Evolving Attacker Perspectives for Secure Embedded System Design

Letitia W. Li[1,2], Florian Lugou[1] and Ludovic Apvrille[1]
[1]*LTCI, Télécom ParisTech, Université Paris-Saclay, 75013, Paris, France*
[2]*Institut VEDECOM, 77 rue des Chantiers, Versailles, France*

Keywords: Model-Driven Engineering, Security, Formal Verification.

Abstract: In our increasingly connected world, security is a growing concern for embedded systems. A systematic design and verification methodology could help detect vulnerabilities before mass production. While Attack Trees help a designer consider the attacks a system will face during a preliminary analysis phase, they can be further integrated into the design phases. We demonstrate that explicitly modeling attacker actions within a system model helps us to evaluate its impact and possible countermeasures. This paper describes how we evolved the SysML-Sec Methodology with "Attacker Scenarios" for the improved design of secure embedded systems.

1 INTRODUCTION

The advent of connected and soon autonomous vehicles is projected to ease traffic congestion, improve safety, and free us from tedious daily commutes. However, even with the limited connectivity of today, connected cars have already been compromised in various hacks (Miller and Valasek, 2015; Constantin, 2016).

The methodology of our lab, SysML-Sec (Apvrille and Roudier, 2015), describes a design process for safe and secure embedded systems. However, during our study of the security of connected vehicles, we found that we lacked the ability to precisely model certain published attacks, and thus to consequently evaluate countermeasures against them. While attack steps can be modeled in Attack Trees, the description of each attack is usually limited to a few words, thus limiting their usage to documentation and not to formal evaluation.

Therefore, this paper introduces a new concept called "Attacker Scenarios", which precisely model attacker actions within a system design, therefore using the same operators as for system design. Thus, attacker scenarios are intended to interact with (and possibly control) system models with communications via signals. The paper explains the modeling capacities of attack scenarios — e.g., how to model message injection in communication channels — as well as how they integrate into our methodology. In section 2, we discuss the high-level approach of using attack scenarios in system design, which we elaborate in section 3 to be specific for embedded systems. We then present the main phases of our methodology in greater detail, and how attacker scenarios can be efficiently used within these phases: Requirements/Analysis in section 4, HW/SW Partitioning in section 5, and Software Design in section 6. Next, we present the related work in Section 7. Finally, Section 8 concludes the paper.

2 INTEGRATING ATTACKER MODELS WITH SYSTEM DESIGN

1. The process starts with considering the needs and threats of the system, which are expressed respectively in **Requirements and Attack Tree Models**. Attack Trees may impact the Security Requirements – for instance, to handle a threat, a new requirement is added – , so these may be designed together. Each Attack Tree is a diagram gradually refining the different steps necessary to realize a single root attack. Regardless of which individual attack steps are possible for the attacker to perform, our ultimate concern is to ensure the root attacks cannot be performed.
2. Next, the **System is Designed**. The functionality of system components is modeled, along with attacker scenarios which describe the attacker's interactions with the system. Attacker Scenarios can be either pre-built attacker models (Dolev-

Li, L., Lugou, F. and Apvrille, L.
Evolving Attacker Perspectives for Secure Embedded System Design.
DOI: 10.5220/0006535802870294
In *Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2018)*, pages 287-294
ISBN: 978-989-758-283-7

