Security Requirement Modeling for a Secure Energy Trading Platform

Yasamin Mahmoodi*, Christoph Groß*, Sebastian Reiter*, Alexander Viehl*, Oliver Bringmann†

*FZI Forschungszentrum Informatik
Haid-und-Neu-Str. 10-14
D-76131 Karlsruhe, Germany
email: [mahmoodi, cgross, sreiter, viehl]@fzi.de

†Universität Tübingen
Sand 13
D-72076 Tübingen, Germany
bringman@informatik.uni-tuebingen.de

***Abstract*—The Internet of Things (IoT) paradigm has become important in many domains, ranging from smart home to medical and industrial applications. However, besides the outstanding advantages, comprehensive networking raises new security challenges. To benefit from IoT, secure embedded systems and resilient architectures are mandatory. Security-by-design is a cost efficient approach to accomplish this objective. Security requirement analysis as the first step of security-by-design plays an important role to design, develop and test secure embedded systems. This paper presents a case study to demonstrate security requirement modeling at three abstraction levels with the focus on the CIA triad (Confidentiality, Integrity, Availability). The methodology is demonstrated by applying the proposed approach to a use case from the energy domain.**

***Keywords: Security analysis; IoT; Security requirement; Security-by-design*.**

I. INTRODUCTION

The Internet of Things (IoT) evolved into a mature technology and is nowadays used in a wide variety of application domains. Besides the numerous advantages offered by connected embedded systems, new security challenges and threats have been reported over the last years [1] [2]. The underlying embedded systems store sensitive information, such as financial data, medical data and passwords. Therefore, security issues are a critical concern.

The introduction of the IoT paradigm into the energy market, offers the opportunity to restyle the centralized energy market. This offers the opportunity to substitute centralized main energy producers with distributed decentralized small-scale energy providers. Households can install a photovoltaic system on the roof to produce their own energy and sell the rest to their neighbors. One efficient approach to sell the extra energy, which is beneficial both for seller and buyer, is the utilization of the IoT paradigm to create an automated local energy trading market. EnerDAG [3] is a platform, which provides local energy trading among neighbors employing a tangle data structure and smart contracts.

Decentralized systems are based on autonomous systems that operate on local information and work together to realize a complex task. Each system can therefore change specific information, issue non-expected transactions or try to compromise the system by other malicious behavior. Authorized nodes may try to manipulate the system in order to get financial benefits or to prevent other nodes from trading energy.

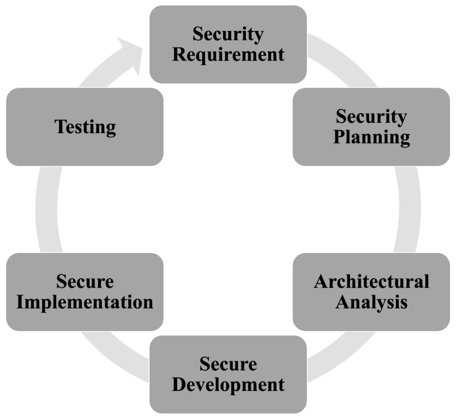

Figure 1. Secure System Development Life Cycle (SSDLC)

On the other hand, unauthorized nodes, which penetrate the system, may be able to keep nodes off the network or to use network data in order to make a Denial of Service (DoS) attack to make services non-available. Another aspect that should be considered is the anonymity of the participants. If the neighbors have access to the energy balance of other nodes, malicious neighbors can determine when people left for a vacation to break into their house. Accordingly, to get the full benefit of the comprehensively networked systems, considering security measures along the system design is inevitable.

To consider vulnerability assessment and penetration testing only on the final embedded system - security after the fact - is not an effective approach and the elimination of weak points could cost a lot of time and money. Integrating security aspects in all phases of system design - security by design - is a promising approach that lets system designers consider security from the requirements phase over the development phase to the final integration phase. The Secure System Development Life Cycle (SSDLC) [4] defines tasks, such as the definition of security requirements and assessing their risks, the planing of the security architecture, the actual design and implementation as well as task regarding testing and security assessment. An exemplary SSDLC is shown in Figure 1.

