Security for Downloadable Automotive Services

Stephan Merk*, Kathrin Scheidemann*, Michael Rudorfer*, Thomas Stauner*,
Johannes Grünbauer**, Gerhard Popp**, Guido Wimmel**

*BMW Car IT GmbH, Petuelring 116, 80809 München, Germany
[stephan.merk,kathrin.scheidemann,michael.rudorfer,thomas.stauner]@bmw-carit.de

**Institut für Informatik, TU München, Boltzmannstraße 3, 84748 Garching b. München, Germany
[popp,wimmel,gruenbau]@in.tum.de

Abstract. The paper explains how a security analysis can be conducted for future adaptive, service-based automotive software systems. The central idea is to use abstract threat trees which are instantiated and worked out in detail when a concrete implementation platform is known. We then consider the main threats resulting from the analysis in the context of the candidate implementation platform OSGi/Java and identify two main problems. For one of the problems a solution is sketched.

1 Introduction

The increasing integration of electronic devices and networks into daily life leads to a growing availability of software based services. Services can autonomously react to events within the system, adapt their behavior to their environment and are even able to alter the whole system: adaptiveness can change the whole structure of a system by exchanging certain components. Services increasingly find their way into new domains such as the automotive world and in such mission critical environments security plays an extremely important role.

In the project MEWADIS [Mew04], we try to find methods and techniques for the development of secure, dynamic, downloadable and context adaptive services. These methods comprise a formal service definition, a formal and automatic composition of services, the ability to verify security requirements and a methodical development process. As a practical application we consider mobile services in cars together with their appropriate user interface (man-machine-interface, MMI). This field is a proper choice as matters of security, extensibility and adaptiveness are concerned.

Dynamically composed, downloadable services require a great amount of security, safety and reliability. These goals cannot be achieved without systematic support from the development process.

A profound knowledge of the threat situation is vital to define security requirements.

The situation in the automotive environment is characterized by high security needs, a long product lifecycle, and difficult processes to perform software updates, if available at all. This special situation requires a tailored procedure for the threat and risk analysis. We start from abstract threats and refine them according to the concrete distribution of the components on the physical devices in the car. This approach is described in Section 2.

The security requirements we derived from the threat analysis have to be addressed by adequate countermeasures provided by a concrete technical environment for services. The OSGi platform is a candidate for such a secure environment and shall be reviewed with regard to security mechanisms. Where OSGi provides no sufficient security mechanisms, it is necessary to define additional ones to fill these gaps and supply the infrastructure for secure, downloadable services. These concepts are explained in Section 3. Finally, in Section 4 we give a short conclusion.

2 Security Threats for Downloadable, Adaptive Automotive Services

Automotive systems in general are very heterogeneous distributed systems, characterized by multiple interactions, strong interrelations and dependencies between functional units.

In [DGP+04a], the authors have introduced a model-based and incremental service–based development process which could be applied to cope with the complexity of such systems. Services, i.e. functional entities with their encapsulation, dependencies and combination are used as abstract modelling units and are finally mapped to components, which can, but do not necessarily need to run within service orientated architectures like OSGi, Openwings or Jini. The service specification includes a syntactic service interface, a behavior, a set of properties and dedicated relationships to other services [DGP+04b].

A service is called adaptive, if its behavior is dependent on its context, where the context comprises attributes of the service's environment. An example of a context adaptive service would be an on-board diagnosis service which is only allowed to carry out some functional tests while the vehicle is parked.

Method of Analysis. In order to derive security needs in a system hosting downloadable, adaptive services, we developed a service-based model for an automotive system based on current and new functionalities and carried out a detailed security analysis.

The multiple interactions of functional units mentioned above also have a strong impact on security aspects. The analysis procedure therefore must take into account these aspects. The applied security analysis method is based on procedures introduced in [BSI03] and [How03] and was adapted to the special circumstances of service-based modeling and of the automotive domain.

