Risk-based Testing of Bluetooth Functionality in an Automotive Environment

Felix Jakob[1], Wolfgang Kremer[1], Andreas Schulze[1,]
Jürgen Großmann[2], Nadja Menz[2], Martin Schneider[2],
Alain-Georges Vouffo Feudjio[2]

[1]Dornier Consulting, Sindelfingen, Germany
[2]Fraunhofer FOKUS, Berlin, Germany

Abstract: This paper describes an approach for risk-based testing of Bluetooth functionality in an automotive environment, recently studied as part of the ITEA-2 research project DIAMONDS.

In the past two decades the functionality of infotainment systems of a modern car increased in complexity. Based on the worldwide evolution towards an information age the driver requests permanently increasing number of functionalities for infotainment systems where the vehicle becomes a connected vehicle which is always on. For instance next generation infotainment systems will provide internet to the cabin and will have integrated of applications like Twitter and Facebook.

These additional online services can be realized by integrating the driver's smartphone with the vehicle infotainment system whereby the phone becomes a logical part of the infotainment head unit. This intense integration of consumer mobile devices with the vehicle electronics and network can also have a negative impact on the security of the vehicle electronics. Even the safety level of the entire automotive system can be affected, which recent studies have shown [Ko10, Ko11].

With respect to these increasing security risks for connected vehicles and future telematics applications, efficient and structured methods are required to verify such systems concerning their robustness against security attacks. One important approach is a risk oriented model-based testing concept as developed in the research project DIAMONDS[1]. Within this project a case study has been defined by Dornier Consulting for the automotive domain focused on connected car applications such as the integration of mobile consumer devices with the in-vehicle network and electronics.

Based on risk analysis a security related prioritization of messages is achieved, which enables a smart and efficient generation of the most relevant test scenarios. Using adapted security test generation methods based on known concepts, such as

[1] ITEA (Information Technology for European Advancement) DIAMONDS project, http://www.itea2-diamonds.org

fuzzing of these pre-defined test scenarios, will then be used to generate the final set of test cases.

This paper discusses the generation process for risk based security test cases using this model-based testing approach, which pays attention to the analysed security flaws. An approach that refers to the generation of test cases that will cover possible unknown flaws by integrating fuzzing methodologies will also be described. The practical implementation of this generation process will be demonstrated by Dornier Consulting's extended Model-Based Testing (MBT) framework do.ATOMS for industrial and commercial application.

1 Introduction

As Information and Communication Technology (ICT) systems become more and more part of our daily lives, current and future vehicles are more and more integrated into ICT networks. The consumer's smartphones, multimedia devices etc. are linked to the vehicles and allow the driver or passengers to use the internet, to access their private phone books or to run their individual applications through the vehicle's integrated infotainment systems.

Today's most common technology to link consumer devices to in-vehicle electronics is Bluetooth, which latest version is 4.0. Such a wireless connection provides the most comfortable way for the driver to access a variety of services. However such a wireless connection implies also the risk of possible misuse which leads to enhanced security issues and risks for the automotive electronics and with that for the passengers.

Koscher et al. [Ko10] showed how today's vehicles can be hacked via security flaws and how these security flaws can even lead to safety critical scenarios for the passengers.

Recently the complexity of ICT systems and automotive electronics increases dramatically and requires modern verification and validation methods, which allow handling of complex systems fast and efficiently. One important approach is the Model-Based Testing (MBT) concept where even complex test scenarios and logics can be defined in models by using the Unified Modelling language (UML), the System Modelling Language (SysML) or other 4th generation languages. State-of-the-art concepts are described in Chapter 2.

Following this MBT approach for complex systems, security aspects of ICT systems and automotive electronics should also be integrated into such MBT methods. In 2010 the European ITEA-2 research project DIAMONDS has been founded by 23 industry and research partners from six countries to develop new methodologies and techniques for model-based security testing and to implement those new concepts into commercial tools for the industry. The DIAMONDS project is very much organized along so called case studies where the research results will be implemented and demonstrated in real industry applications. Preliminary results of the project are stated in Chapter 3.

One case study within the DIAMONDS project has been defined by Dornier Consulting for the automotive domain. It focus on an embedded connectivity module as part of an actual vehicle infotainment system which supports today's smartphones via Bluetooth link for various services/profiles such as hands-free phone, phone book access etc. This automotive case study demonstrates how the new developed model-based security testing concept and tools can be used in the automotive industry.

In Chapter 4 of this paper the first results and findings from the automotive case study in DIAMONDS are shown.

2 State-of-the-Art

2.1 Security in the automotive domain

Security testing in the automotive domain is quite a new discipline. Despite the fact that modern cars provide a wide variety of comfort or multimedia services the corresponding testing measures ensuring the service quality have been mainly taken from a more functional aspect or have been related to passenger safety. Especially the increasing number of wireless communication interfaces (e.g. for internet based services or related to car to car or car to infrastructure communication) lead to the fact that the attack surface of a modern car has increased quite dramatically (as can be seen in the Figure 1).