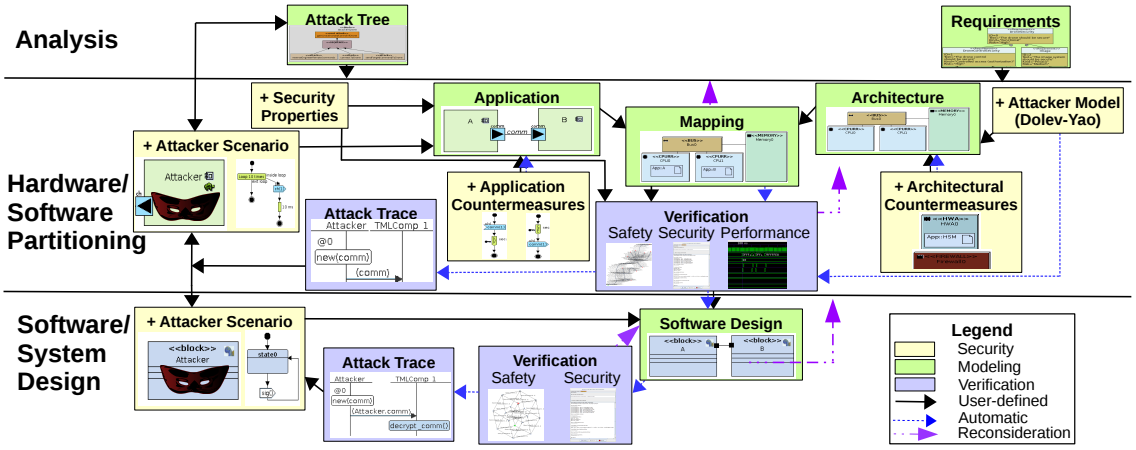

Figure 1: Design Methodology.

Yao) or a custom-generated attack behavior.

3. Based on the Requirements of Step 1, **Security Properties** are modeled with regards to the system design. Security properties describe if a system or data should be secure against a specific type of attack. For example, if a security requirement states that all communications shall be confidential (unrecoverable by the attacker), then one security property of confidentiality is added for each communication exchange.
4. **Formal Verification**: The security properties described in Step 3 are verified on the system model (step 2) with a security prover. The results of verification are either:
   (a) All Security Properties are Satisfied, and no root attacks can be performed. The system, if implemented as specified, should be secure against these conceived attacks in this form and fulfill the requirements. (If new attacks arise afterwards or the system model needs to be modified, then the process must be repeated.)
   (b) At Least One Security Property is Proven False. If the prover can output an execution trace demonstrating the property violation, then the Attack Trees of Step 1 can be enhanced with this information. Then, to secure the system, either we must reconsider (i) the Requirements of Step 1 (if the security requirements are too strict and unrealistic), (ii) reconsider the security mechanisms used to secure the model, or (iii) reconsider the attacker model (An attacker is generally assumed not to be willing to spend a cost more than the value of the information recovered/system damaged, and we can reconsider it if we have assumed he/she has more resources than realistic.) After modifications, we re-iterate through the process until our system is verified secure.

Using this framework for integrating attacker behavior into system models, we next present our detailed SysML-based methodology specialized for embedded systems.

3 MODELING ATTACK FRAMEWORKS FOR EMBEDDED SYSTEMS

Embedded systems have faced a growing number of attacks in recent years (Newman, 2016; Larson, 2017). Combining the ideas in the previous section with the 3 phases of embedded system design in SysML-sec (Analysis, Hardware Software Partitioning, and System/Software Design), we present the new methodology shown in Figure 1. As indicated, we differentiate between the original modeling sub-phases for embedded systems (in green), the security-relevant sub-phases (in yellow), and the verification steps (in blue). The solid black arrows show steps the designer must take in developing models manually, the blue arrows show automated steps such as model translation, and purple arrows show steps for reconsidering the model.

First, the **Analysis** phase considers the general needs of the system. We rely on Requirements Diagrams, which describe the functional and security needs of the system. Also, we design Attack Tree Diagrams, which are based on SysML Parametric Diagrams, (Kordy et al., 2013), one for each main attack against the system.

Next, in the **Hardware/Software Partitioning Phase**, we design the functionality and candidate architectures of a system in a highly abstracted way. This phase follows the Y-Chart approach, modeling

application and architecture separately, and then mapping tasks to architecture (Kienhuis et al., 2002). The Mapping model can then be evaluated in terms of Safety, Security, and Performance. Security verification of the mapping is performed based on Security Requirements and the Attacker Model. These results can lead us to reconsider any of the existing models. For security results, an attack trace is automatically generated for each falsified security property. Based on this attack trace, we can add Attacker Scenarios to the application model, modeling possible attacks. By examining the interaction of the attacker and system, we can update our Attack Trees, and also accordingly protect our system with Application-based and Architecture-based security countermeasures.

