Model-based security testing

Deriving test models from artefacts of security engineering

Armin Lunkeit
OpenLimit SignCubes GmbH
Berlin, Germany
armin.lunkeit@openlimit.com

Ina Schieferdecker
Fraunhofer FOKUS / Technical University Berlin
Berlin, Germany
ina.schieferdecker@fokus.fraunhofer.de

***Abstract*— Security-related tests should check and validate in engineering that the functionality derived from the security requirements has been correctly implemented. The design of effective tests for this security functionality requires, in addition to test models, a procedure for obtaining these models from previous artefacts of security engineering. This paper describes such an approach and focuses on the use of the artefacts of previous phases of security engineering for security testing.**

Keywords—security engineering; test models; security testing

I. Introduction

The integration of security engineering into software engineering is a key aspect for the development of secure software-based systems. Among other things, [1] discusses the state of the art and makes a proposal for the model-based integration of both disciplines. Testing can benefit from the results of previous work in security engineering. Software security aspects can be rather complex, especially for larger systems, so model-based approaches are helpful to properly address this complexity and meet the demands of effective testing of the security properties of a system. In this context, it should be noted that the definition of security engineering as used by us does not cover only the phase of requirement elicitation and consolidation or system design architecture. Rather, security engineering spans all phases of the software lifecycle. The security-oriented test is a self-contained sub-activity within this approach and relies on the results of previous phases.

The complexity of the interface between requirements, architecture and tests in terms of IT-security is higher than commonly assumed. This manifests itself when vulnerabilities become known and the subsequent analysis shows that covert security requirements exist which not covered in the test due to incorrect or incomplete models of the addressed security problem. Without the model-based approaches of security engineering, such problems are inevitable. By contrast, the model-based definition of the security problem supports finding an optimal test architecture and defining safety-oriented tests that go far beyond functional testing.

II. State Of The Art

Security-oriented testing is part of security engineering. Like in software engineering in general, often model-based approaches are used [1], [2], [3]. Model-based security engineering combines model artefacts of various phases of software and security engineering. A fundamental basis is both the existence of architectural and functional models as well as the definition of threats and potential vulnerabilities. Our approach incorporates the modelling approaches of CORAS [19], fault trees [24], event trees [25] and threat modelling [26].

In security testing, basically functional security tests – testing if the security features function correctly, heuristic security tests (aka fuzz tests) – testing against unknown attacks or unexpected usages, and penetration tests – testing along known vulnerabilities – are being differentiated. Known vulnerabilities can be obtained, among others, from the National Vulnerability Database (NVD, https://nvd.nist.gov/), the MITRE CVE repository (https://cve.mitre.org) or from other sources such as the Google Hacking Database (GHDB, https://www.exploit-db.com/google-hacking-database/).

According to the Common Criteria methodology [4], functional security tests relate to security functional requirements (SFR's). The aim of the functional security tests is to check that these functional security requirements have been implemented correctly. For the interfaces that allow these security requirements to be addressed, test cases are generated with the additional aim of checking that security functionalities cannot be bypassed. A common model-based approach is to generate test suites automatically to achieve targeted test coverage as well as to avoid redundant tests for functionalities already covered [5], [6].

Functional security tests are used at system level (system tests), at module level (module tests) and at the level of individual code artefacts such as classes and functions (unit test). Module and unit testing is typically used within development because of its proximity to code artefacts and popular test frameworks such as JUnit [7] or CPPUnit, while system level testing is often carried out by independent test teams. System-level functional security test can include many individual functions of a software in a single test case, as a prerequisite for addressing a specific functional security requirement. In the context of Common Criteria, this sometimes leads to very complex test scenarios since all functional security requirements are to be checked using the interfaces provided for this purpose. These interfaces provide access to the security functionality of the system under test (SUT) and are equivalent to what is declared to be an TSFI (TOE Security Functional Interface) in the Common Criteria

0-7695-6432-1/18/$31.00 ©2018 IEEE
DOI 10.1109/ICSTW.2018.00056

approach. For the purpose of checking functional security requirements, the testing of individual code modules and artefacts are obviated if the utilization of the TSFI's provide a sound coverage.

In case of module and unit testing, the complexity of the code of the implemented security requirements is essential. The relation of code complexity and error susceptibility are shown in [8], [9]. Manually developed module or unit tests are often very limited pertaining the number of functions and states and test input data. An alternative approach, symbolic program execution, is described in [10]. Symbolic execution generates test cases and test data with far better coverage. Test data is generated and purposefully based on the modelling of accepted inputs. In [11], the authors describe testing of the Linux *coreutils*, the *Minix* and *busybox* environment, and the *HiStar* operating system kernel using the symbolic program execution test tool KLEE. As a result, ten bugs in the Linux coreutils and 21 bugs in the *Minix* and *busybox* environment were identified. No details are given in the publication for the *HiStar* operating system kernel.