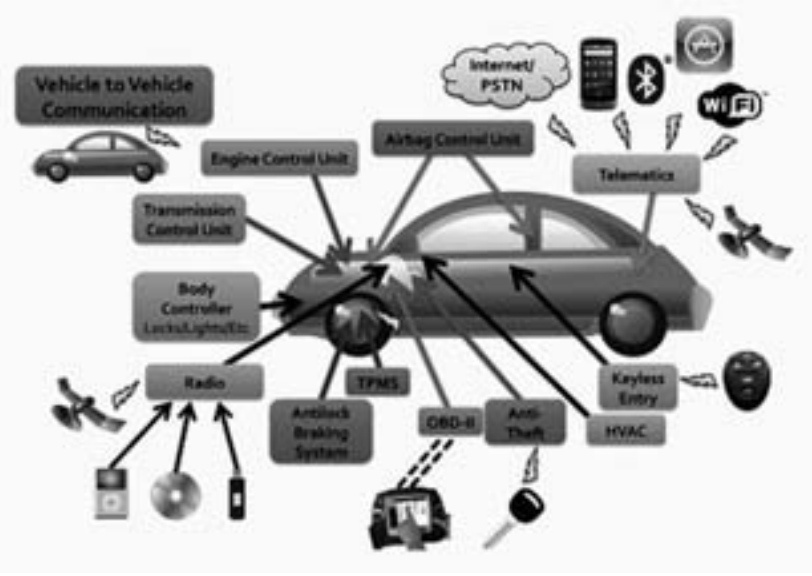

Figure 1: Digital I/O channels appearing on a modern vehicle (from [Ko11])

The main danger that arises from this initial position is the possibility that an attacker, who has no direct (in terms of "physical") access to the car, can nonetheless gain access to the car's interior electronic components. The communication of the car's internal Electronic Control Units (ECUs) relies on various bus systems (like CAN (Controller Area Network), LIN (Local Interconnect Network), or MOST (Media Oriented Systems Transport)) and it should be noted explicitly that the technical design of these bus systems alone does not provide any security features at all. For example, if someone is able to gain direct access to the car's CAN bus communication he can control all car components that are interconnected with the infected component (e.g. by successfully hijacking an ECU like a head unit by using exploits that are accidentally provided by an active Bluetooth connection of the head unit). This threat scenario can be considered as the definitive worst case since an attacker could potentially use the brake control unit to block the left front wheel completely causing the car to spin while driving on the motorway at high speed [Ko10, Ko11].

2.2 Risk-Based Security Testing

Security testing aims for finding any weakness or potential vulnerability that may result from poor or improper system configuration, known and/or unknown hardware or software flaws, or operational weaknesses in process or technical countermeasures. Unfortunately security testing, especially security vulnerability or penetration testing, lacks systematic approaches, which enable the efficient and goal oriented identification, selection, and execution of test cases. Similarly, classical test approaches address risks implicitly rather than systematically. Systems, functions, or modules, which are known to be critical, are tested more intensively than others. The basis of such a test planning is often a very simple and unstructured risk assessment, which usually is performed during or in preparation of the test process. In contrast, there are a number of systematic approaches that can be summarized under the term Risk-Based Testing (RBT) in general and under the term Risk-Based Security Testing (RBST) when security risks are directly addressed.

RBT approaches have in common that the identified systemic risks become the central factor that is used either for a systematic identification of test objectives, for scoring and selecting test cases, for the evaluation of the test quality or for assessing the test results. For example Redmill [Re04, Re05] describes the use of risk analysis to optimize the test planning. The quantified risks are used to prioritize the testing and the analysis of system failures and their impact. Amland [Am00] calculated the risk of the individual software functions and the expected failure probability for these functions weighted with risk factors such as design quality, size, and complexity of the evaluated software. Finally Zimmermann et al. [Zi09] and Stallbaum et al. [SMP08] describe model-based approaches, which allow the automatic derivation of system test cases from different kind of functional models as well as their prioritization based on risk. The latter is called RiteDAP (Risk-based test case Derivation And Prioritization) and uses activity diagrams

to automatically generate test cases. The models are annotated with risk values that are used for the test case prioritization.

Risk-Based Security Testing characterize a methodology that makes software security risks the guiding factor to solve decision problems during testing, e.g. the selection and prioritization of test cases. General technical recommendations on security testing techniques [He03, MTL09] already propose the use of risk analysis results to guide security testing. However, these recommendations are very general in nature and describe in this sense no real methods for risk-based testing. A more detailed approach is described in [Ze11]. The model-based approach for risk-based security testing of cloud environments is based on so called negative security requirements. These requirements are derived from risk analysis in order to cover those security aspects not covered by existing positive requirements. Misuse cases are then defined based on the negative requirements.

One of the key challenges of security testing in general is the detection of potential weaknesses and vulnerabilities on the system under test (SUT) to evaluate the associated risks as accurately as possible. Fuzz testing or fuzzing is a black box testing technique that is broadly used for that purpose. It consists of injecting invalid input data into a system and assessing how it would react [Be11]. The aim is to find deviations of the SUT to its specification (unexpected behaviour, errors, and weaknesses) that lead to vulnerabilities because invalid input is processed instead of being rejected. Such deviations may lead to undefined states of the SUT and can be exploited by an attacker, e.g. by successfully performing a denial-of-service attack because the SUT is crashing or hanging [TDM08]. Because it relies on injecting invalid input data to the SUT, fuzzing is often referred to as negative testing or as interface robustness testing.