Once a mapping is satisfactory, we proceed to the **Software Design Phase** – a skeleton of this model can be automatically obtained from partitioning models – where software models are much more detailed that the high-level partitioning models. For instance, partitioning models abstract algorithms as complexity operations, while software models give the details of these computations. Security verification on the software algorithms including security protocols may find that certain security requirements are not met. The attack traces can again be used to generate new attacker scenarios, which may require changes to the Attacker Scenarios modeled in HW/SW Partitioning or Attack Trees. The HW/SW Partitioning models may also be revised. For example, if a security issue may not be easily corrected with the selected hardware architecture, we must return to the previous stage to reconsider those models.

System and Attacker Models are continually refined over iterations until our system is secure against all predicted attacks.

4 ANALYSIS: SECURITY REQUIREMENTS AND ATTACK TREES

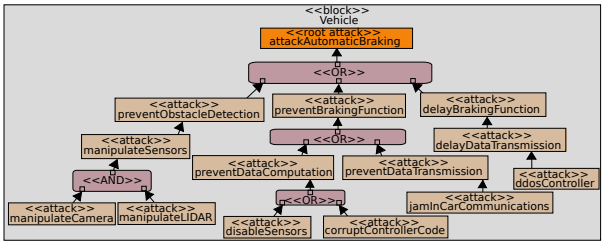

Figure 2: Attack Tree for Attacking Automatic Braking Function on Connected Car.

The Analysis phase involves describing the requirements and use cases of the system, and simultaneously considers the possible attacks that the system may face (Roudier et al., 2013). This analysis is expected to prepare the future system to be adapted to counter the listed attacks. Analysis also prepares the verification phase since properties to be evaluated are derived from requirements, and root attacks represent attacks that shall not be reachable.

Throughout this paper, we demonstrate our methodology on the design of a connected car equipped with sensors, V2X, etc. Based on data from the sensors, the system may need to send an emergency brake command to the ECUs. V2X data may be used to determine heavy traffic along the road, and the system should send its own V2X messages in the case of unexpected reduced speed. Even without a concrete model of the system, we can choose security requirements like “An attacker should not be able to affect vehicle function”, refined to “An attacker should not be able to prevent emergency braking”.

Next, we model the possible attacks violating this requirement, as shown in Figure 2. The root attack, ‘attack Automatic Braking’, is the ultimate goal of the attack, and to perform this attack, an attacker needs to perform a combination of the refined attack steps. Each attack step can be marked as possible or not, and analyzer determines if the root attack is logically possible. For example, if none of the 3 ways to attack automatic braking ‘Prevent Brake Function Activation’, ‘Prevent Braking Function’, and ‘delay Braking Function’ are possible, then the root attack cannot be carried out.

Depending on the system design, attack steps may be added or removed, and the analyzer should be run again to determine if each root attack is possible.

5 HW/SW PARTITIONING

The HW/SW Partitioning Phase involves the design of the high-level functions and architecture of the system. Based on the models and evaluation in this phase, we can refine our developed attack trees or discover new possible attacks, as explained below.

5.1 Modeling

A partitioning consists of three main types of models: the application model, architectural model, and mapping model. First, we describe the base models in this phase, and then discuss how to consider the attacker capabilities.

The Application Model is a set of communicating tasks built upon SysML block diagrams. The behav-

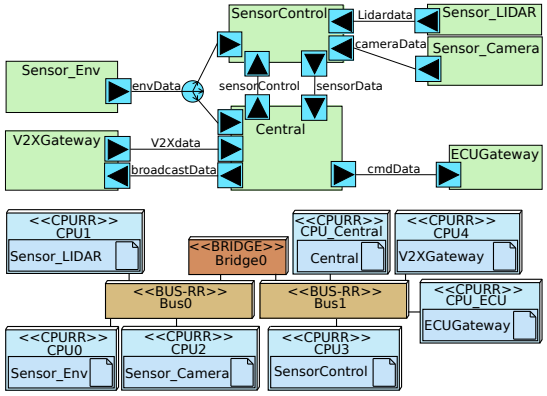

Figure 3: Application and Mappings Models for Connected Vehicle Example.

ior of tasks is described abstractly with activity diagrams. The Architecture Model displays the underlying architecture as a network of abstract execution nodes, communication nodes, and storage nodes. The Mapping Model then partitions the application into software and hardware, and specifies the location of their implementation on the architectural model. A task mapped onto a processor will be implemented in software, and a task mapped onto a hardware accelerator will be implemented in hardware.