An alternative approach to symbolic execution is heuristic test data generation. As described in [12], the heuristic approach completes functional tests by searching for any noticeable behaviour in the system when confronted with seemingly random input data. Identified incidents, for example in the runtime of completed operations or steps of the tested system, are indicators of vulnerabilities. In this respect, heuristic testing as well as the penetration testing as described below differ from functional test as they explore the system in a randomized manner. Heuristic tests identify gaps in IT security that are not necessarily recognized by functional security testing. A variant of the heuristic test is automated fuzzing, in which the input vectors are generated for the function to be tested [13], [14].

Penetration tests are implemented both manually and automatically [15], [16]. Penetration testing requires a high level of expertise to identify the starting points to exploit known vulnerabilities. The aim of penetration tests is to check the practicability of circumventing an implemented security function. Penetration tests are well-known from the context of Web applications and network components [17]. However, penetration tests can be used in general. In addition, the line between heuristic testing approaches and penetration tests is fluid. Test vectors, which in the course of heuristic tests cause an unintentional system reaction, offer potential approaches to carry out penetration tests.

III. Key Concepts For the Security Problem Definition

The identification and processing of functional security requirements require a detailed understanding of security challenges for a given system. Modelling can help identifying and characterizing values to be protected – the so-called assets, applicable security objectives, threats, attacks and attackers, and their respective potentials. The core elements are briefly presented here, for a more complete description see [1].

A. Scenario Definition

A general concept is the scenario. Scenarios are used to outline the context of assets. Not every scenario involves all the assets. Likewise, the security objectives for an asset may vary between scenarios.

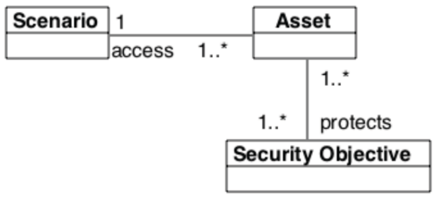

Fig. 1. Scenario Concept

B. Using Perspectives

The security problem is modelled using two perspectives.

1) The *socio-technical perspective* models the assets to be protected, applicable security objectives and threats from the point of view of the actors interacting with the system. The system is considered a black box.
2) The *technical perspective* models the assets to be protected, applicable security objectives and threats from the system viewpoint. Internal assets whose security objectives and applicable threats are characterized and linked to the objectives of external actors.

Both perspectives are stakeholder perspectives. The socio-technical perspective models the view of the user, meaning that the user has an interest in the implementation of the requirements. The technical perspective deals with the technical implementation of the system and provides the mapping to the system design and implementation. In this case, the project participants or the manufacturer are to be considered as stakeholders. The use of the model of the technical perspective ensures the fulfilment of the security objectives identified in the socio-technical perspective.

The socio-technical perspective examines the interaction of external actors with the technical system and describes values, security objectives and threats. External actors identify other values and security objectives that need to be protected than is the case with an exclusively technical view of the system (technical perspective). The advantage of using both perspectives is founded by the comprehensive analysis of all sub-aspects of the security problem.

C. Meta Model of the Security Problem Definition

The central element of the model is the asset. Assets may be resources or data. The context of assets is established by their associated security objectives, which lead to functional and non-functional requirements. This approach establishes the link between assets, security objectives and requirements.

New elements are attack potential, resistance and protection level. Attack potential qualitatively characterizes the abilities of attackers. The implementation of an attack corresponds to the capabilities of the attacker characterized in the attack potential. This characterization is a significant change to abstracted threats, as the abilities granted to the attacker are considered in the analysis. The resistance characterizes the effectiveness of a security mechanism. Both are in a 1: 1 ratio, with the security mechanism mitigating a threat. Resistance contributes to the level of protection which counteracts an attack in the context of the assigned attack potential and protects an asset. The security mechanism contributes to the achievement of a security objective.