Security requirements are the foundation to design a secure system and execute security tests for each phase of the system design [5]. The better the security requirements are defined, categorized and refined, the more efficient tests can be designed and performed. Security requirements may come from different sources in various formats and abstraction levels. Managing these requirements is difficult, especially considering the fact that they may change during the system development life cycle [6].

This paper proposes three layers of abstraction for security requirements of embedded systems, which starts with general questions about security considerations at the most abstract layer and continues to explain them in more detail and finally puts them into several defined categories and deploys them to system entities. The third, most concrete security information is applied to an Unified Modeling Language (UML) model of the system with several predefined stereotypes. This procedure helps to guide the user, to keep track of the security requirements and offers the possibility to integrate them into the actual design environment. By using a well-defined interface, even an automated processing of security requirement information is possible. The paper is structured as follows: Section II-A highlights the importance of security requirement analysis in designing secure embedded systems. It mentions the three main principles that guide the information security modeling. Section III introduces the approach we applied for security requirement analysis and modeling of embedded systems. The following Section IV demonstrates the application of the methodology on enerDAG, a framework for energy trading.

II. V-MODEL FOR DESIGNING SECURE SYSTEMS

The SSDLC can be mapped to the traditional V-model [7], which provides a testing phase associated with each development phase. In Figure 2, the blue diagram represents the traditional V-model, with the system development on the right side. It starts with requirements, then it goes into the design of the system, the architecture and finally each module. Each step on the left side is directly associated with a testing phase on the right side. The V-model proposes a hierarchical perspective, which lets the designer start with a very abstract system specification and gradually add more design details. By mapping the SSDLC to the V-model design flow, the iterative nature of the SSDLC should be applied to the traditional V-model. Resulting in a repeated adjustment and refinement of the system and a repeated execution of the overall V-model.

Integrating security features and the associated validation to the overall system development process helps to discover and reduce vulnerabilities and design flaws at early stages, hence saves time and reduces costs. In Figure 2, the gray diagram depicts the SSDLC inspired, security-based V-model alongside the traditional V-model. At the right side of the diagram, security requirements alongside the other requirements of the system enable the security experts to get a comprehensive system view. As system designers go further in the development process, e.g., deriving the security architecture, more details both with regard to the design as well as to the test phases can be added. In this process, the security requirements should be refined too, and associated with components of the system design. Today's system design processes often apply a set of models, based on the UML, to specify the system

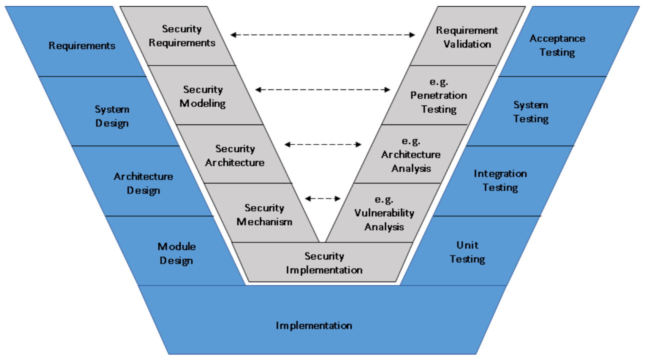

Figure 2. Security-Based V-Model

architecture as well as software features. For that reason, we propose a UML model augmented with security features as a security profile [8]. This enables the designer to refine security requirements in the same development environment as the system design. Our proposed security modeling approach provides models for security requirements and helps to discover the inherent weak points of the system architecture or chosen implementations by specifying the protection goals of the system, potential attack points, as well as additional security related documentation. Furthermore, by providing a well-defined profile the users are guided in specifying attack scenarios as well as protection goals. The UML security profile can be attached to the UML models to entitle the information about protection goals and assets, weak points and attack surfaces, as well as documentation-based information such as security mechanism and software version. These models can later on be beneficial for validation, e.g., with static analysis and vulnerability assessment. In addition, the model simplifies manual analysis, such as penetration testing, by boosting the documentation and helps identifying underlying potential design flaws, e.g., by enabling an automatic lookup in well-known security vulnerability databases.

