Aspect-oriented Modeling of Attacks in Automotive Cyber-Physical Systems

Armin Wasicek
University of California, Berkeley
arminw@berkeley.edu

Patricia Derler
University of California, Berkeley
patriciad@berkeley.edu

Edward A Lee
University of California, Berkeley
eal@berkeley.edu

ABSTRACT

This paper introduces aspect-oriented modeling (AOM) as a powerful, model-based design technique to assess the security of Cyber-Physical Systems (CPS). Particularly in safety-critical CPS such as automotive control systems, the protection against malicious design and interaction faults is paramount to guaranteeing correctness and reliable operation. Essentially, attack models are associated with the CPS in an aspect-oriented manner to evaluate the system under attack. This modeling technique requires minimal changes to the model of the CPS. Using application-specific metrics, the designer can gain insights into the behavior of the CPS under attack.[1]

Categories and Subject Descriptors

[**Computer systems organization**]: Embedded and cyber-physical systems; [**Security and Privacy**]: Software and application security—*Software security engineering*

Keywords

Aspect-oriented Modeling, Security, Cyber-Physical Systems

1. INTRODUCTION

Providing secure and safe mobility is a key challenge for a modern society. More and more electronic components and control systems are deployed to solve this challenge. Such systems incorporate networked computational and physical processes and are generally referred to as Cyber-Physical Systems (CPSs) [16]. Security, which is the protection against malicious manipulation, is a mission-critical property of any CPS. In particular, intrusions that can propagate from the security to the safety domain with potentially catastrophic consequences are a major concern in CPSs.

Recent incidents such as the Stuxnet attack [9] suggest that CPSs are vulnerable to malicious attacks. In the past, security has been no or only a minimal concern in CPS design and automotive engineering for several reasons: First, systems have operated predominantly in isolation and, hence, there was no need for security considerations. Second, during development, functionality has often taken priority over security. Security measures were implemented late as an add-on resulting in brittle designs that lack proper integration. Third, security measures did not cover the intended or required attack model and attack models did not take into account the particularities of CPS. Our approach addresses these challenges early during the design phase of a CPS.

Building a CPS [15] is, by nature, a multi-disciplinary design challenge that requires expertise from the control, software, and hardware domain. Model-based design is proven to be an effective, computer-aided design methodology that increases productivity and reduces cost through the early location and detection of faults and design reuse [3]. When carrying out a model-based design approach, several models reflecting different aspects of the required system are developed and used throughout the system development process. Building these models is a complex task that requires experts from different engineering fields to work closely together. The model-based design tool chain is the main point of interaction between different branches of engineers. Therefore, we engage at this focal point and extend the model-based design process by modeling attacks on the CPS.

In particular, we propose to extend the model-based design process to include security concerns, to use Aspect-Oriented Modeling (AOM) techniques to reason about systems and attacks, to integrate attack models in the analysis of control systems, and to reuse patterns and models of prototypical attacks.

The evaluation of the system behavior in case of attacks should not require major changes to the system model. We use AOM techniques to associate attack models with system components. An attack model is considered to be an aspect that is associated with the communication paths in the system under attack. This association is done by annotating

[1] This research was in part supported by a Marie Curie IOF Action within the 7th Framework Programme under the funding ID PIOF-GA-2012-326604 (MODESEC) and the iCyPhy Research Center (Industrial Cyber-Physical Systems, supported by IBM and United Technologies), and the Center for Hybrid and Embedded Software Systems (CHESS) at UC Berkeley (supported by the National Science Foundation, NSF awards #0720882 (CSR-EHS: PRET) and #0931843 (ActionWebs), the Naval Research Laboratory (NRL #N0013-12-1-G015), and the following companies: Bosch, National Instruments, and Toyota).

DAC '14, June 01 - 05 2014, San Francisco, CA, USA
Copyright 2014 ACM 978-1-4503-2730-5/14/06…$15.00.
http://dx.doi.org/10.1145/2593069.2593095

the communication between components. As a result, we can develop various attack models independently from the system model and eventually build up a library of attack models.

The organization of the paper is as follows: Section 2 discusses the background and the related work of attack models and AOM. The core idea of using AOM in a security engineering process is described in Section 3. Section 4 is dedicated to an automotive case study and, finally, Section 5 draws conclusions.