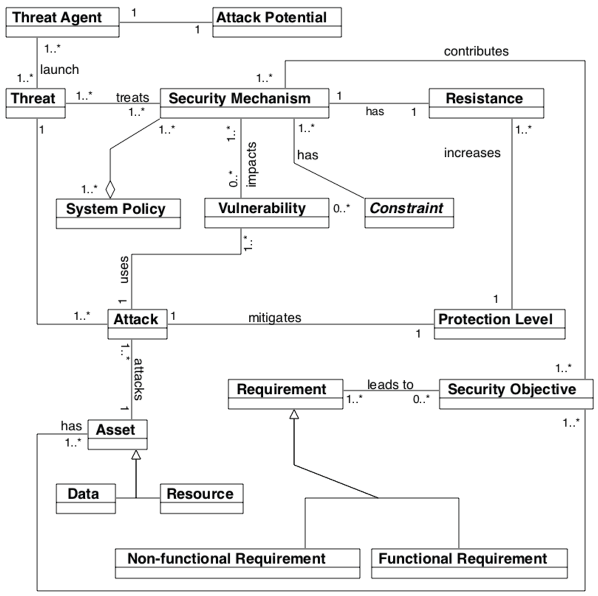

Fig. 2. Meta Model of the Security Problem Definiton

IV. The Framework For Security Engineering

The development of secure systems utilizes a framework for security engineering. The overview of such a framework is shown in Fig. 3. Model-based approaches such as Unified Modelling Language (UML) [27], [28], threat modelling [26] or the CORAS [19] framework for risk analysis are widely used throughout. In the implementation phase, in addition to the classical approaches such as code analysis, in particular domain-specific approaches are included. Specifically, mechanisms such as firewalls or Security-Enhanced Linux (SELinux) access rules are modelled using Domain-Specific-Languages (DSLs) and then converted into code. The code generation is done to limit the error sources if possible.

I. Derivation of the Test Model

A. *Integration of the analysis model with security testing*

The analysis model of the security problem provides the assets, the applicable security objectives, threats and attacks. From the combination of security objective and security mechanism, the mainly functional requirements with security relevance emerge. From this requirement reference, functional tests are derived on the basis of which the implemented security functionality is subjected to a test. This security functionality can take on a high level of complexity that is no longer fully testable based on functional testing. For this reason, a risk assessment takes place already during the identification of functional tests, on the basis of which they are prioritized.

Phase	**Activity**	**Models**	**Pattern**
Requirements, Design, Architecture	Security Problem Definition, Security Design and Architecture	Table based analysis of the SPD[1], Threat Modelling, UML, CORAS Risk Assessment	Security Engineering Pattern
Implementation	Code Reviews, Code Reading, Code Generation	UML, Domain Specific Approaches	Secure implementation Patterns**
Testing	Security Testing	UML, Threat Modelling, CORAS Risk Assessment	Security Test Pattern
Operation and Maintenance	Operation, Monitoring, Application of Patches	Table based analysis of the operational model, Threat Modelling, CORAS Risk Assessment	Secure Operations Patterns

Fig. 3. Security Engineering Framework

B. *The ETSI Framework for Risk-based Security-Testing*

A generic framework for using risk-based tests is defined in ETSI Guide ETSI EG 203 251 [18]. The procedure described there introduces the risk analysis before the test process and uses the results of this analysis in the definition of the test priorities. Risk analysis and security-based testing interact. This means that results of the security-based test can lead to new insights into the risk analysis and thus to the adaptation of the test strategy. The procedure outlined in [18] differentiates between security functional testing, performance testing, robustness testing and penetration testing. Between these vertical approaches, regression tests are used to record corrections and reassess them in the test. Unlike the model presented in [18], integrating the regression test into the verification and validation phase may be important. There can be several reasons for this. Products can include many features and related tests require a longer period of time. Against the background of time and cost pressure in software development, the use of regression tests with high modularization and differentiation of the security functionality to be tested can be advantageous.

C. *Threat Modeling*

Threat Modelling is a technique for identifying threats [23]. In order to conduct a risk analysis in connection with security-oriented tests, a model of the participating entities and data

[1] Security Problem Definition

flows is needed. Among other things, this model is a result of a threat modelling, whose task is precisely the identification and characterization of the trust boundaries, involved entities and data flows. Threat modelling is already used in the definition of the security problem and, in conjunction with the model of the security problem, provides a basis for the development of test models in security testing. Threat modelling does not have to be limited to the boundaries between the system being tested and its environment. It is also used to identify internal entities and data flows as needed; these can thus be the focus of the test. This is useful when security features are difficult or impossible to access through the outer interfaces of the system. Threat Modelling information can be used to design tests that target these internal interfaces. Examples include security domain separation tests, such as those based on SELinux, lightweight containerization (LXC), or virtualization.