A. *Security Requirements of Embedded Systems*

The requirement specification is the entry point of a system development process. It specifies the goals, functions and constrains of the system, and the relationship of them [9]. Requirement engineers should communicate frequently with stakeholders, system designers, developers and system analysts. A precise requirement description of the system is the basis to ensure stakeholders interests and manage the development process and budget. Bringing up security aspect into the system development process necessitates the need to integrate security requirement analyses in the first phases of the development process. Security requirement analysis is an essential step in today's system development processes, which should become the standard between stakeholder's requirement validation, development and testing. Defining security requirement categories and standards, which are pragmatic for both developers and testers, play a fundamental role. The Federal Information Processing Standard (FIPS) (The National Institute of Standards and Technology (NIST), 2010)

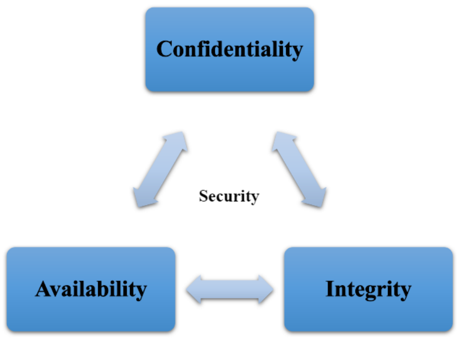

Figure 3. CIA Triad

offered three security core principles that guide the information security area:

- Confidentiality: ensures that access to the critical data is available only for authorized users.
- Integrity: assures the correctness and completeness of the data over its entire life cycle.
- Availability: makes sure that data and services are available for authorized users.

The CIA triad is shown in Figure 3.

Breaking any of the triad security principles (CIA) may lead to a level of impact, which endanger resources, information or individuals. The impact severity is define with the following three levels:

- Low: leads to limited adverse effect.
- Medium: generates serious or critical damage effect.
- High: occurs severe or catastrophic damage effect.

A security requirement modeling, which covers the CIA triad can offer a desirable standard for security requirement analysis.

III. Proposed Approach

The following section explains the proposed approach for a consistent, guided security requirements modeling. Considering the importance of security requirement analysis in designing secure embedded systems, we provide a three-level requirement modeling, which follows the sequential abstraction layers principle of the V-model. It starts with general security aspects of the system and concludes with a detailed specification of protection goals assigned to single entities in the system. The revealed information in last level is advantageous for architectural analysis as well as penetration testing. Because this last level should ensure a seamless transition into the system development process, we applied it to a traditional UML modeling approach by proposing a security profile.

The first abstraction level brings up general questions about security issues regarding the system. The stakeholders discuss about their desire system with the system designers and security experts to make a security checklist. Here is a list of the most important questions, which should be answered during security requirement analysis:

- What are the important parts of the system, which should be protected from attackers?
- Who are the potential attackers?
- Which parts are potential entry points for attackers?
- What are the effects of potential attacks on the critical data?
- Which security measures are needed?
- What is the network topology of the system and which security mechanisms will be applied to the system?
- How secure are the chosen hardware, software and technologies?
- What is required security level for the system?
- How are software updates handled?

The answers to these questions give an overall perspective about the security aspects of the system. However, this information is qualitative and subjective and in order to apply this knowledge, well-defined classifications and security metrics are needed.

In the second abstraction level, the protection goals of the system will be derived and categorized in the three classes of the CIA triad: confidentiality, integrity, availability. From this informal specification, all relevant information will be extracted in the next step.