The application and architecture model for our connected car is shown in Figure 3. The systems contains a Central controller communicating with V2X, and Sensor Control unit which manages the different sensors. Different environmental conditions affect the functionality of the system: during rain, braking distance changes, and the reliability of different sensors is also affected. The architecture/mapping consists of two central bus systems and one processor per task.

5.2 Security Countermeasures

In our previous work (Li et al., 2017), we demonstrated how security can be considered in the HW/SW Partitioning phase. As the V2X Gateway offers an exterior connection to the outside world, we investigate the consequences if it is compromised and allows an attacker access to the internal vehicular communications (Henniger et al.,). In our example, we assume the attacker has access to all communications along the two main buses (i.e., *Bus*0 and *Bus*1).

Even before performing formal verification, we can add preliminary security countermeasures to our mapping model to protect important communications. As all inter-task data is being sent on buses accessible to the attacker, we must secure the communications with Application Countermeasures, which include encrypting data to prevent it from being comprehensible to an attacker, or concatenating data with a calculated Message Authentication Code (MAC) to ensure the message cannot be modified.

Architectural Countermeasures such as Hardware Security Modules and Firewalls can also be added. Since security operations can be computationally-intensive and require the secure storage of cryptographic elements (e.g., keys), there exist specialized co-processors Hardware Security Modules which can perform cryptographic operations faster than a normal processor and can store cryptokeys.

Firewalls filter communications and can block an attacker without having to attempt to decrypt/analyze the contents of his/her communications. Filtering communications could prevent an attacker from injecting code to a certain extent, if we limit the size of allowed messages.

In our first secured model, we only add application countermeasures, in the form of MACs to data that we do not want the attacker to forge, most importantly commands to the ECU. In our next step, we evaluate if these countermeasures are sufficient and prevent the 'Attack Braking Function' attack described in the Analysis Phase.

5.3 Evaluation of Security

Data sent across channels can be evaluated in terms of confidentiality and authenticity. We perform security verification with ProVerif, a verification tool operating on a pi-calculus specifications of communicating processes (Blanchet, 2001). ProVerif assumes a Dolev-Yao attacker, who can read and write on any public channel, create new messages, and apply cryptographic primitives. We also transform the security requirements described in the Analysis phase into formal security queries, which we send to ProVerif. The verifier can check if queried data can be recovered by the attacker (which violates confidentiality), injected by the attacker (which violates weak authenticity), or replayed by the attacker (which violates strong authenticity).

As shown in the methodology, verification results for properties proven false are shown as 'Attack traces' showing the attacker actions performed. Command data sent from the central controller to ECUs was concatenated with their MAC, to ensure that the attacker cannot modify the command.

This attack could be prevented if the original message was concatenated with a sequence number or timestamp. We could also reconsider the architecture, such that the attacker would have no access to the bus between Central and ECU Gateway.

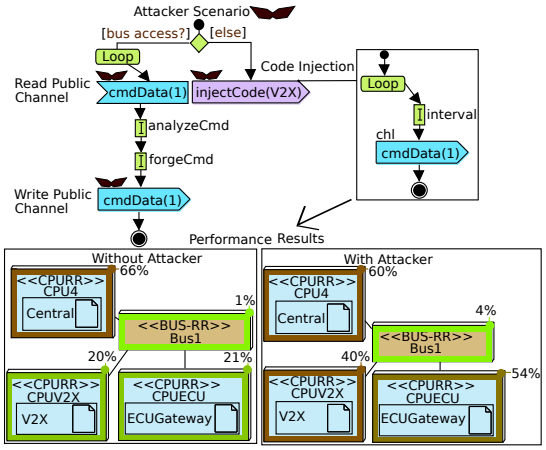

Figure 4: Attacker Scenario and impact on performance.

5.4 Attacker Scenarios

To better evaluate the attacker impact on a system, we model Attacker Scenarios in the form of an additional task on the component diagram. They describe their abstract behavior in terms of their injected data.