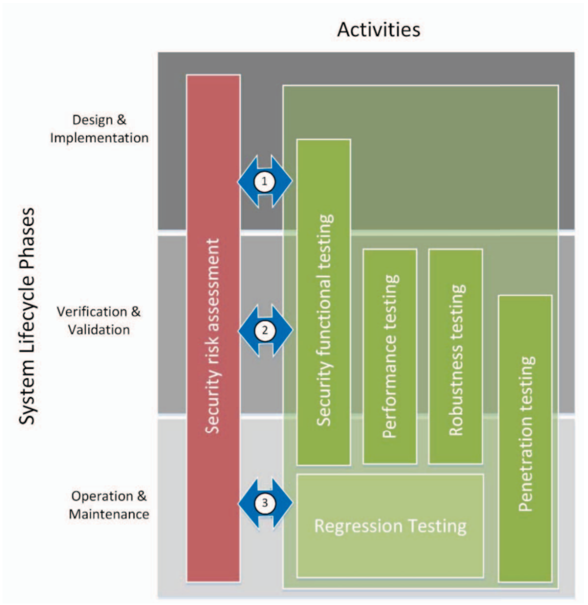

Fig. 4. System Security Lifecycle Phases [18]

A useful complement to Threat Modelling is the use of UML diagrams, which can serve as an artefact from the architecture and design process in the acquisition of the test models. UML models' static structures of components and classes and can also provide dynamic information about interactions. Threat modelling focuses more specifically on processes and data flow characterization, the roles of entities and trust boundaries.

D. *Use of the results in the risk analysis*

Threat Modelling identifies the interfaces between the system and its environment, the data flows and threats. Threats cannot be removed but only mitigated. Threats are implemented through specific attacks, each of which can require different attacker profiles. This information flows directly into the risk analysis and is subjected to an evaluation there. When developing secure systems, various factors can be incorporated, such as the potential for harm from specific threats and attacks or economic issues, such as mitigation costs.

A framework for risk analysis and assessments is CORAS [19], which offers a procedure for risk analysis. Based on this, attacks whose damage potential is considered to be the most serious are automatically given the highest priority. Acceptable risks and corresponding assigned tests are neglected in the test procedure. CORAS and threat modelling are compatible techniques, with the graphical modelling offered by CORAS going beyond the model elements of threat modelling and providing a more comprehensive context for the analysis. While the power of threat modelling is to identify potential threats, CORAS provides the full risk analysis framework with the ability to integrate the results of threat modelling.

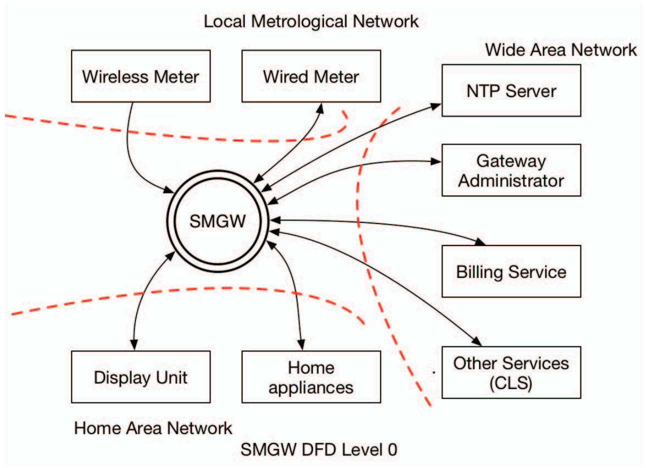

Fig. 5. Real World Example of an Threat Model DFD Level 0

E. *Use of Security Test Pattern*

In the analysis phase, attacks against assets are identified and mitigated by security mechanisms. In this respect, the security-oriented test is a simulation of attacks with the aim of demonstrating the resistance of the security mechanisms against those attacks. At the same time, the aspect of bypassing the defined security functionality becomes more important. For example, a product can implement role-based access control. The security-oriented test then checks the implementation of the role assignment and, in addition, the unauthorized taking of another role or the escalation of privileges assigned to a role in order to achieve an overall assessment of the tested security mechanisms. Testing security functionality utilizes security-related test patterns, which aggregate best practices for these test approaches, comparable to the security patterns of the software design. One such compilation is the MITRE CAPEC[2] library (https://capec.mitre.org), which already offers a larger collection of such test patterns. The CAPEC library attempts to provide a structured compilation of known attack patterns in

[2] Common Attack Pattern Enumeration and Classification

order to make them accessible, among other things, to the security-oriented test.

Element	Description
Summary	Summary of the Attack-Pattern describing the intention of the attack
Attack Prerequisites	Preconditions for the successful implementation of the attack
Typical Severity	Description of the severity of the attack (impact)
Typical Likelihood of Exploit	Description of the probability of successful exploitation of an attack
Solutions and Mitigations	Description of approaches to handling attacks based on this pattern
Attack Motivation-Consequences	Description of scope, technical impact and further remarks
Related Attack Patterns	Relation to other attack patterns

Fig. 6. Elements of CAPEC attack pattern

This compilation provides clues for designing functional tests and performing any penetration tests designed to bypass defined security features.