2. BACKGROUND AND RELATED WORK

In this section we discuss variants of attack models as well as the aspect-oriented programming paradigm. We also mention other approaches on aspect-oriented attack modeling.

2.1 Attack Models for CPS

An attack model describes an adversary by its capabilities and/or behaviors. For example, the Dolev-Yao adversary model [7] assigns an attacker unlimited computational power and bandwidth. Commonly, attack models are used to identify weaknesses in a design, to support the search for mitigation strategies, and to promote the understanding of attacks. Thus, the first step in a security engineering approach is to develop appropriate attack models. Subsequently, the risk of potential attacks is estimated and adequate countermeasures can be defined.

Traditionally, most attack models on computer systems are represented either textually or graphically. Textual models are used to describe *security incidents* [12] with the goal of exchanging information on attacks across system boundaries. Graphical attack models are either tree-based or graph-based. Attack trees [22] are similar to fault trees in Fault-Tree Analysis (FTA). The root of the tree represents the goal of the attack and the different branches represent different attack paths that lead to a successful execution. Graph-based methods [14] provide a more general flexible structure to express causes and consequences of attacks. Trees and graphs are parsed to gain insights on viable attack paths or to estimate risks. If nodes and relations are enriched with probabilities, they can be processed, for instance, with Bayesian inference algorithms.

Traditional approaches for attack modeling are not applicable for CPS. When including physical processes in the system model, emphasis is put on the interaction between cyber and physical parts of the CPS and physical entities such as time in the computer system need to be considered. Cárdenas et al. [5] therefore postulate *'realistic and rational adversary models'* for CPS. Liu et al. [18] model attackers by malicious sensors that inject false data in the controller of an electrical power grid. Cárdenas et al [4] describe attackers with a similar capability to influence the output of a chemical reactor plant. Pasquetti et al. [19] include a special disturbance signal which is used to represent a specific attack strategy. All these approaches aim to model prototypical attacks on computer systems like replay, injection, or Denial-of-Service (DoS) by control theoretic means and as a result, they translate the security problem into a robustness problem for control.

2.2 Aspect-oriented Modeling

The ideas for AOM are based on the Aspect-Oriented Programming (AOP) [13] paradigm, where cross-cutting functionality is modeled as aspects and annotations in object-oriented code connect aspect functionality to object functionality. The motivation behind AOP is separation of concerns, i.e. while object-oriented programs separate concerns into independent objects, AOP also separates out aspects that concern multiple objects. Benefits of AOP are reduction of code duplication, code scattering, and code tangling resulting in cleaner, more understandable designs.

2.3 Other aspect-oriented approaches for attack modeling

Other works have used aspect-oriented modeling techniques to create attack-defense models. Xu et al. [23] extend a system described by a Petri-net with additional Petri-nets formalizing attacks and countermeasures. Verification is done by evaluating whether a threat path is part of a possible firing sequence. Georg et al. [11] refine a UML model with attacks generating the *misuse model* and countermeasures resulting in the *security-treated model*. The latter is then analyzed using formal methods to verify if assets are sufficiently protected.

Our work diverges from these approaches by targeting CPS rather than web services and e-commerce applications. Moreover, our modeling environment facilitates heterogeneity, which allows the definition of system, attack, and countermeasure with different formalisms. Finally, the verification step in this work focuses on proving presence or absence of a property in the secured system model. Our approach suggests the detailed analysis of the impact of an attack on the system by using domain-specific metrics and methods (e.g., stability criterion for a control system).

3. MODELING ATTACKS AS ASPECTS

Aspect-oriented attack modeling is the concept of modeling attacks as aspects of a system and executing aspect and attack models together with the goal of evaluating the system under attack. We explore the use of AOM as discussed in [6], which facilitates loose coupling of attack and system models.

Aspect models can be elaborate and internally use specific Model of Computations (MoCs). Therefore, we explore modeling aspects as actors in actor-oriented designs. An aspect actor annotates other actors, the execution of other actors, or the communication between actors with aspect-specific attributes. Attack aspect actors modify the communication between actors by introducing undesirable behavior.