The last abstraction level maps the information from the second layer about security requirements to a well-defined schema. The layer defines entries such as protection goal confidentiality, protection goal integrity and protection goal availability, and offers parameters to assign more details to each entry. No security mechanism by itself can protect the system completely and layered security defenses based on the nature of the protected information and weaknesses of the system are required. Untangling this issue, security experts and system designers require detailed information about the entities of the system, which should be protected, and the potential vulnerabilities and entry points. Therefore, this layer not only specifies the protection goals as well as the attack surface in a well-defined manner, it also associates each specified security entry with a system component or a sub-system. The specified data of this level should be used to decide about the design and integration of security mechanism as well as the specification of security tests. Several approaches such as [10] are already using the CIA classification to provided security mechanisms dedicated to each class and therefore integrate seamlessly into the proposed procedure.

To structure the information on the third layer in a well-defined manner, we propose to specify detailed information about protection goals, attack surfaces and documentation-based information in form of a security profile for the UML. A detailed description of the profile is present in [8]. Our security profile proposes stereotypes, which can be attached to UML modeling elements and are advantageous for both documentation and threat modeling. For security requirement modeling, we employ the protection goal category offered in our security profile. The protection goal category proposes the stereotype ≪PG.DataConfidential≫ (PG.C), which defines that the associated property should be protected against reading or predicting by an attacker. This stereotype implies confidentiality from the CIA triad. A parameter for the severity level can be added, specifying the criticality of the potential damage, if the protection goal is violated. The second stereotype of

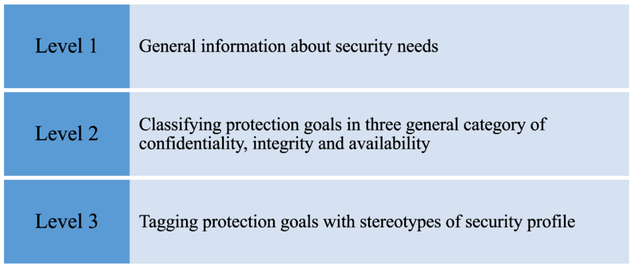
Level 1
General information about security needs
Level 2
Classifying protection goals in three general category of confidentiality, integrity and availability
Level 3
Tagging protection goals with stereotypes of security profile

Figure 4. Three Abstraction Level Of Proposed Security Analysis Model

this category, ≪PG.DataModify≫ (PG.M) indicates that the property should be protected against modification. A severity level also mentions how severe a violation of the goal is. ≪PG.DataDelete≫ (PG.D) characterizes the data that should not be deleted by an unauthorized transaction. Also a severity level was added. The stereotype ≪PG.DataAdd≫ (PG.A) indicates with its severity that this property should be protected against adding new data by an attacker. The stereotypes PG.C, PG.A and PG.D refer to the integrity of the CIA triad. The last stereotype ≪PG.ServiceAvailability≫ (PG.Ava) indicates that the stereotyped operation must be available and should not be stopped. Availability from the CIA triad is mingled with this stereotype. Figure 4 illustrates the three abstraction levels of security requirement modeling for embedded systems.

IV. Use Case

As a use case to demonstrate the security requirement analysis and modeling, we consider enerDAG (energy Directed Acyclic Graph) a local energy trading platform offered [3]. enerDAG offers an expandable platform for households to trade energy with their neighbors efficiently and securely using the tangle directed acyclic graph data structure. It is a highly distributed computing system, which applies smart contracts and majority voting for energy trading. Every household has a smart meter to measure the energy balance and a enerDAG software to communicate with other nodes and operate the contract. Each node in the neighborhood market may have a positive energy balance as energy producer or prosumer. Prosumers are households that produce energy, e.g., with photovoltaic, and also consume energy. Similarly each node can have a negative energy balance for consumer households or prosumers, if they consume more energy than they produce.

Energy trading will take place in five-minutes intervals. At the beginning of each time frame, the nodes will send their energy balance and their proposed maximum selling or minimum buying price to the network. Then, they receive the same offers from other nodes and based on an algorithm each node calculates the trade results of this contract for this time period. The contract results will then be sent to the tangle data structure and based on a majority voting will be part of the tangle. For security reasons, all the transactions are encrypted with a public encryption key of the neighborhood market as well as the private encryption key of the individual nodes.