II. Smart Meter Gateway Security

A. *Characterization of the Smart Meter Gateway*

The Smart Meter Gateway (SMGW) is a decentralized system for the collection of energy consumption data, their storage, tariffing and transmission to a metering point operator. In order to fulfill these tasks, the SMGW provides several physically and logically separated communication interfaces. It defines three network segments called Local Metrological Network (LMN), Home Area Network (HAN), and Wide Area Network (WAN). It allows communication between entities in the WAN and Controllable Local Systems (CLS) in the HAN by a cryptographically secured channel (CLS-channel). The LMN contains the metering systems for measuring energy consumption. The communication between the energy meters and the SMGW is secured by cryptographic mechanisms. The SMGW is multi-client capable and can accept the data of several meters. In addition, it makes a logical assignment of the collected and tariffed data to the users. The result of the tariffing will later be transmitted to the measuring point operator.

There are several classes of devices in the HAN: intelligent home appliances - meaning devices with communication function - and display systems. In the HAN, the roles of the consumer and service technician act. Consumers are the users whose energy consumption data are measured and recorded. They have access to the display unit (display) and can view the measured energy consumption data. The service technician is responsible for commissioning the SMGW and has the authorization to retrieve log information from the system.

There are three different roles in the WAN segment. First, there is the role of the gateway administrator. He is responsible for the correct configuration as well as the safe operation of the SMGW. The administrator may change the configuration of the SMGW and carry out the installation of a firmware update. In addition to the administrator, the role of the meter operator is defined. It provides the interfaces for receiving the measured values recorded in the SMGW for billing purposes. Value-added service providers can use a transparent channel (Controllable Local System channel) to communicate with intelligent systems in the user's network segment (HAN).

B. *Security Requirements Derivation*

The security requirements for the Smart Meter Gateway are formulated in an abstract manner in the SMGW protection profile [20] by utilization of Common Criteria security functional requirements. However, these formulations can only be used with restrictions in software engineering. Deriving the engineering security requirements utilizes the meta model of the security problem definition for identification of assets, security objectives, threats, attacks, and security mechanisms. These elements were analysed on the basis of the protection profile, the technical guideline TR-03109 [21] and potential realization variants of the SMGW.

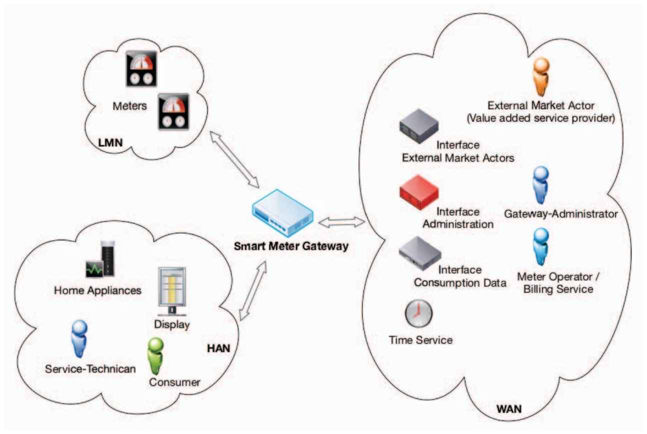
Meters
LMN
Smart Meter Gateway
Home Appliances
Display
Service-Technican
Consumer
HAN
External Market Actor (Value added service provider)
Interface External Market Actors
Interface Administration
Gateway-Administrator
Interface Consumption Data
Meter Operator / Billing Service
Time Service
WAN

Fig. 7. Smart Meter Gateway Infrastructure

Some important findings are listed below:

- Potential inconsistencies in dealing with remote and local attackers. According to the protection profile, remote attackers are given a high potential for attack, while local attackers are assessed moderately on the basis of a presumed lower motivation, leading to low physical protection measures.
- The hardware security module used requires PACE[3]-based authentication. This technical asset is not taken into account in the protection profile and is only addressed by a note in the technical guideline TR-03109. The required resistance reaches at best moderate level.
- User interfaces in the HAN segment that rely on HTML[4] techniques for issuing consumption data are

[3] Password Authenticated Connection Establishment

[4] Hypertext Markup Language

potentially exposed to the full range of network-based attacks.