An attacker scenario consists of one or many attacker tasks, representing separate functions working together to carry out an attack on a system. An attacker task's behavior includes all possible actions of a normal application task (read/write data on a channel, control operations, execution of calculations, etc). In addition, where regular tasks can only read and write on channels directly associated with themselves, the attacker tasks may read and write on any public channel. A public channel is defined to be any channel mapped to a path including at least one bus accessible to the attacker. Furthermore, attacker tasks possess an additional capability: 'Code Injection', which replaces an application task's behavior with an attacker-determined one, modeling an attacker's capability to change a task's execution flow which may include code modification.

We can also examine the impact on the system when sending forged messages, whether sent by the attacker directly or by a task whose code has been modified. We previously noted that replay attacks could be prevented by adding a sequence number to messages before calculating the MAC. However, the ECU Gateway must still process those messages even if it will detect that they are forged. We can evaluate the attacker impact on the system using simulation. If we assume a rate at which the attacker can inject, then we can evaluate using a simulation the impact of performance. If we notice that the bus or processors are processing mostly only the attacker messages or show dramatically increased load, these performance metrics can be used to conclude about the Availability of a system.

The top of Figure 4 shows an attacker scenario where the attacker has two options depending if it has read/write access to the bus directly. The three Attacker Behavior Elements are marked with the Attacker mask symbol, and labeled with their action. With access to the bus, the attacker attempts to read the public channel of current ECU commands and then inject a fake one (by writing to the public channel ECUcmd) that differs significantly (for example, injecting a command to turn sharply while driving straight on a freeway). Otherwise, the attacker will perform code injection and change the functionality of the V2X Gateway to constantly inject messages to prevent legitimate commands from being processed. We analyze the performance results in the second case, as shown in the bottom of Figure 4. The load of each architectural component is calculated as active cycles divided by total execution cycles. In this case, we notice a significant increase in computations performed on the ECU, and an increase in Bus load. This new modeling capability allows us to analyze the property 'Availability'.

These Attacker Scenarios can be refined in the following Software/System Design Phase, after which we reconsider the attack trees to consider these new attacks.

6 SOFTWARE DESIGN

The Software Design phase models the software part of the system in greater detail. The general frameworks of the Software models are generated automatically from HW/SW Partitioning models, leaving the user to refine abstract elements, e.g. the implementations of algorithms.

The security of channels is determined by the architecture in the previous stage. Indeed, while the architecture diagrams are not part of this phase, they are initially used to choose whether the connections between blocks of the Software Design models are accessible to the attacker or not.

6.1 Attacker Model

Attacker Scenarios in Software/System Design can be used to analyze the data communications in more detail. For example, we can analyze the communications between the sensors. In the previous phase, we assumed that the attacker could access the bus on which the different sensors communicate. Therefore,

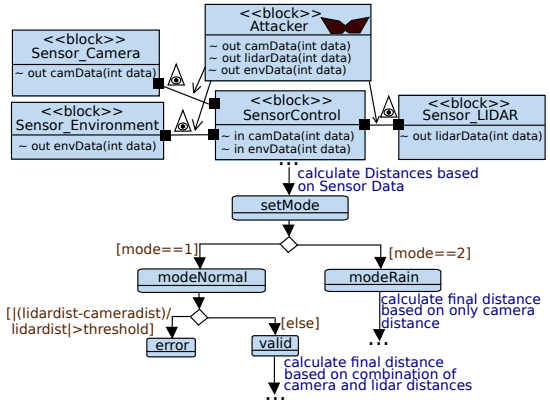

Figure 5: Sensor Software Design Models with Attacker.

we model the communications between the components in this phase as occurring on a public channel accessible to the attacker, as shown in the top of Figure 5, where public channels are marked with the eye in a triangle. In our system, the Sensor Control block implements a plausibility and coherence check on the data received by the sensors as shown in Figure 5. The attacker cannot inject arbitrary data that would greatly differ from the other data present in the system, for then the system will detect an anomaly. If the calculated distance to the nearest obstacle differs by over $X\%$, then the system could assume a malfunction or attack was taking place and warn the user. In case of rain, however, LIDAR data is not used and no coherence check takes place.

However, if it were possible for the attacker to imitate both the LIDAR and the camera, then the system would accept the forged data. If it should be possible to attack the environmental sensor (which may be user set or acquiring meteorological data by internet), then the attacker would only need to attack one of the other two sensors.

