements* on *capabilities*. Capabilities are attribute sets in a specific namespace and requirements are filter expressions that assert the attributes of the capabilities. A requirement is satisfied when there is at least one capability that matches the filter. Bundles can interact through two complementary mechanisms: the export/import of packages and the service registration/lookup facility. A *service* is a normal Java object registered under a Java interface with the *service registry*. Each bundle is associated with a set of permissions, that are queried at runtime. The OSGi specification defines ServicePermission, BundlePermission and PackagePermission, which are used for getting/registering a service, importing/exporting bundles and packages respectively. The platform can authenticate code by download location or by signer (digital signature). The Conditional Permission Admin service manages the permissions based on a comprehensive conditional model.

We assume the framework can host multiple third-party bundles, and these bundles can freely register services. The goal of the telecom provider running the platform is to make sure that there are no undesired security or functionality problems among different bundles installed by the end user (who most likely does not even know what is a bundle and just sees the web interfaces of the services). Thus a threat scenario under investigation is a case when a bundle gains unauthorized access to the sensitive data of another bundle (security threat), or a bundle is malfunctioning due to unavailability of a certain service (functionality threat). We now discuss these threats separately in the light our scenario.

A confidentiality attack can be realized by the bundle A of provider $FSM.com$ getting access to the sensitive stock market prices service S_B of provider $BH.fr$.

This might happen if A imports the package containing the service S_B definition, requires the bundle B (thus importing all its exported packages), or tries to get a reference to this service from the Service Registry and then get access to the object referenced.

We now discuss how the current OSGi security management can address this security threat. Import of a package or a require-bundle action can be granted if the requiring bundle has corresponding permissions. Simple reviewing of the manifest file and permissions file of the bundle A can report about a (potential) attempt to interact with the bundle B. However, there is no convenient and simple way for the owner of the bundle B, the $BH.fr$ provider, to declare which other bundles are allowed to import its packages.

Package importing can be guarded by the permissions mechanism, as we discussed before. Currently only the platform owner (the telecom provider) can define and manage policies in the Conditional Permission Admin policy file. The $BH.fr$ provider might contact the telecom provider to ask him to set the required permissions, or its bundle B, being granted the necessary permissions, can add new permissions to the Conditional Permission Admin policy file. These approaches are organizationally cumbersome and costly, as they require the operator to push the changes to its customers before any downloads of $BH.fr$ bundles, even the customers have no intention of using them.

Service usage is another, more trickier issue. Again, the necessary authorizations for the service usage (more precisely, GET permissions for service retrieval) can be delivered within bundle contracts and incorporated into the policy file of the system. But the invocations of the methods within a service, once the necessary reference is obtained, are not guarded by the permission check, and usually the security checks are placed directly within the service code, thus mixing the security logic with the execution logic.

Another solution, that is traditional for mobile Java-based component systems, could be to ask Alice each time a specific permission is needed. But Alice is not the owner of the bundles to make such decisions, nor is she interested to do so. Let us consider a more complex scenario now.

Example 1 *Alice wants to install the Sims add-on from the EA.com provider. This add-on is packaged into the bundle C and it will provide an integration of the Happy Farm account with her the Sims account. The functional requirement of the EA.com provider is the following: "The bundle C can be installed if and only if the F bundle is available on the platform and provides the Happy Farm service S_F.*

The requirement in Example 1 means that bundle C can be installed only if the service S_F is already provided on the platform. This requirement prevents the denial of service by the Sims bundle. The bundles are running on top of a single JVM, thus the denial of service attack can cause a restart of the whole system [4]. This functional requirement is, in fact, unsupported by the current OSGi specification. Requirements/capabilities model cannot provide guarantees on the provided services (except that their definition exists on the platform).

3 Proposed solution

Our proposed solution is to adopt the Security-by-Contract that was initially investigated and implemented by Bielova et al. for mobile Java-based devices [2] and by Dragoni et al. for the Java Card platforms [3]. Further we provide details on possible architecture of S×C for OSGi.

The S×C framework consists of two main components: the ClaimChecker and the PolicyChecker. The verification workflow is described on Figure 1.