- The SMGW is based on COTS[5]-hardware. Attacks against hardware microcode - such as Ethernet controllers - must be considered. However, little information is available on commercial hardware products to make an estimate of fault tolerance, robustness, or security.
- The various security domains (WAN, HAN, LMN, CLS) must be supported by separation mechanisms to limit the impact in case of a successful attack (e.g., buffer overflow) and to prevent unauthorized data flow between the segments.

The result of these studies led to the definition of security concepts for the implementation of the Smart Meter Gateway. These security concepts include both the definition of threats, the characterization of potential attackers, concrete attacks and identify specific security mechanisms. Examples of such specific security mechanisms include the use of secure boot mechanisms based on high assurance boot, the use of a hardware security module (HSM), the use of SELinux or the implementation of cryptographic functions. The security concepts are the result of the consistent mapping between assets and security objectives as well as threats and attacks and resulting security mechanisms.

Asset	**Description**	**Technical**	**Socio-Technical**
Consumption Data	Data on the consumption of electrical energy		x
CLS-Data	Data exchanged between an external market actor and a system in the HAN segment.	x	
Authentication data of the user	Data used to prove the identity of a user.		x
Authentication data of the Gateway Administrator	Data used to prove the identity of the Gateway Administrator.		x
Persistent cryptographic keys	Includes all keys permanently stored and used by the SMGW.	x	
Cryptographic session keys	Includes all temporary session keys negotiated by the SMGW.	x	

Fig. 8. Exemplary extract of identified assets

The analysis resulted in nine generic threats to be considered. By way of example, these threats include unauthorized data access, unauthorized data changes, identity misuse, and unauthorized hardware modifications. Based on these threats, specific attacks were identified, which were taken into account when defining the security mechanisms and functional security requirements.

[5] Commercial-off-the-shelf

	Integrity	Confidentiality	Authenticity	Availability
Consumption Data	x	x	x	
CLS-Data	x	x	x	x
Authentication data of the user	x	x	x	
Authentication data of the Gateway Administrator	x	x	x	
Persistent cryptographic keys	x	x		
Cryptographic session keys	x	x		

Fig. 9. Exemplary assignment of security objetives to assets

	Boot Phase	Operation	Data Transmission	Data Storage	Shutdown	Switched Off
Consumption Data	x	x	x	x	x	x
CLS-Data		x	x			
Authentication data of the user	x	x	x	x	x	x
Authentication data of the Gateway Administrator	x	x	x	x	x	x
Persistent cryptographic keys	x	x	x	x	x	x
Cryptographic session keys	x	x	x	x	x	

Fig. 10. Exemplary assignment of assets to scenarios

C. *Threat Model*

For determination of the assets and threats in the security problem modelling, a threat model has been created. After defining the security mechanisms and deriving the functional security requirements, another iteration was performed to identify potential security vulnerabilities through inaccurate specification of the security requirements identified in the first step. This means that the threat model poses a potential security vulnerability when a threat is identified but no mitigation has been defined.

An example of an attack is the manipulation of the firmware on the device. Variants include the acceptance of fake update packages, the change of the firmware in the off state and also attacks that take place at runtime. The attacks or the attack vectors used were identified using attack trees. The information obtained is an input for specific test cases.

Once these gaps have been closed, special aspects - for example, the secure initialization and boot process - were later carried out on the basis of the existing implementation and questioning of the developers involved. This approach has addressed the fact that the development of the Smart Meter Gateway has been done in an iterative process, and in particular the combination of code review and data flow diagram generation has provided granular information that is not provided by less formal approaches such as purely descriptive specifications, As an important effect, this provided the basis for security-related testing of internal mechanisms and vulnerability analysis in the ongoing Common Criteria evaluation.

D. *Security Testing*

The presented case study uses the approaches of risk-based testing with regard to IT security. Based on the security requirements determined, a functional testing is performed. In addition, tests have demonstrated the non-circumvention of the security functionality. The basis for the security-related tests is provided by the security requirements of the protection profile [20], the functional requirements of the technical guideline TR-03109 and the requirements that were obtained in the model-based analysis. The test strategy follows different paths:

- Requirement-based: All involved entities in the environment of the Smart Meter Gateway are simulated. With such a test setup, the functionality is demonstrated in the first step.
- Risk-based: Changes in the simulation environment simulate improper behaviour of the entities involved. Examples include cryptographic mechanisms (using legacy cipher suites in TLS, too small or different key lengths, invalid domain parameters when using elliptic curves), protocol errors (errors in XML-encoded messages, errors in data-simulated counters). Security test patterns are used to demonstrate the resistance of the implementation to known attacks. This procedure should be considered as risk-based for the reason that assumptions were made in the phase of generation of the test suites to be used, which attack patterns appear particularly promising. In addition, typical error patterns from the implementation of security mechanisms should be avoided. In consequence, the assigned tests take place with a higher priority.
- Risk-based / exploratory: Implementation of penetration tests to detect existing security issues, for example in the implementation of the HAN interface. Heuristic tests are used in this context, the results of which are used for further tests. An example of this is the use of the Frankencerts tool [22], which generates a pool of degenerate X.509 certificates and uses them in the course of certificate-based authentication at the HAN interface.