Depending on the amount of knowledge about the SUT's protocol, there are different categories of fuzzers, from dumb fuzzers that nearly know nothing about the protocol up to smart fuzzers that use a model of the protocol for communicating with the SUT. The more a fuzzer knows about the protocol, the more efficient is the fuzzer in finding errors [Ta09]. Original fuzzers randomly generate input data for the SUT. Because of this, most generated input data will be rejected because it is invalid in such a large portion that the SUT won't parse it. Hence, most test cases that inject randomly generated data will not find bugs because the invalid data is not processed by the SUT. For that reason, the goal is to generate so called semi-valid input data meaning that the input data is invalid only to a small extend. In order to achieve this goal knowledge of the format of valid input data is needed. A smart fuzzer can employ this knowledge to generate semi-valid input data that may pass the SUT's parser and penetrates deeper into the SUT [Oe05, Ta09]. There, the semi-valid data is processed and may expose security-related weaknesses in the implementation. Additionally, model-based fuzzers can user their knowledge of the protocol behaviour to generate semi-valid input data only for certain messages in certain protocol states, e.g. after authentication.

A big issue while fuzzing is how to determine that a weakness was found. In functional testing, for each input data there exists an expected output that can be compared to the actual output of the SUT. Because fuzzing uses invalid input data to find security-related weaknesses, the expected reaction of the SUT is difficult to determine. At best, it is

specified how the SUT should react on invalid input. But if the SUT does not reject the invalid input data that does not necessarily mean that a weakness was found. Obvious weaknesses or vulnerabilities that can be easily detected are a crash or a hang of the SUT. But a weakness does not need to be exposed immediately after submitting invalid input data. Abnormal behaviour of the SUT can occur much later and it can be more subtle than a crash, e.g. performance degradation or increased memory consumption.

A simple approach in finding hints whether the SUT could be affected by a weakness stimulated by invalid input data is valid case instrumentation [TDM08]. After each fuzz test case a functional test case is executed to determine if the SUT behaves normally. If not, the previous fuzz test case probably found a weakness. Monitoring of the SUT can also assist in finding more subtle weaknesses.

3 Risk-Based Model-Based Testing

The DIAMONDS project develops a risk-based model-based security testing methodology being demonstrated on case studies from a variety of application domains, including the banking and transport domain. Risk-based testing approaches can help to optimize the risk analysis and the risk assessment itself. Risk analysis and risk assessment, similar to other development activities in early project phases, are mainly based on assumptions on the system itself. Risk-based testing results can act as an indicator for the correctness of the assumptions that have been made during the risk analysis and risk assessment. In particular, risk-based testing can help in

- reducing the likelihood of security incidents by discovering unknown vulnerabilities,
- optimizing risk analysis by identifying new risk factors and reassessing the probabilities of risks and their consequences,
- providing further evidence on the functional correctness of countermeasures.

Risk-based model-based security testing is an approach being developed in the DIAMONDS project, combining risk analysis and MBT techniques to assess the security properties of a software-based system. The key goal of the approach is to allow both process groups, i.e. risk analysis and testing to benefit from each other in a consistent and systematic manner. As illustrated by Figure 2, the risk-based model-based security testing process can involve several iterations whereby risk-analysis always plays a central role.

Figure 2: Overview of the Risk-Based Model-Based Security Testing Process

The next sections describe each of the phases of this process.

3.1 Risk Analysis

The DIAMONDS approach proposes to use multiple well-known sources as input for the risk analysis phase: Weakness and vulnerability enumerations from known bodies (e.g. MITRE's CWE, and CVE), attack pattern enumerations as well as security standards (e.g. ISO 27000 series). This will result in a well-defined CORAS threat model including assets, threat scenarios, and vulnerabilities. The CORAS language [LSS11] integrates different tree-based approaches for risk modelling. It is a graph-based modelling approach that emphasizes the modelling of threat scenarios and provides formalisms to annotate the threat scenarios with probability values and formalisms to reason with these annotations.

3.2 Test Model Design

To find an optimal set of test cases, one requires an appropriate identification and selection strategy. Such a strategy on the one hand needs to take into account the available test budget and on the other hand to provide, as far as possible, the necessary test coverage. In functional testing, coverage is often described by the coverage of requirements or the coverage of model elements such as states, transitions or decisions. In risk based testing the aim is for coverage of the identified risks of a system. Yet, security testing often lacks systematic approaches that enable the efficient and goal oriented identification, selection, and execution of test cases.