3.1 Benefits of Aspect-oriented Modeling

The construction of CPS follows a rigorous engineering process that handles functional as well as non-functional properties. However, the requirements for both kinds of properties are often contradictory or the work on one kind might obstruct the work on the other. For instance, encrypting messages establishes confidentiality, but encumbers debugging, because inspecting the payload is not trivial anymore. As a consequence, security properties were often neglected during the development process.

We suggest to use AOM as part of a security engineering process to systematically secure a system against a certain attacks. Using AOM techniques aligns very well with existing model-based design approaches for CPS. Particularly, adding attack aspects to the system model enables:

Separation of concerns:
Separating functional and attack models through aspects facilitates domain experts to work on different aspects without interfering. For instance, the control expert can develop robust control algorithms whereas the security engineer defines attack models. Both views are synthesized through the model-based design tool.

Design space exploration:
Anticipating the effects of an intrusion is not trivial, because the effects might manifest in a very different part of the system than the attack. Moreover, differentiating secure and insecure system states is challenging, because an attacker will most likely aim to remain covered. Simulation of different attack scenarios promotes an understanding of attack vectors. Moreover, model-based design facilitates the creation of arbitrary attack scenarios, even ones that would be risky in the real world.

Reduction of complexity:
System models can become very complex, i.e., the number of entities and relations is very high. AOM reduces this complexity by making only those aspects that are relevant visible.

Improved Testing:
In-the-loop testing is an important testing strategy for CPS. Incrementally, parts of the system model are replaced with instances of real system components. For example, in restbus simulation controllers are connected to a model of the environment and the other parts of the system. Using aspects, the system model can quickly be adapted to run different testing campaigns including security tests.

Reuse of models:
General versions of an attack model can be instantiated for different systems, since an attack aspect actor is portable. It is part of this research to devise a library of attack models that can be embedded into different system models at different places in the system.

3.2 Design Process with Aspects

Introducing aspects-oriented techniques in the system design, does not necessarily change the development method for a CPS. The basic steps of an according systems engineering process are:

1. *Design:* Develop the system model according to requirements and specification. Include countermeasures, or intrusion tolerance strategies.

2. *Annotate:* Develop attack models that extend the system model in a non-intrusive way.

3. *Analyze:* Reason about the system. Verify and validate functional correctness, apply tests, checks, and metrics to measure properties like integrity, reliability, stability, etc. Go to step 1, if analysis fails.

4. *Synthesize:* Devise an implementation that can detect or dismiss certain attacks.

After refining the system model with appropriate attack aspect models, the systems engineer can verify, if the system achieves certain security goals. Steps 2 to 3 are iterated, until the system model satisfies requirements and security goals. The early location and correction of faults is a particular strength of the model-based design approach. This basic method can be followed partly automated, for instance, steps 3 and 4 have a high potential for automation. Each step, however, requires a supervision by an educated engineer. Therefore, no total, but reasonable automation is possible.

3.3 Model Analysis and Assessment

Our approach particularly aims at revealing potential vulnerabilities that manifest as malicious design and interaction faults [1]. Discovery and removal of bugs and vulnerabilities during design time is most cost efficient and much easier than in later the stages of the development. For instance, fixing software of an automotive system in production or even after rollout can get very costly. Removing a defect after deployment is 5× to 30× more expensive than a removal in the phase of introduction [21].

Model-based design tools offer various strategies to gain knowledge of a system. Executing models in a simulation is a very powerful method to examine a system's behavior. Particularly when dealing with models of complex physical components (which is the case in CPS), simulation is often the only way to study the behavior. Model executions support system engineers to answer what-if questions, get reference values for subsequent tests, and promote an understanding of the system.

The behavior of a system can be represented as a trace that is interpreted by applying metrics that measure properties of the system. An example for an application-specific metrics for a control system is the stability criterion. An automotive engineer, for instance, needs to know if an attack can influence the control system, i.e., if the control system remains stable despite an attack. Special security metrics might be used to verify security properties or to validate the effectiveness and performance of countermeasures. Using AOM, analysis and assessment methods can be implemented as aspects in the system model.