A maintainer node registers the nodes, assembles the neighborhood and manages the market if it is necessary, as well as calculates the amount of traded energy and issues bills. Figure 5 represents a model of a neighborhood with different types of nodes.

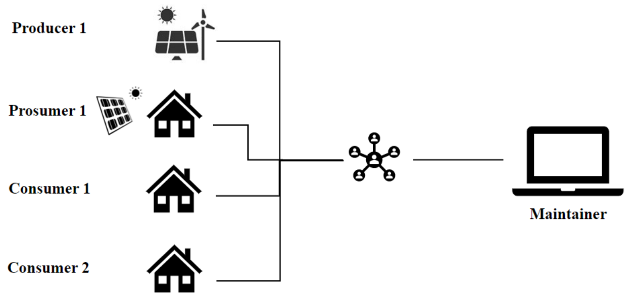

Figure 5. Model Of A Neighborhood With Different Types Of Nodes

enerDAG applies the tangle data structure, which is a directed acyclic graph that consists of transactions and their references. Each node can add transactions to the tangle and receive transactions from other nodes over the gossip protocol and insert the transactions in its view of the Tangle structure. In order to insert transactions in the tangle, the nodes have to follow the rules, e.g., by referencing at least two older transactions called tips and then publishing it to its neighbors. The enerDAG daemon, which is installed on each node, first establishes a database connection before running the main async loop forever that includes the following parts:

- *contractEngine()*: This part runs every minute and search for available contracts for execution. If it determines that a contract needs to be run, it loads and executes it via the *contactExecuter()* function. The result of a contract will be sent to the node itself for transaction handling.
- *connectionEngine()*: This part starts a server that will listen on a port for incoming messages via the *handleIncomingEvent()* function. When a transaction is received, it is processed and the correct action is taken.

A. *Security requirement analysis of enerDAG*

The following section highlights the proposed method applied to a security requirement analysis of enerDAG. Especially the three abstraction levels will be motivated.

1) Level one: The first abstraction level summarizes the stockholder's demand or general security needs in an informal way. Exemplary security demands of enerDAG are:

- Secure energy trading.
- Transparent transactions.
- Anonymity of the participants should be ensured.
- Non repudiation.
- The amount of energy that each node produces or consumes should be secret.
- The offered price from nodes should remain secret.
- Unauthorized user should not be able to participate in the market.
- Authorized users should not be able to cheat.
- A potential attacker may be both an unauthorized or an authorized user.

2) Level two: In the following abstraction level, the security requirements will be categorized into the three classes of the CIA triad (Confidentiality, Integrity and Availability).

- Confidentiality:
  - Energy balances of the participants.
  - Offered prices of participants.
  - The bids offered by participants.
  - Private key of the household nodes.
  - Public key of the neighborhood.
  - The transactions.
  - The seed sent by the maintainer to the household nodes.
  - List of neighbors.
- Integrity:
  - Energy balances of the participants.
  - Offered prices of participants.
  - The bids offered by participants.
  - Contract execution.
  - Majority voting.
  - The tips (least two older transactions in tangle data structure).
  - List of neighbors.
- Availability:
  - The server listening for new transaction
  - Transaction handling
  - Contract execution

3) Level three: In the last level of abstraction, we go through the software, respectively the corresponding UML model, and tag parts of the system with our proposed stereotypes, e.g., the five stereotypes to specify the protection goals of the system. This information extends the traditional UML model with security features and guides the architectural analysis and penetration testing later on. Additionally it is a good starting point for the identification of the required security mechanisms to achieve the protection goals.

The enerDAG software running on household nodes comprises of several functions, e.g., for receiving the list of neighbors from the maintainer node, handling the incoming transactions, sending the bids to the market, executing the contract and adding transaction to the tangle data structure. The maintainer node offers functions such as setting up a new node, assembling the neighborhood, sending the list of neighbors and calculating the bills for participants. With regard to the space limitations, it is not possible to explain all of the functions in detail. Therefore, we picked a few of the functions to demonstrate the utilization of the last abstraction level.