Within DIAMONDS, a framework was developed to resolve this problem. RBMBT is used to characterize a methodology that makes software risks the guiding factor to solve decision problems during testing, e.g. the selection and prioritization of test cases. A comprehensive risk assessment introduces the notion of risk values, that is the estimation of probabilities and consequences for certain threat scenarios. These risk values are then used to weight threat scenarios and thus help identifying which threat scenarios are more relevant and thus identifying the threat scenarios that are the ones that need to be treated

and tested more carefully. The DIAMONDS framework therefore allows to systematically capture security risks (i.e. vulnerabilities and threat scenarios), treatments and risk values and trace them to test artefacts so that test identification and test selection within the test model design is effectively supported. The implementation of this test case prioritization approach as part of the Dornier Consulting case study is described in Chapter 4.2.

3.3 Test Generation

In addition to traditional fuzzing – with the selection of the messages depending on the message prioritization from the Test Model Design as described in Chapter 3.2 – the DIAMONDS approach injects invalid sequences of messages into the SUT. This kind of fuzzing is referred to as behavioural fuzzing. When behaviour fuzzing, the input data of the messages remains disregarded, only the order of messages is considered. The goal is to find security relevant weaknesses that consist rather in errors of the implemented state machine than in processing invalid input data. To a minor degree this has already been done by using state machines where invalid transitions are taken to determine the next message to be sent [Be10, HSD08]. This is rather random fuzzing than smart fuzzing because only invalid transitions are considered independently of their special meaning.

An example of a vulnerability that can be found only by behaviour fuzzing is given in [TMK10]. In a regular HTTP 1.1 GET request, two messages are sent to the web server: a GET message, that contains the requested document, and a host message that contains the domain name of the host that is requested. Earlier versions of Apache web server are vulnerable for a denial-of-service attack that consists in repeating the host message of a request. Repeating the host message can result in a crash of the web server. This vulnerability cannot be found using data fuzzing.

A more sophisticated kind of behaviour fuzzing can be done using UML sequence diagrams. Since UML 2, sequence diagrams can contain control structures like combined fragments, e.g. for branches and loops, that are guarded by constraints. By breaking these constraints, invalid behaviour can be triggered in a more model-based manner by evaluating more information of the model than just the order of states or messages. Moreover, different sequence diagrams can be merged starting with a valid sequence of messages representing one functional behaviour and ending with another.

3.4 Test Execution

Test execution is the phase whereby test cases obtained with the risk-based model-based security testing techniques proposed by the DIAMONDS project are actually executed against an SUT. Therefore this phase takes as input the results of the test generation phase, which may be automatically executable test scripts, manually executable test scenarios, or a combination of both. As depicted in Figure 2 the output from the test execution phase may consist of new weaknesses and vulnerabilities, which in turn would

provide input for a new iteration of the process, starting all over again from the risk analysis phase. An important benefit of the risk-based model-based security testing with regard to test execution is the fact that it facilitates the traceability of test results back to elements of risk analysis (threats, vulnerabilities, weaknesses, treatments …) they originated from, as well as to elements of system design potentially affected by the identified risks. This form of feedback will eventually have a positive impact on the overall security of the SUT.

4 Case Study

For this research project a connectivity module was chosen to represent the automotive domain. The so called connectivity module is used in vehicles to interconnect user electronic devices with the car's own infotainment system and in-vehicle network via Bluetooth and USB. The connectivity module is connected via CAN to the vehicular network as shown in Figure 3.

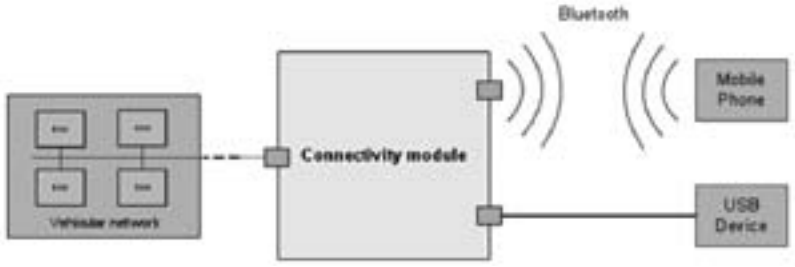

Figure 3: Connectivity module as used in the case study of Dornier Consulting

For the case study an existing test environment is used. This test environment provides a physical set-up of selected dashboard components of an automotive OEM. The behaviour of the set-up is described with a UML model, which basically consists of different sequence diagrams. The main use case which is being investigated in more detail in this case study is the connection of a Bluetooth device to the car's connectivity and communication module and the usage of several Bluetooth-based services.

The UML-based system model of the SUT is supplemented with a dedicated UML profile that was tailored to address the needs of the automotive domain. It is called Automotive Domain Modelling Language (ADML) and is part of Dornier Consulting's testing framework do.ATOMS. Most of the SUTs that are tested with do.ATOMS are reactive embedded real-time systems, e.g. head units, on-board-units and others which are relevant in the automotive or telecommunication domain. The test case design and execution supports modelling parallel, loop, and conditional activities, where discrete and continuous signals like brake/acceleration-sequences, speech recognition, tone detection, GSM calls, GPS-tracking, voltage-ramps, and other scenarios can be tested under real-time conditions.