Informally, the S×C process starts when a new bundle B is loaded. The ClaimChecker component then accesses the manifest file, retrieves the information about imported and exported packages and obtains the bundle contract. Then the ClaimChecker reads the permissions.perm file, which contains local bundle permissions, extracts permissions requested by the bundle B and related to services retrieval, packages importing, requirements of bundles, etc., and combines this information into the overall "security claims and needs" of the bundle. Having these claims, the ClaimChecker then analyzes the bytecode of the bundle to verify that the claims match actual code. If the verification fails, meaning that the claims are not supported by the code, the bundle is removed from the platform. Otherwise, the PolicyChecker component receives the result from the ClaimChecker and matches it with the security policy of the platform, that aggregates the security policies of all the installed bundles, and with the functional state of the platform (installed bundles, running services, etc.). If the PolicyChecker failed on either of the checks, the bundle is removed from the platform. Otherwise, it is installed and the security policy of the platform is updated by including the security requirements of B.

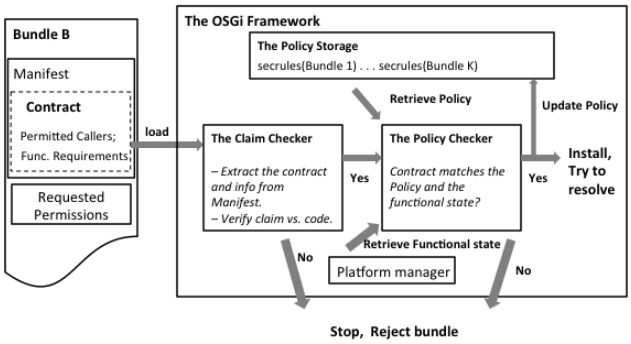

Fig. 1. SxC Workflow

In terms of technical realization, the S×C steps can be integrated with the OSGi framework. The key requirement is getting the correct and up-to date information about the state of the platform and being able to access the received bundle *before* it is deployed on the platform. The S×C framework itself can be

a bundle, provided it can access the service registry, the framework policy file, the lifecycle layer and the manifests of the bundles.

4 Conclusions

In this proposal we have identified the security problems of some of the Java-based service platforms on the example of OSGi platform. We have presented and idea of solution, which is a Security-by-Contract paradigm for the OSGi platform. We discussed the security and functionality challenges and proposed how to enable the bundle providers with ability to effectively express their security and functional requirements on the platform.

The main benefits that the S×C approach can bring to service platforms are the following. From the security aspect, the bundle providers can now specify the authorizations for access to their bundles, packages and services. The policies can be updated easily and the update does not require an interaction from the platform owner, an access to the framework policy file or an update of the execution logic of the bundle. For the functionality aspect, the bundle providers have now a more powerful tool for expressing their functional requirements than the requirement/capability model of OSGi. The contracts can express requirements on the current state of the platform (including requirements on the states of the bundles or certain services provision, or absence of the competitor's resources).

References

1. T. O. Alliance. OSGi service platform core specification. Version 4.3, 2011.
2. N. Bielova, N. Dragoni, F. Massacci, K. Naliuka, and I. Siahaan. Matching in security-by-contract for mobile code. *Journal of Logic and Algebraic Programming*, 78(5):340 – 358, 2009.
3. N. Dragoni, O. Gadyatskaya, and F. Massacci. Can we support applications evolution in multi-application smart cards by Security-by-Contract? In *WISTP-2010*, LNCS 6033, pages 221–228.
4. N. Geoffray, G. Thomas, G. Muller, P. Parrend, S. Frénot, and B. Folliot. I-JVM: a Java Virtual Machine for Component Isolation in OSGi. In *DSN'2009*. IEEE.
5. T. Gu, H. Pung, and D. Zhang. Toward an OSGi-based infrastructure for context-aware applications. *IEEE Perv. Computing*, 3:66–74, 2004.
6. F. Innerhofer-Oberperfler, S. Löw, R. Breu, M. Breu, M. Hafner, B. Agreiter, M. Felderer, P. Kalb, R. Scandariato, and B. Solhaug. D2.2: A configuration management process for lifelong adaptable systems. Public deliverable of the Secure Change project, 2011.
7. C. Lee, D. Nordstedt, and S. Helal. Enabling smart spaces with OSGi. *IEEE Perv. Computing*, 2(3):89 – 94, 2003.
8. A. Ngu, M. Carlson, Q. Sheng, and H. Paik. Semantic-based mashup of composite applications. *IEEE Tran. on Services Computing*, 99:2–15, 2010.
9. P. Phung and D. Sands. Security policy enforcement in the OSGi framework using aspect-oriented programming. In *COMPSAC'2008*, pages 1076–1082.