In household nodes, each time frame is five minutes. The first event in each time frame is to check if the result of the previous market execution has already been posted to the tangle and if so, to receive and to save it into the state file of the contract. Then the energy balance and the proposed price will be extracted and the bid will be structured. Afterwards, the bid will be encrypted first with the private key of the node then with the public key of the neighborhood and will be inserted into a transaction. The transaction is sent to the tangle using the neighborhood market contract's address as the receiver address. Meanwhile each node also receives the bids from the other nodes, decrypts the outer layer and puts them into the contract folder. In a next phase, each node will send its decryption key to the participating nodes and each node will decrypt the corresponding bid upon receiving this key. These two phases happen inside a 5 minute frame. The contract engine will then execute the smart contract at the end of the five minute time frame and send the results to the tangle data structure. Here we focus on the contract engine to illustrate security requirements and protection goals.

The contract engine searches through available contracts and if it finds a contract ready for execution, it provides it to the *contractExecutor()* function where the result of the contract execution is gathered, packed into a message of type *contractResult* and sent to the node itself. Then based on a majority voting the contract results will be accepted as the final results.

For the neighborhood smart contract, the *contractExecutor()* starts a loop and searches in all the contracts and find the negative energy balance (as buyers) and positive energy balance (as sellers). At the next step, it matches the best seller, sellers with lowest price, with the best buyers, buyer with highest prices, and then conduct the trade between them with the average price of both. The loop will continue until there is no energy to either sell or buy left. The list of assets and protection goals in the *Contractengine()* are represented underneath:

- *contractExecutor()* as a service should be available.
  - Asset: PG.Ava
  - Severity: Medium
  - Offered Security mechanisms: security policy (restricting sending message), IDS, firewall.
- Contract folder
  - Asset: PG.M, PG.A, PG.D
  - Severity: Low
  - Offered security mechanisms: Verifying integrity of the data using HMAC (Hash Message Authentication Code), AAA (Authorization, Authentication, Accounting) and to prevent hackers to be able to modify contract folder
- Majority
  - Asset: PG.M, PG.A
  - Severity: Medium
  - Offered security mechanisms: Encryption, hashing, verifying integrity of the data using HMAC, AAA
- Minimum selling/ maximum buying price in database
  - Asset: PG.C, PG.M, PG.A, PG.D
  - Severity: Medium
  - Offered security mechanisms:Pproper separation of Database, encryption, hashing, verifying integrity of the data using HMAC, AAA
- Validation key
  - Asset: PG.C, PG.M, PG.A, PG.D
  - Severity: High
  - Offered security mechanisms: Proper separation of Database, encryption, hashing, verifying integrity of the data using HMAC, Key management AAA
- Bid

  - Asset: PG.C, PG.M, PG.A
  - Severity: Medium
  - Offered security mechanisms: Encryption, hashing, verifying integrity of the data using HMAC, AAA

The decentralized system of enerDAG needs to validate the nodes, which are allowed to participate in the different neighborhood markets. Therefore, the energy and infrastructure providers run a node within each neighborhood that is called the neighborhood maintainer node. The maintainer sends hashed *Validation Keys* to the neighborhood nodes, sends the neighborhood list to the participants, sets up new nodes and calculate the amount of traded energy and publishes the bills. Here we focus on the function of adding a new node to show our security requirement analysis in level three.

To add a new node, the maintainer should change the complete neighbor structure of all nodes randomly in order to prevent malicious activities, e.g., by a cluster of bad nodes. The algorithm here takes each node from the database and tries to randomly assign nodes to the neighborhoods that do not have enough neighbors already. The limit is set to five neighbor nodes to allow for nice majority voting while not flooding the network with messages between nodes. The neighborhood maintainer node then sends the new neighbor list to all the nodes. At the next step the neighborhood *Validation Seed* will be generated for the new node and together with other neighborhood information, like the neighborhood encryption key, neighborhood maintainer node address and neighborhood contract address, will be sent to the node. The protection goals and assets of adding new node function are provided below.