The do.ATOMS framework

The name “do.ATOMS” stands for Automated Testing of Model Based System Design. Following this approach the framework serves as a MBT environment for discrete interconnected embedded systems. Test cases are generated from a given system model. The system model is based on UML and specifies the structure as well as the behaviour of the SUT. The system model gets processed by one of several model parsers – depending on the UML tool used for creating the system model (e.g. Sparx Systems Enterprise Architect or IBM Rational Rhapsody). The test cases can be transferred into a test case library and can be edited in an easy to use editor (see Figure 4) that shows the transformed test cases as workflows. During test case creation the system model parser also parameterize the test cases so that they are directly executable on the corresponding SUT test setup. The test case framework also handles test case management, execution, and reporting as well. There are also interfaces to support external test management solutions like HP Quality Center. The execution layer of do.ATOMS has a direct access to the interfaces provided by the SUT and follows a service-orientated approach. This gives do.ATOMS the flexibility to support a wide variety of different communication, data exchange, and functional interfaces which also simplifies the integration into the SUT environment.

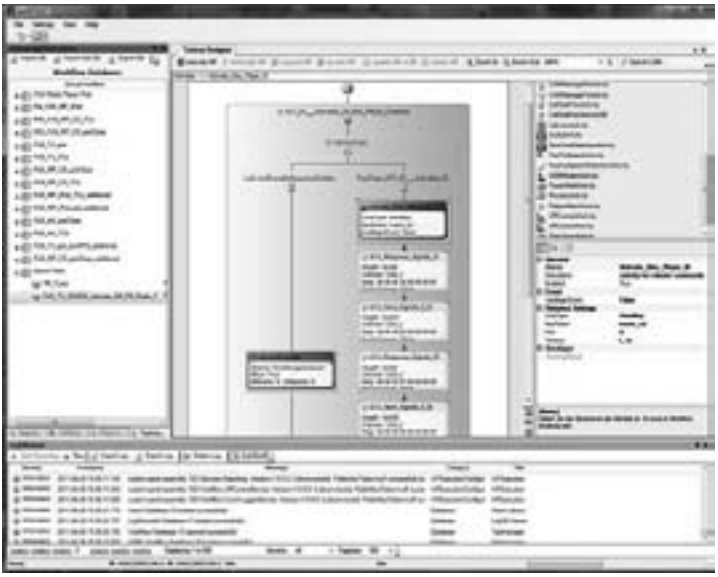

Figure 4: The do.ATOMS interface

Using do.ATOMS as a stable and flexible base Dornier Consulting has developed several concrete products on top of it. These products play a vital role in the realization of the company’s projects and are also used by its customers. One product example is do.ARTS (Automatic Robot Test System), a robot controlled test bench solution for

automated and model-based testing of complex Human-Machine Interface (HMI) functionalities.

In the progress of the DIAMONDS project the following aspects of do.ATOMS are being extended or enhanced:

- Support for new approaches and methodologies that allow the specification, generation and execution of IT security related tests in an automotive context (e.g. risk analysis and modelling).
- System modelling techniques get extended to allow the modelling of security requirements and features of the SUT.
- The model parser components will be able to convert the security requirements of the system model into corresponding security relevant test cases.
- Test case execution gets extended to be able to analyse and to verify the conformity of the SUT's security requirements.

4.1 Risk Analysis

The test approach of Dornier Consulting is based on a risk analysis made in CORAS as mentioned in Section 3.1. Each diagram in CORAS represents a threat scenario, which can be reached by using several different vulnerabilities. These vulnerabilities are applied with a value from 0.0 to 1.0, where 0.0 represents no protection and 1.0 represents full protection.

For each vulnerability an own connection from the threat to the threat scenario is defined. The connection is of type ‘initiates’ and is applied with a likelihood. According to the CORAS language specification the ‘threat scenario’ leads to an ‘unwanted incident’, which impacts an ‘asset’. All connections are applied with a likelihood-value, just like the ‘initiates’ connection. In Figure 5 the threat scenario ‘DoS without Pairing’ is shown as an example.

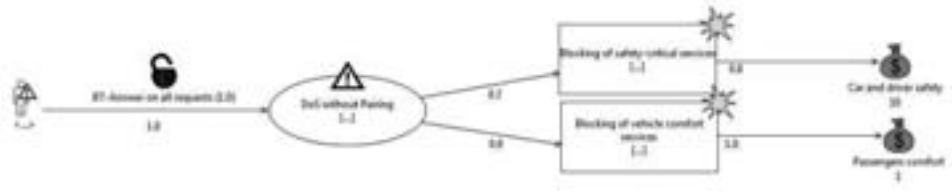

Figure 5: CORAS model for the threat scenario 'DoS without Pairing'

For this approach the used terms have the following meaning:

Likelihood Represents the likelihood that an event happens, which is expressed by the connection.

Vulnerability By vulnerability a flaw in the system is addressed. The value at the vulnerability represents the degree of protection. The main purpose of which to prevent such an event. Therefore 1.0 means full protection and 0.0 means no protection at all.

Consequence The consequence is applied to an asset. The value of the consequence represents the dimension of the consequence with respect to security. The higher the values, the stronger is the security impact.