Security testing was used integratively during the development process and in particular showed the maturity of the implementation. Indicators are that the requirement-based tests can be successful at an early stage and the presence of a security feature can be reliably demonstrated. In contrast, it should be noted that the attempt to circumvent the implemented security functionality, even then seems promising if the security feature is already implemented. In this context, threat modelling is an important help, as the potential threats have been targeted and their feasibility has been assessed through one or more attacks based on data flow diagrams and STRIDE analysis.

E. *Results of the Case Study*

The case study yielded the following results:

- The security engineering approach used identified assets and threats which have not been listed in the related protection profile.
- Twenty assets overall were identified. Of these twenty assets, twelve are classified as technical and eight as socio-technical assets.
- Six different scenarios with different involvement of the assets were identified.
- Nine different threats have been identified.
- The attacker model used in the protection profile and the technical policy are inconsistent. Key motivational factors to carry out attacks by the local attacker were not taken into account. The newly defined security problem model gives local attackers more motivation and appears more realistic in this regard.
- 113 attacks against the external interfaces of the system were checked for their relevance and considered in security testing.
- Physical manipulation attempts can only be warded off to a very limited extent.

To sum up, the Smart Meter Gateway is a system with weak physical protection, which is limited to passive detection of hardware integrity violations. The software-based security mechanisms are suitable for the defence against attackers with high attack potential. Through the network interfaces, an SMGW can be considered to be a secure system. For systems of this class of equipment, there are significant problems, especially in the context of economic efficiency with regard to physical protection. The desire to place secure communication systems in the low-cost sector leads to a systematic renunciation of physical protection measures such as passive encapsulation or active intrusion detection. By contrast, the implementation of attacks that require physical access to the system becomes easier, so that the attacker model and the respective granted motivations and capabilities have to be reconsidered at this point.

III. Conclusion

Between software development and security engineering, the security tests should be understood as an independent activity within the security-engineering methodologies. The objective is checking the security functionality with the purpose of demonstrating their correct implementation and non-bypassibility. Test models are formulated along functional security requirements. Clearer requirements lead to more precise test models, generated test suites and test cases. Security tests can address security requirements precisely only if the definition of the security problem is precise itself. In this paper, a precise model for the definition of the security problem and its use for obtaining a test model was shown. The presented model provides concepts for assets, security mechanisms, threats, attacks and attackers. Functional security requirements are derived and form the basis for the requirement-driven security tests. In addition, however, risk-based approaches are necessary because security testing cannot just be limited to requirements but must take into account the security of the system in terms of the unavoidability of defined security mechanisms and the resistance to specific attacks. These must be prioritized based on the characterization of the attackers. Security testing needs to weigh which attacks are more likely to be exploited and successful. The ETSI framework for risk-based security testing defines a procedural model, and CORAS provides the risk analysis framework. An artefact that can be used in this context are the results of a threat modelling. In the analysis and architecture phase, threat modelling is indispensable to detect existing threats. Threat modelling provides concrete information regarding the data flow at relevant interfaces as an input to security testing.

The presented security modelling and testing approached was implemented in connection with the development of a Smart Meter Gateway (SMGW). It generated relevant results with regard to existing requirements from the protection profile and technical guideline and showed security requirements that were not recognized before. The detailed handling of the attacker model resulted in additional security measures that go beyond the original security requirements. It also became clear that the low physical protection of devices intended for the market does not mitigate motivated attackers.

References

[1] A. Lunkeit: „Modellbasiertes Security-Engineering in der Softwareentwicklung“ (in German), PhD Thesis, TU Berlin, Jan. 2018..

[2] I. Schieferdecker, J. Grossmann, M. Schneider, “Model-based security testing”. arXiv preprint arXiv:1202.6118 (2012).

[3] M. Felderer, B. Agreiter, P. Zech, R. Breu,”A classification for model-based security testing.”, in Advances in System Testing and Validation Lifecycle (VALID 2011) 2011, pp. 109–114.

[4] Common Criteria Management Board (CCMB), “Common Criteria for Information Technlogy Security Evaluation, Part 1: Intro- duction and general model”, CCMB-2012-09-001, Version 3.1 Revision 4. 2012.