6.2 Reconsideration of Attack Trees

In the last two sections, we examined different possible attacks on the system, including Denial of Service attacks on the ECU Gateway, forging sensor data, etc.

We notice that the preliminary Attack Tree in Figure 2 was incomplete, as it did not cover all these attack scenarios. For example, the attack 'Manipulate Sensors' could also involve manipulating the Environment sensor. Also, we can elaborate on some of the listed attack steps, such as 'prevent Braking Function' that would involve the attack sub-steps 'attack V2X Gateway' to gain control, and then 'replay ECU commands'. The preliminary Attack Tree of Figure 2 must be enhanced to add the new attacks which may be possible in our system, as shown in Figure 6.

When we add countermeasures, certain attack steps become impossible. For example, we could add a firewall to ensure in-car communications cannot be jammed. Furthermore, we could add timestamps to ECU commands before calculating a MAC to prevent replay attacks as we described in section 5.4.

As the root attack is still logically possible, more countermeasures must be added to protect our system. Subsequent iterations through the design process could involve ensuring the sensor data could not be forged, or considering if keys should be periodically distributed. Then, the security verification could evaluate if there exist vulnerabilities in the implementation of those security protocols, which may suggest future attacker scenarios, and consequently, modifications to the displayed Attack Tree or new Attack Trees.

7 RELATED WORK

Regarding security in embedded systems, multiple works develop toolkits for the design process only (Balarin et al., 2003; Rosales et al., 2014; Kangas et al., 2006), and many study the verification of security, but few study both in one framework. Of the works which investigate both, (Hansson et al., 2010) relies on Architecture Analysis and Design Language (AADL) models to consider architectural mapping during security verification. The authors note that a system must be secure on multiple levels: software applications must exchange data in a secure manner, and also execute on a secure memory space and communicate over a secure channel. Our work, however, investigates protections against an external attacker instead of access control.

Another approach performs Design Space Exploration on a vehicular network protecting against replay and masquerade attacks (Lin et al., 2015). The project evaluates possible security mechanisms, their effects on message sizes, and candidate architectures during the mapping phase. While their work targets automotive systems and network communications, our analysis may be applied more broadly for any embedded system. Also, we focus on the integrity and confidentiality of data itself instead of on specific attacks.

7.1 Security Modeling and Verification

Attack Defense Trees (Kordy et al., 2013) analyze the possible attacks against a system, in conjunction with the defenses that the system may implement. The supporting toolkit ADTool analyzes attack scenarios to

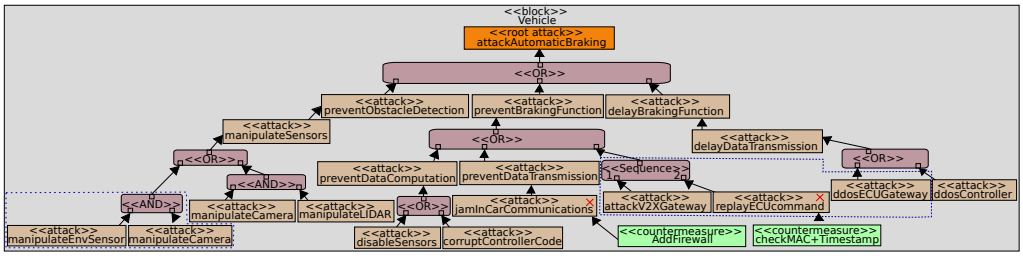

Figure 6: Modified Attack Tree after System Design and Evaluation.

determine the cost, probability, time, etc, required for a successful attack.

The Knowledge Acquisition in Automated Specifications approach Security Extension aims to identify security requirements for software systems (van Lamsweerde, 2004). The methodology uses a goal-oriented framework and builds a model of the system, and then an anti-model which describes possible attacks on the system. Both models are incrementally developed: threat trees are derived from the anti-model and the system model adds security countermeasures to protect against the attacks described in the anti-model.

The Combined Harm Assessment of Safety and Security for Information Systems (CHASSIS) method considers safety and security together in a common model (Raspotnig et al., 2013). Safety and security hazards in the form of misuse cases are developed, and then trade-off analysis to unify all requirements and identify when safety and security conflict. While these techniques targeting the requirements and analysis phase offer a detailed approach to considering threats against safety and security, they are not yet automated.