The elements shown in Figure 6 are used in a CORAS diagram. The diagram usually begins with a **threat**, which has an 'initiate' connection to a **threat scenario**. From here a 'leads to' connects one or several **unwanted incidents**. The **unwanted incidents** have an 'impact' connection to the assets.

Figure 6: CORAS legend of used elements

4.2 Test Model Design

In order to cover the complete consequence of a vulnerability, it is necessary to iterate over each path of the CORAS diagram. The following equation for the calculation is used. N represents the number of possible paths from the threat scenario to the asset.

$$consequence = \sum_{i=1}^{N} (likelihood_{LeadsTo_i} \times likelihood_{Impacts_i} \times consequence_i)$$

In the example of Figure 5 the consequence would be 6.5, which was calculated by summing up both paths:

$$0.7 \times 0.8 \times 10.0 + 0.9 \times 1.0 \times 1.0 = 6.5$$

Based on the values for the vulnerability, likelihood, and consequence a score can be calculated. For this approach the following equation is used:

$$security\ score = \frac{(likelihood \times consequence)^2}{vulnerability}$$

This is a simple approach to categorize the different vulnerabilities. In this approach the highest security score does not represent the most relevant message, but indicates a general relevant message in respect to security.

In the example of Figure 5 the likelihood is 1.0, the vulnerability 1.0, and the consequence 6.5 as calculated in the previous paragraph. After putting these values into the presented equation we get a value of **42.25**.

To generate a sufficient set of test cases based on the system model, it is necessary to map the calculated security score to the system model. The most uncompromising way would be to model each path of CORAS to the system model. But this is not practical, since some threat scenarios are too complicated to model (e.g. exploits). Therefore the approach requires a scenario for each vulnerability to be modelled. These 'CORAS scenarios' describe how to reach certain vulnerabilities and are added in a separate package to the system model. It should be noticed that with this modification the system model becomes a test model.

In the sequence diagram 'functions' are modelled to represent the messages sent between parts. The messages used in the 'CORAS scenario', which are derived from the CORAS analysis, have the same identifier as the messages used in the system model. Therefore the security score of the messages of the 'CORAS scenarios' can be mapped to a single message of the system model. If a message is used in several 'CORAS scenarios' the security score will be aggregated to a single value. With this approach messages of the global system model can be applied with a security score.

For instance all the messages, which lead to the threat scenario 'DoS without Pairing', will get a value of 42.25. If one of these messages is used in another CORAS scenario as well, the value of the other CORAS scenarios will be added to the 42.25.

For testing purpose the security score is used to calculate a priority of the different messages. Together with the occurrence of a single message in the system model a priority is calculated which indicates the testing framework the relevant messages with respect to security. This priority influences the test case generation.

The presented approach allows mapping the CORAS risk analysis to the system model. The introduced equation for the security score is a simple way to establish a security ranking between the different vulnerabilities of the CORAS risk analysis. Based on the analysis the scenarios can be extracted to the system model. The containing messages can then aggregate the security scores of the CORAS scenarios, in which they are included. In combination to the occurrence of a message within the system model a priority order can be introduced with respect to security.

4.3 Test Generation

As mentioned in the do.ATOMS overview, during test case generation the system model gets analysed by one of the framework's model parsers and is then transformed into a do.ATOMS internal representation, the so-called workflow. These workflows in turn can also be considered as an independent test model, which is – in this case – automatically derived from the system model. This step is therefore also called "model-to-model-transformation". Each workflow consists of several workflow activities that interact with the SUT (e.g. via sending a CAN or Bluetooth message) or that monitor its state (verification of the presence of certain CAN messages and the like). The sequence of the activities can also be altered in various ways using workflow flow control constructs (like if/else branches or loops). All test cases follow the uniform approach that do.ATOMS

simulates the external Bluetooth end user device to test the Bluetooth connectivity features of the connectivity module.

During the transformation phase security relevant information (like the security score metadata) that is part of the system model is also transferred and is still present in the do.ATOMS workflow model. This allows the application of security testing techniques like data fuzzing.

Each derived test case (workflow) that is transferred into the test case library of the do.ATOMS test case designer can be transformed into a fuzzing test case. A fuzzing test case can be seen as a "malicious" counterpart of a test case that represents the normal system behaviour. As mentioned before, the workflow activities are responsible for sending stimuli to the SUT.

For finding entry points for data fuzzing in these activities the following approach was chosen: Each activity of the test cases is examined and its security score information is evaluated. If the security score is above a configurable threshold, the activity is enclosed by a so-called fuzzing container. This element is responsible for injecting the fuzzing functionality by manipulating its child element e.g. the payload of a message. This can be done at design time when the workflow is created or at runtime when the workflow gets executed.

Figure 7 shows an excerpt of a fuzzing workflow that handles the Bluetooth pairing process (e.g. between the connectivity module and the driver's mobile phone). The data fuzzing container has been applied to "EnterPIN_I4" activity, which in this example has the highest security score with a value of 532.