Starting from the specification of the services, a catalog of 81 basic threats (potential events causing damage) was developed. Threats were recorded per service in 5 different categories: malfunction of the service, unavailability, confidentiality of data, integrity of data and non-repudiation of service interactions. For each basic threat, values were determined for the damage potential and the estimated motivation of potential attackers.

To keep damage potential constant when services are recombined, damage resulting from effects on related services was not taken into account at this stage, but will be considered in the next step of the analysis.

The values of the damage potential and the estimated motivation of potential attackers were used to determine critical threats that ought to be examined more closely.

The aim of the next analysis step was to list as many attack scenarios as possible leading to the realization of one of the critical threats. A well known structure for representing attack scenarios is called 'threat trees' (see e.g. [And01]). Similarly to fault trees, they model an undesired event (a threat) in their roots, and successive nodes represent possible causes.

Threat tree modeling in our automotive context was subdivided into two phases. Within the first phase only abstract attack scenarios were taken into account, leaving aside all platform and deployment specific details. The set of abstract threats that had to be taken into consideration as potential child nodes proved to be limited to a few typical abstract attack patterns. Therefore, a set of template trees was developed which could be used as a systematic support for the first phase of the threat tree modeling. For each of the critical threats determined above, we started tree modeling using one of the templates. Templates contain sub trees that describe attack scenarios which directly affect the service, e.g. tempering with the code of the service implementation. There are also sub trees in the templates which describe attack scenarios involving related services. If the service has relations to more than one other service, the corresponding sub tree in the template has to be instantiated respectively. Some of the sub trees may not be relevant on account of the specification of the service. Therefore these sub trees are pruned.

Leaf nodes in the tree can sometimes be seen as threats to related services that are also part of an attack scenario that leads to the threat modeled in the current threat tree. In that case, the threat tree which models the threat to the related service can be inserted as a sub tree. We call that *threat tree composition*.

In the second phase the abstract trees were elaborated, according to one concrete deployment scenario. Therefore the leaves of the abstract trees had to be substantiated by platform specific details, now allowing for an estimation of the likelihood of certain attack scenarios, derived from attack scenario characteristics like necessary resources, technical knowledge or its exploitability. Figure 2.1 illustrates our threat tree modeling procedure.

The advantage of this two step procedure is that changing the deployment of the services doesn't affect the abstract trees, but only the concrete refined ones. Therefore it would be possible to optimize system deployment according to security aspects without having to remodel the trees from scratch.

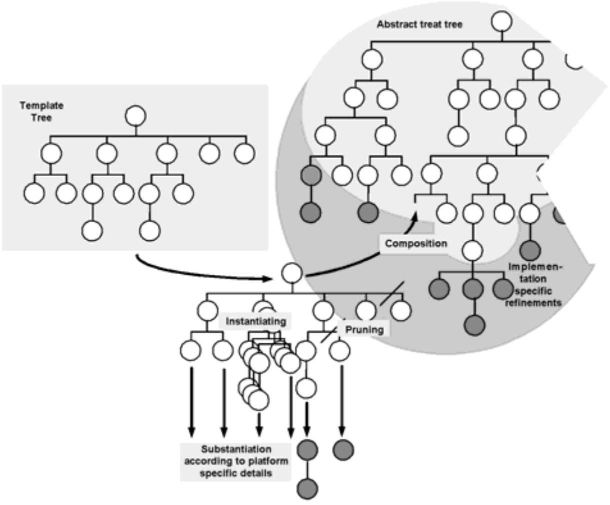
Abstract treat tree
Template Tree
Composition
Implemen-tation specific refinements
Instantiating
Pruning
Substantiation according to platform specific details

Figure 2.1.: Two stage threat tree modeling.

One of the threats we found was that an attacker could try to manipulate displays in the car in order to deflect the driver. If such an attack was possible, damage potential is estimated to be high , since it could lead to safety risks or negative publicity. We assumed the motivation of potential attackers to be medium high, because no financial benefit could be expected. The likelihood of an attack to be successful was estimated to be high in an unsecured environment, in case non-trustworthy code had full access to the displays. This has to be prevented by appropriate system design.