[5] S. Sprenkle, S. Sampath, E. Gibson, L. Pollock, A. Souter, “An empirical comparison of test suite reduction techniques for user-session-based testing of web applications” in: 21st IEEE International Conference on Software Maintenance (ICSM’05) IEEE, 2005, pp. 587–596.

[6] M. Utting, B. Legeard, “Practical Model-Based Testing” Elsevier LTD, Oxford, 2007, ISBN 978–0–12–372501–1.

[7] Y. Cheon, G. T. Leavens, “ A simple and practical approach to unit testing, The JML and JUnit way”, in: European Conference on Object-Oriented Programming Springer, 2002, pp. 231–255.

[8] H. Zhang, X. Zhang, M. Gu, "Predicting defective software components from code complexity measures" in Dependable Computing, 2007. PRDC 2007. 13th Pacific Rim International Symposium on IEEE, 2007, pp. 93–96.

[9] Y. Shin, L. Williams, "An empirical model to predict security vulnerabilities using code complexity metrics" in Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement ACM, 2008, pp. 315–317.

[10] J. C. King, "Symbolic execution and program testing" in Communications of the ACM 19, 1976, Nr. 7, pp. 385–394.

[11] C. Cadar, D. Dunbar, D. R. Engler, "KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs" in OSDI Bd. 8, 2008, pp. 209–224.

[12] P. Godefroid, "Random testing for security: blackbox vs. whitebox fuzzing" in Proceedings of the 2nd international workshop on Random testing: co-located with the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2007) ACM, 2007.

[13] P. Godefroid, M. Levin, D. Molnar, "SAGE: white-box fuzzing for security testing" in Queue 10, 2012, Nr. 1, pp. 20-27.

[14] A. Takanen, J. D. Demott, C. Miller, “Fuzzing for software security testing and quality assurance”, Artech House, 2008.

[15] J. P. McDermott, "Attack net penetration testing" in Proceedings of the 2000 workshop on New security paradigms ACM, 2001, pp. 15–21.

[16] B. Arkin, S. Stender, G. McGraw,"Software penetration testing" in IEEE Security & Privacy 3, 2005, Nr. 1, pp. 84–87.

[17] G. F. Lyon, “Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning” USA : Insecure, 2009. – ISBN 0979958717.

[18] ETSI, “ETSI EG 203 251, Methods for Testing & Specification Risk-based Security Assessment and Testing Methodologies v1.1.1”, 2015.

[19] R. Fredriksen, M. Kristiansen, B. A. Gran, K. Stølen, T. A. Opperud, T. Dimitrakos, "The CORAS framework for a model-based risk management process" in International Conference on Computer Safety, Reliability, and Security Springer, 2002, pp. 94–105.

[20] Bundesamt für Sicherheit in der Informationstechnik (BSI) „Protection Profile for the Gateway of a Smart Metering System (Smart Meter Gateway PP) Schutzprofil für die Kommunikationseinheit eines intelligenten Messsystems für Stoff- und Energiemengen Version 1.3“, 2014.

[21] Bundesamt für Sicherheit in der Informationstechnik (BSI) „Technische Richtlinie BSI TR-03109-1 Anforderungen an die Interoperabilität der Kommunikationseinheit eines intelligenten Messsystems“, Version 1.0. 2013.

[22] C. Brubaker, S. Jana, B. Ray, S. Khurshid, V. Shmatikov, "Using Frankencerts for automated adversarial testing of certificate validation in SSL/TLS implementations" in Security and Privacy (SP), 2014 IEEE Symposium on (2014), pp. 114–129.

[23] A. Lunkeit, T. Voß, H. Pohl "Threat Modeling Smart Metering Gateways" in Smart Objects, Systems and Technologies (Smart-SysTech), Proceedings of 2013 European Conference on VDE, 2013

[24] C. A. Ericson, „Fault tree analysis. Hazard analysis techniques for system safety”, 2005, 183-221.

[25] J. D. Andrews, S. J. Dunnett. "Event-tree analysis using binary decision diagrams." *IEEE Transactions on Reliability* 49.2, 2000, pp. 230-238.

[26] Shostack, Adam: “threat modeling: designing for security”. John Wiley & Sons Inc., 2014. - ISBN 978-1-118-80999-0

[27] J. Jürjens, “Secure systems development with UML”, Springer Science & Business Media, 2005.

[28] J. Jürjens "UMLsec: Extending UML for secure systems development." International Conference on The Unified Modeling Language. Springer, Berlin, Heidelberg, 2002