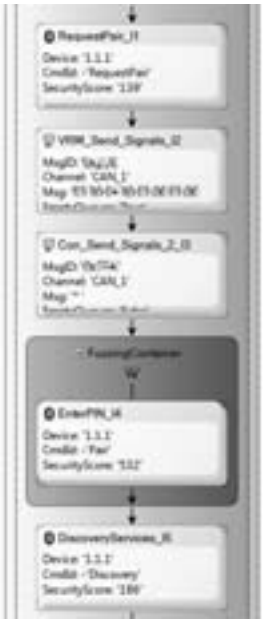

Figure 7: An excerpt of a fuzzing workflow that handles the Bluetooth pairing process

To sum up, the following figure shows the complete conceptual approach for test case generation that is realized with do.ATOMS:

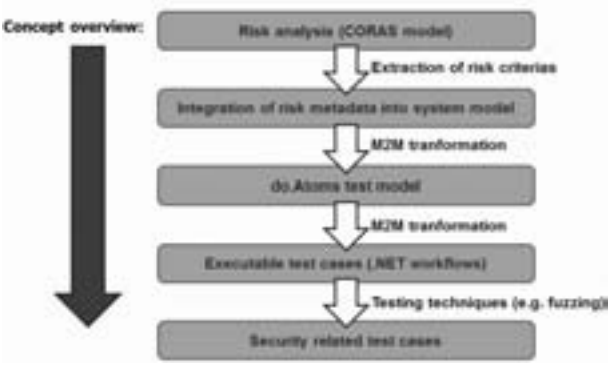

Figure 8: An overview of the case study approach

4.4 Test Execution

In general, the execution of a fuzzing test case is somewhat different compared to the execution of a normal test case. What differs the most is the process of determining the test result. When a functional test case is finished the SUT can be examined is some way

(e.g. by inspecting the state of the SUT) in order to find out whether the test was successful or not. When it comes to the execution of fuzzing test cases the situation changes since the determination of the test result is often not clear. For each fuzzing test case one has to find measurable criteria for deriving the test result, e.g. the SUT could ignore a fuzzing attack completely and just continue to function normally or it faces a fatal crash.

In terms of do.ATOMS-based test execution the workflow itself defines the test case control flow, so each activity (test step) is executed in the pre-defined order and triggers a certain test step functionality (like sending or validating CAN or Bluetooth messages to the SUT or triggering an image-based validation of the head unit display). This is also the case for fuzzing test cases – the main difference is the fuzzing container which manipulates its child activity to apply data fuzzing. The particular manner in which this data fuzzing is applied depends on the action performed by the child activity.

Concerning data fuzzing, the creation of semi-valid input is of particular importance. In general, how this input is generated depends on the specific protocol that is fuzzed. Taking the example of sending a PIN to the connectivity module the fuzzing container investigates the child activity and identifies the parameter, which represents the PIN. It inspects the data type of the parameter, which in this case is specified as “string”, consisting of at least a 4-digit number. Depending on the data type various fuzzing methods are applied by the fuzzing container to create a semi-valid version of the expected input data (like sending characters instead of digits or a PIN with an unexpected length, e.g. consisting of 1000 random digits). Additionally it can also be helpful to use invalid characters that have special meaning, for instance null characters that terminate C strings, or the percentage character that is used for formatted strings. While a dumb fuzzer would generate invalid input data for most of the messages, a model-based fuzzer has enough knowledge to fuzz only the parameters of a certain activity, e.g. the activity sending the message that carries the PIN. The fuzzing container itself can also be configured in various ways to control its runtime behaviour. One example is that the container can be instructed to create a new fuzzed input value for each test run.

This fuzzing approach is realized by implementing the fuzzing library provided by Fraunhofer FOKUS. This library encapsulates all fuzzing related business logic and allows do.ATOMS the retrieval of fuzzed input data values by simply sending an XML request. This request contains a definition of the data type or structure that should be fuzzed and the fuzzing generators that should be applied to it. The fuzzing library will also be capable of supporting behavioural fuzzing which will be applied to the Dornier case study later on as well. Behaviour Fuzzing will affect the order in which the workflow activities get executed.

5 Conclusion

Increasing complexity in ICT systems and automotive electronics as well as the increased integration of vehicle networks with ICT networks lead to increased IT security risks for the consumer. This trend and the accelerating development cycles for new

products, features, and applications in this area require efficient and fast development and integration methods and tools.

One important approach is to transfer model-based development concepts to the test and integration phases by introducing MBT. Due to the mentioned increase of security issues in ICT and automotive electronics, within the European research project DIAMONDS the MBT approach is currently extended to security focused testing.

Within DIAMONDS the authors developed, together with project partners from six European countries, a new method and process on how to efficiently generate security focused test scenarios and test cases.

This new risk-based methodology has been applied to identify possible flaws in the implementation of a vehicle infotainment system by analysing its UML system model. Based on this risk analysis a security related prioritization of the different Bluetooth messages can be achieved which enables a smart and efficient generation of the most critical and relevant test scenarios. Adapted security test generation methods based on known concepts such as fuzzing of these pre-defined test scenarios will then be used to generate the final set of test cases.