Results. In our study, we were able to map the adaptiveness of services to well known structures, more precisely to normal service relations: context information was assumed to be provided by other services. This way our method supports the analysis of adaptive services as well.

The most significant risks in the system are associated with the interaction between safety critical vehicle functions and possibly not trustworthy downloadable services. It is also necessary to maintain a degree of isolation for downloaded services from different vendors, to protect them against each other. Therefore, a service-platform hosting downloaded applications must provide mechanisms to grant fine grained access permissions to services according to the authenticated source of code. A platform hosting both, safety critical and downloadable services, must also provide mechanisms to ensure that possibly not trustworthy services can not illegally influence safety critical functional units, e.g. by occupying necessary resources.

The OSGi platform, which was specified by the Open Services Gateway Initiative, a non-profit cooperation of industrial members including IBM, Sun and the BMW Group, is one possible platform that could be used to host downloaded services within next generation cars, as it provides mechanisms to enable the delivery and execution of managed services.

In the next section, we give a brief introduction to the OSGi Platform itself, with emphasis to its major security mechanisms and its ability to meet the security demands derived from our analysis.

3 Security Support in the OSGi Architecture

OSGi-Overview. The core OSGi specification [OSGi03] defines a Java-based framework with a minimal component model, management facilities for the components and a service registry. The two central concepts of OSGi are *services* and *bundles*. A service is described by an interface and has a predefined behavior. A bundle can be regarded as a physical unit of deployment and as a logical component within the framework. In the first case it is essentially a JAR file with a manifest, Java class files and eventually other resources. Among the classes in the JAR are the service-interfaces, its implementations and a special class, called *bundle activator* that serves the framework as a starting point for the execution of the bundle. From a conceptual view, a bundle is a dynamic life cycled component that can be installed started and stopped by the framework. Once started, a bundle can register services with the registry by providing the name of the service interface, the service implementation and optional properties. Other bundles can get available services via the registry and use them. As there is a strict separation between the specification of a service by its interface and its implementation, a OSGi service could have multiple implementations provided by different bundles.

Furthermore, a bundle can import and export Java packages by declaring the names of the packages in IMPORT/EXPORT headers of the manifest. In a process, called bundle resolution, the framework automatically maps each imported package to an exported package, provided by exactly one other bundle. Thus all bundles importing a certain package use the same implementation of the package. The import/export mechanism is necessary in order to get references to service interfaces and especially imposed by the classloading concepts of Java and OSGi. Besides it allows for the sharing of libraries between bundles.

Evaluation of the OSGi Security Model. Starting from the security analysis we evaluated the security mechanisms of OSGi. We assumed a simple onboard architecture with an OSGi-framework running on an ECU, that is connected to the CAN (Controller Area Network) and MOST (Media Orientated Systems Transport) busses. Inside the framework there are several core bundles, providing high-level services to access vehicle functions, and a set of bundles downloaded from an offboard server, that offers bundles from different providers. Based on the trustability of their provider, we enhanced the OSGi security mechanisms by associating bundles with trust levels (TL). Depending on their trust level downloaded bundles get restricted access to vehicle functions via the core services. Additionally they can register services that can be accessed by other downloaded bundles. To be able to enforce protection of downloaded services against each other, we set up a generic service access policy, saying that bundles in general are only allowed to use service implementations that are provided by bundles having a trust level which is at least as high as their own.

OSGi security, as defined by the specification, is based on the Java security model. It focuses on access control to critical operations, like e.g. getting a service, and requires the calling bundle to have a special permission for the action. Each bundle is granted a single set of permissions that are configured via the *PermissionAdminService*. Permissions are bound to a so called *bundle-location* identifier. OSGi does not permit to grant permissions depending on the signer of the bundle JAR file.