Unlike these works, our work allows formal verification of attack trees to check the possibility of executing a root attack, and then continues on to designing the system itself in the subsequent stages.

(Vasilevskaya and Nadjm-Tehrani, 2015) proposed modeling security in embedded systems with attack graphs to determine the probability that data assets could be compromised. While their approach is also UML-based, they focus on estimating probabilities of success for attacks, while ours focuses on verifying adequate placement of encryption.

UMLSec (Jürjens, 2002) is a UML profile for expressing security concepts, such as encryption mechanisms and attack scenarios. It provides a modeling framework to define security properties of software components and of their composition within a UML framework. It also features a rather complete framework addressing various stages of model-driven secure software engineering from the specification of security requirements to tests, including logic-based formal verification regarding the composition of software components. However, UMLSec does not take into account the HW/SW Partitioning phase necessary for the design of IoTs.

The authors of (Dawkins and Hale, 2004) leveraged Attack Trees to analyze network security. Both system vulnerabilities and attacker capabilities are modeled and analyzed to determine the possible attacks a system may face. (Hong and Kim, 2016) also used a hierarchical attack representation model (HARM) to assess different moving target defenses, modeling both system vulnerabilities and attack graphs and trees. (Ge et al., 2017) described a similar framework modeling attacks and defenses in IoTs. While similar to our work as they described the system vulnerabilities along with the attacker actions, our toolkit models system behavior including countermeasures instead of only vulnerabilities, and is supported by formal verification.

8 CONCLUSION

Our Model-Driven methodology combines Attack Trees with Embedded System Design, developing an iterative design process evolving system models and attacker models collaboratively. As demonstrated using a connected car case study, Attack Trees developed in the preliminary first steps of modeling have an incomplete vision of the system, and need to be further refined as the system is developed. Security Evaluations reveal additional attacks, or provide implementation details for existing predicted attacks. As long as there exist executable root attacks, the system must be further protected with additional countermeasures.

Future work will seek to better automate the generation of Attacker Scenarios. Attack traces could be automatically translated into the attacker's functional model. We could reconstruct functional attack scenarios based on attack traces and automatically integrate them to the attack trees so that the designer is able to improve system design or rework the attacker model depending on the effect of the scenario on the system.

We also plan to provide a toolbox of Attacker Models, to provide the designer with some templates or example attack scenarios, such as DDoS attacks, replay attacks, brute-forcing of keys, etc., that he/she can then further enhance. While our current security prover ProVerif has limited support for mathematical operations, we plan to integrate safety and security verification to allow formal evaluation of plausibility and coherence checks that we currently perform with simulation. We may find a different security prover, or integrate security reasoning into our reachability graphs. Furthermore, we'd like to better connect analysis and implementation, such as connecting Requirements Diagrams with implemented countermeasures, etc. Our ultimate goal will be a comprehensive methodology for systematic modeling and verification of security.

REFERENCES

Apvrille, L. and Roudier, Y. (2015). SysML-Sec: A Model Driven Approach for Designing Safe and Secure Systems. In *3rd International Conference on Model-Driven Engineering and Software Development, Special session on Security and Privacy in Model Based Engineering*, France. SCITEPRESS Digital Library.

Balarin, F., Watanabe, Y., Hsieh, H., Lavagno, L., Passerone, C., and Sangiovanni-Vincentelli, A. (2003). Metropolis: An Integrated Electronic System Design Environment. *Computer*, 36(4):45–52.

Blanchet, B. (2001). An efficient cryptographic protocol verifier based on prolog rules. In *Proceedings of the 14th IEEE Workshop on Computer Security Foundations*, CSFW '01, pages 82–, Washington, DC, USA. IEEE Computer Society.

Constantin, L. (2016). Researchers hack Tesla Model S with remote attack. http://www.pcworld.com/article/3121999/security/researchers-demonstrate-remote-attack-against-tesla-model-s.html.

Dawkins, J. and Hale, J. (2004). A systematic approach to multi-stage network attack analysis. In *Information Assurance Workshop, 2004. Proceedings. Second IEEE International*, pages 48–56. IEEE.