- Neighborhood list
  - Asset: PG.M, PG.A
  - Severity: Medium
  - Offered security mechanisms: Hashing, verifying integrity of the data using HMAC, AAA
- *Validation seed*
  - Asset: PG.C, PG.A, PG.M
  - Severity: High
  - Offered security mechanisms: Encryption, hashing, verifying integrity of the data using HMAC, AAA, key management
- Neighborhood cryptography
  - Asset: PG.C, PG.M, PG.A
  - Severity: High
  - Offered security mechanisms: Encryption, hashing, verifying integrity of the data using HMAC, AAA and to prevent hackers to be able to modify contract folder
- Contract address
  - Asset: PG.C, PG.M, PG.A
  - Severity: Medium
  - Offered security mechanisms: Encryption, hashing, verifying integrity of the data using HMAC, AAA

V. Conclusion

Considering the importance of security requirement analysis in designing secure embedded systems, we proposed three abstraction levels for security requirement modeling in this paper, to enable a guided security consideration. The first abstraction level answers general question about security needs, considering stakeholders demands. In the second abstraction level, protection goals will be distinguished based on the information derived from the first level. Then the protection goals will be categorized in three classes, for confidentiality, integrity and availability. The last abstraction level goes into more details and classifies the exact assets of the system in five categories and protection goals: data confidentiality, data modification, data deletion, data addition and service availability. To foster the usage, we integrated the information in a classic UML based design flow, by providing a UML profile with dedicated stereotypes to specify the information. The applicability of the approach is demonstrated by modeling and transformation of security requirements for a energy trading platform (enerDAG) that enables households to create localized energy markets. We picked exemplary security requirements, modeled them on the three abstraction levels and showed the consistency and guided workflow to generate detailed protection goals and attack vectors in the use case.

Acknowledgement

This work has been partially supported by the Federal Ministry of Education and Research (BMBF) within the project COMPACT (grant number 01|S17028C).

References

[1] J. Viega and H. Thompson, “The state of embeddeddevice security (spoiler alert: It’s bad),” Security Privacy, IEEE, October 2012, pp. 68–70.

[2] B. Schneier, “The internet of things’ dangerous future,” Jan 2017 (accessed October 2020), https://www.schneier.com/blog/archives/2017/02/security_and_th.html.

[3] C. Groß, M. Schwed, S. Mueller, and O. Bringmann, “enerdag – towards a dlt-based local energy trading platform,” in 2020 International Conference on Omni-layer Intelligent Systems (COINS). IEEE, Aug 2020, p. 1–8.

[4] R. Chopra and S. Madan, “Security During Secure Software Development Life Cycle (SSDLC),” International Journal of Engineering Technology Management and Applied Sciences, vol. 3, 2015, pp. 1–4.

[5] ISO/IEC 15408-3, “Evaluation criteria for IT security – Part 3: Security assurance components,” ISO, Tech. Rep., 2009.

[6] D. Mellado, E. Fernández-Medina, and M. Piattini, “SREPPLine: Towards a Security Requirements Engineering Process for Software Product Lines ,” Security in Information Systems, Proceedings of the 5th International Workshop on Security in Information Systems, vol. 125, 2007, pp. 220–232.

[7] K. Forsberg and H. Mooz, “The relationship of system engineering to the project cycle.” NCOSE, Chattanooga, Tennessee, 1991.

[8] Y. Mahmoodi, S. Reiter, A. Viehl, O. Bringmann, and W. Rosenstiel, “Model-guided security analysis of interconnected embedded systems,” 6th International Conference on Model-Driven Engineering and Software Development, 2018, pp. 602–609.

[9] S. R. Kourla, E. Putti, and M. Maleki, “Importance of Process Mining for Big Data Requirements Engineering,” International Journal of Computer Science & Information Technology (IJCSIT), vol. 12, August 2020.

[10] “A Survey of Information Security Implementations for Embedded Systems,” 2017 (accessed October 2020), URL: https://www.windriver.com/whitepapers/.