The security part of the specification defines three different security permissions, namely *AdminPermission*, *ServicePermission* and *PackagePermission.*

AdminPermission is required for highly critical tasks like managing the lifecycle of a bundle or configuring the permissions of bundles and should be restricted to trusted management bundles.

ServicePermission (*REGISTER* resp. *GET*) grants the authority to a bundle to register a service with the registry or to get a given service that is implemented by another bundle. ServicePermissions are defined with respect to the fully qualified name of the service interface to be accessed. Implementation specific properties can not be taken into account. As mentioned above, the same service can be implemented by two bundles with different trust levels. To enforce our service access policy in this context, we need to grant trust level dependent permissions, which are not supported by the OSGi permission concept. To cope with this problem an interceptor concept can be used.

PackagePermission (*EXPORT* resp. *IMPORT*) allows the export and import of packages, that are implemented by bundles. The concept of sharing packages by exporting and importing them violates the separation of namespaces of different bundles. By exporting a malicious or incompatible package, one bundle can start a spoofing or compatibility attack against other bundles, which import the package. Forbidding the export and import completely, that is not granting *PackagePermissions* to user-defined bundles, does not solve the problem, as the ability to export and import the packages with the service interfaces is an absolute need for the registration and lookup of services. The import/export problem is essentially a consequence of the automatic package resolving by the framework. As the specification doesn't provide an interface to alter the package resolution process, there is actually no OSGi-compliant solution to this security-critical problem.

Attacks against the availability of the onboard platform and its connected devices are important threats in the context of a resource constrained environment. OSGi doesn't provide means to counter these threats.

4 Conclusion

We expect that future extendibility of automotive systems by software download will be based on context-adaptive services running on a central platform. To provide an absolutely safe operation of a vehicle it is vital to analyze security of the service download and provide countermeasures against possible threats. This paper therefore presented some results from the MEWADIS project.

First, it outlines how a detailed security analysis can be conducted for downloadable adaptive services in a car. The main result here is that a number of abstract attack patterns can be identified and abstract threat trees can be defined for them. Once the concrete deployment platform is known, a concrete threat tree can then be elaborated by instantiating and detailing the abstract trees. Concerning the analysis itself, the central result is that a service platform for downloadable services has to provide access permissions depending on the authenticated source of the code and measures to ensure that non trustworthy services do not influence other services on the platform.

The second part of the paper then considers how the candidate platform architecture OSGi supports these security requirements. For the requirement of trust level based access to services, we propose to use interceptors to enforce this extension of the OSGi access policy. A further problem is that the concept of sharing of packages in OSGi can be used to disrupt services using a package. However, package sharing is a prerequisite for a service-centered architecture. From our analysis it seems that this problem can only be alleviated but not completely solved.

5 Bibliography

[DGP+04a] Martin Deubler, Johannes Grünbauer, Gerhard Popp, Guido Wimmel & Christian Salzmann “Towards a Model-Based and Incremental Development Process for Service-Based Systems”, Proceedings of the IASTED International Conference on Software Engineering, 2004

[DGP+04b] Martin Deubler, Johannes Grünbauer, Gerhard Popp, Guido Wimmel & Christian Salzmann “Tool Supported Development of Service-Based Systems”, submitted.

[Mew04] MEWADIS Web site, available via http://www4.in.tum.de/~mewadis/ (in German).

[BSI03] Bundesamt für Sicherheit in der Informationstechnik "IT-Grundschutzhandbuch“, 2003

[And01] Ross Anderson „Security Engineering“, John Whiley & Sons, Inc. 2001

[How03] Michael Howard & David LeBlanc „Writing Secure Code“, Microsoft Press, 2003

[OSGi03] The OSGiAlliance “OSGi Service Platform”, IOS Press, 2003

[HaCe04] Richard S. Hall & Humberto Cervantes “An OSGi Implementation and Experience Report”, CCNC, 2004