This paper discussed the generation process for risk based security test cases using the model-based testing approach, which focus on the analysed security flaws. An approach that refers to the generation of test cases that cover possible unknown flaws by integrating data and behaviour fuzzing methodologies is also be examined.

The practical implementation of this generation process has been demonstrated by the extended MBT frame work do.ATOMS from Dornier Consulting for industrial and commercial use and application.

Within the next months the developed methodology will be concretized, optimized, and validated through the automotive case study to demonstrate the efficiency, robustness, and usability based on the implementation in the do.ATOMS test design and execution frame-work.

6 References

[Am00] Amland, S.: Risk-based testing: Risk analysis fundamentals and metrics for software testing including a financial application case study. J. of Systems and Software 53 (3), 287-295 (2000)

[Be10] Becker, S., Abdelnur, H., State, R., Engel, T.: An Autonomic Testing Framework for IPv6 Configuration Protocols. In: Proceedings of the Mechanisms for autonomous management of networks and services (AIMS'10), pp. 65-76. Springer (2010)

[Be11] Bekrar, S.; Bekrar, C.; Groz, R.; Mounier, L.: Finding Software Vulnerabilities by Smart Fuzzing. In: Proceedings of the 2011 Fourth IEEE International Conference on Software Testing, Verification and Validation, pp. 427-430. IEEE Computer Society, Washington, D.C. (2011)

[GOR07] Güdemann, M., Ortmeier, F., Reif, W.: Using Deductive Cause Consequence Analysis (DCCA) with SCADE. In: Saglietti, F., Oster. N. (eds.): Proceedings of SAFECOMP

2007, LNCS, vol. 4680, pp. 465-478. Springer (2007) [He03] Herzog, P.: OSSTMM 2.1. Open-Source Security Testing Methodology Manual. Institute for Security and Open Methodologies (2003)

[HSD08] Hsu, Y., Shu, G., Lee, D.: A Model-based Approach to Security Flaw Detection of Network Protocol Implementations. In: Proceedings of IEEE International Conference on Network Protocols (ICNP 2008), pp. 114-123. IEEE (2008)

[Ko10] Koscher, K., Checkoway, S., et al.: Experimental Security Analysis of a Modern Automobile. In: Proceedings of the IEEE Symposium on Security and Privacy (S&P'10), pp. 447-462 (2010)

[Ko11] Koscher, K., Checkoway, S., et al.: Comprehensive Experimental Analyses of Automotive Attack Surfaces. In: Proceedings of the 20th USENIX conference on Security. USENIX Association, Berkeley (2011)

[LSS11] Lund, M.S., Solhaug, B., Stlen, K.: Model-Driven Risk Analysis. The CORAS Approach. Springer (2011)

[MTL09] Murthy, K. K., Thakkar, K. R., Laxminarayan, S.: Leveraging Risk Based Testing in Enterprise Systems Security Validation. In: Proceedings of the First International Conference on Emerging Network Intelligence (EMERGING 2009), pp. 111-116 (2009)

[Oe05] Oehlert, P.: Violating Assumptions with Fuzzing. IEEE Security & Privacy, vol. 3, no. 2, pp. 58-62. IEEE Computer Society, Los Alamitos (Mar.-Apr. 2005)

[Re04] Redmill, F.: Exploring Risk-based Testing and Its Implications: Research Articles. J. Softw. Test. Verif. Reliab. 14 (1), 3-15 (Mar. 2004)

[Re05] Redmill, F. 2005. Theory and practice of risk-based testing: Research Articles. J. Softw. Test. Verif. Reliab. 15 (1), 3-20 (Mar. 2005)

[SMP08] Stallbaum, H., Metzger, A., Pohl, K.: An Automated Technique for Risk-based Test Case Generation and Prioritization. In: Proceedings of the 3rd international workshop on Automation of software test (AST'08), pp. 67-70. ACM New York (2008)

[Ta09] Takanen, A.: Fuzzing: the Past, the Present and the Future. Actes du 7ème symposium sur la sécurité des technologies de l'information et des communications (SSTIC), pp. 202-212 (2009)

[TDM08] Takanen, A., DeMott, J., Miller. C.: Fuzzing for Software Security Testing and Qualitiy Assurance. Artech House, Boston (2008)

[TMK10] Takahisa, K., Miyuki, H., Kenji, K.: AspFuzz: A State-aware Protocol Fuzzer based on Application-layer Protocols. In: Proceedings of Conference on IEEE Symposium on Computers and Communications (ISCC 2010), pp.202-208 (2010)

[Ze11] Zech, P.: Risk-Based Security Testing in Cloud Computing Environments. PhD Symposium at the Fourth IEEE International Conference on Software Testing, Verification and Validation (ICST 2011), pp.411-414 (2011)

[Zi09] Zimmermann, F., Eschbach, R., Kloos, J., Bauer, T.: Risk-based Statistical Testing: A Refinement-based Approach to the Reliability Analysis of Safety-Critical Systems. In: Proceedings of 12th European Workshop on Dependable Computing (2009)