Ge, M., Hong, J. B., Guttmann, W., and Kim, D. S. (2017). A framework for automating security analysis of the internet of things. *Journal of Network and Computer Applications*, 83:12–27.

Hansson, J., Wrage, L., Feiler, P. H., Morley, J., Lewis, B., and Hugues, J. (2010). Architectural Modeling to Verify Security and Nonfunctional Behavior. *IEEE Security Privacy*, 8(1):43–49.

Henniger, O., Apvrille, L., Fuchs, A., Roudier, Y., Ruddle, A., and Weyl, B. Security Requirements for Automotive On-Board Networks. In *ITST 2009, Lille, France*.

Hong, J. B. and Kim, D. S. (2016). Assessing the effectiveness of moving target defenses using security models. *IEEE Transactions on Dependable and Secure Computing*, 13(2):163–177.

Jürjens, J. (2002). UMLsec: Extending UML for Secure Systems Development. In *Proceedings of the 5th International Conference on The Unified Modeling Language*, UML '02, pages 412–425, London, UK, UK. Springer-Verlag.

Kangas, T., Kukkala, P., Orsila, H., Salminen, E., Hännikäinen, M., Hämäläinen, T. D., Riihimäki, J., and Kuusilinna, K. (2006). UML-based Multiprocessor SoC Design Framework. *ACM Trans. Embed. Comput. Syst.*, 5(2):281–320.

Kienhuis, B., Deprettere, E., van der Wolf, P., and Vissers, K. (2002). A Methodology to Design Programmable Embedded Systems: The Y-Chart Approach. In *Embedded Processor Design Challenges*, pages 18–37. Springer.

Kordy, B., Kordy, P., Mauw, S., and Schweitzer, P. (2013). Adtool: Security analysis with attackdefense trees. In Joshi, K., Siegle, M., Stoelinga, M., and DArgenio, P., editors, *Quantitative Evaluation of Systems*, volume 8054 of *Lecture Notes in Computer Science*, pages 173–176. Springer Berlin Heidelberg.

Larson, S. (2017). FDA confirms that St. Jude's cardiac devices can be hacked. http://money.cnn.com/2017/01/09/technology/fda-st-jude-cardiac-hack/.

Li, L. W., Lugou, F., and Apvrille, L. (2017). Security-Aware Modeling and Analysis for HW/SW Partitioning. In *Conférénce on Model-Driven Engineering and Software Development (Modelsward'2017)*, Porto, Portugal.

Lin, C.-W., Zheng, B., Zhu, Q., and Sangiovanni-Vincentelli, A. (2015). Security-Aware Design Methodology and Optimization for Automotive Systems. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 21(1):18.

Miller, C. and Valasek, C. (2015). Remote exploitation of an unaltered passenger vehicle. *Black Hat USA*.

Newman, L. (2016). The Botnet That Broke the Internet Isn't Going Away. https://www.wired.com/2016/12/botnet-broke-internet-isnt-going-away/.

Raspotnig, C., Katta, V., Karpati, P., and Opdahl, A. L. (2013). Enhancing CHASSIS: A Method for Combining Safety and Security. In *Availability, Reliability and Security (ARES), 2013 Eighth International Conference on*, pages 766–773.

Rosales, R., Glass, M., Teich, J., Wang, B., Xu, Y., and Hasholzner, R. (2014). MAESTRO— Holistic Actor-Oriented Modeling of Nonfunctional Properties and Firmware Behavior for MPSoCs. *ACM Trans. Des. Autom. Electron. Syst.*, 19(3):23:1–23:26.

Roudier, Y., Idrees, M. S., and Apvrille, L. (2013). Towards the Model-Driven Engineering of Security Requirements for Embedded Systems. In *proceedings of MoDRE'13, Rio de Janeiro, Brazil*.

van Lamsweerde, A. (2004). Elaborating Security Requirements by Construction of Intentional Anti-Models. *In Proc. of the 26th International Conference on Software Engineering , ICSE '04*, pages 148–157.

Vasilevskaya, M. and Nadjm-Tehrani, S. (2015). *Quantifying Risks to Data Assets Using Formal Metrics in Embedded System Design*, pages 347–361. Springer International Publishing, Cham.