Certainly, there are as well some limitations. The outcome of a model execution is only as good as the input. If the attack model or the assumptions on the input do not hold, the analysis might lead in the wrong direction. Moreover, arbitrarily enabling and disabling aspects in the system model might change the model formalism such that system analysis is not straightforward anymore. For instance, an aspect might introduce new logic in an Linear Time-Invariant (LTI) system that invalidates the linearity property. Consequently, some stability criteria like Routh Hurwitz is not applicable anymore. This is not a real deficit, but rather a circumstance that the engineer must be aware of when modeling and analyzing.

4. AUTOMOTIVE CASE STUDY

We illustrate our research objectives on a model [17] of an adaptive cruise control system. More precisely, we use platooning which is a common application of adaptive cruise control as an example. A platoon couples two or more vehicles in a single sphere of control such that the vehicles accelerate or brake simultaneously.

4.1 Adaptive Cruise Control Model

Models in this case study are developed with Ptolemy II [8], a modeling and simulation framework for heterogeneous systems. Ptolemy models are actor-oriented, meaning that they consist of actors that execute concurrently. Actors communicate via ports. The semantics of actor execution and communication is determined by a special actor, the director. Ptolemy contains various directors that represent heterogeneous execution semantics, also referred to as MoC. Examples for MoCss are Discrete Event (DE), Continuous Time (CT), and Process Network (PN). Ptolemy supports heterogeneity, meaning that a model can embody different MoCss [20].

Ptolemy supports AOM [6]. An aspect actor is a special actor that *decorates* other model components with additional parameters (see the decorator design pattern [10]). In order to model attacks on a system, we focus on aspects that decorate communications between actors. Following example shows how an aspect works: Actor A sends a message via its output port to the input port p of actor B. Port p stores messages, also called tokens, until actor B is ready to consume them. An aspect that annotates a communication is associated with the input port of this communication - in this example it is port p. Tokens sent from actor A to actor B are first processed by the aspect and only then forwarded to the receiver in port p.

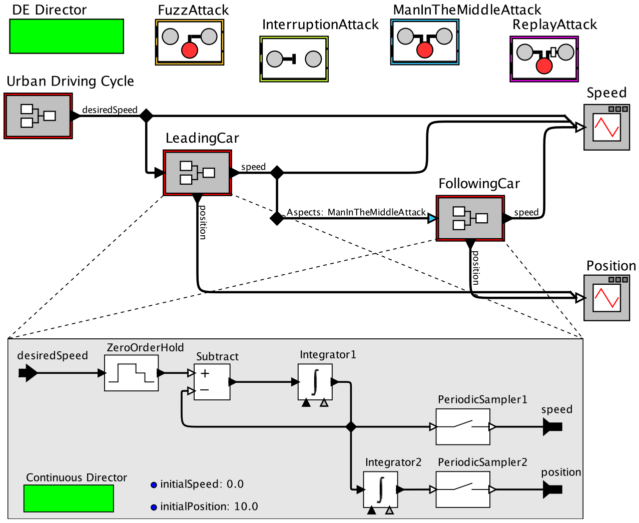

Figure 1: Aspect-oriented attack models decorating the communication between a controller its sensor/actuator interface in Ptolemy II

Figure 1 shows the Ptolemy model (Java Web Start version[2]) for adaptive cruise control with two cars: a leading car (actor *LeadingCar* and a following car (actor *FollowingCar*). The implementation of the car model, which is equivalent for leading and following car, is shown in the grey box in Figure 1. The car model is a composite actor that takes as an input the desired speed and outputs the speed and position of the car. It matches the desired speed using feedback control. The model is continuous, thus it uses the *Continuous Director*. The car model is initialized by two parameters: the *initialSpeed* and the *initialPosition*. These parameters are used to initialize the integrator actors; *Integrator1* is initialized with *initialSpeed* and *Integrator2* is initialized with *initialPosition*. We start both cars with an initial speed of 0.0. The initial position of the leading car is 10.0, the initial position of the following car is 0.0 to model that the leading car is 10 meters ahead of the following car. The control goal is to maintain this distance to keep the cars in the platoon.

The car model simulates acceleration, cruising, deceleration and breaking. The input for the leading car is provided by the actor *UrbanDrivingCycle* that emulates a driver according to ECE-15 [2].

We want to explore the effect of attacks on the communication between leading and following car. The model contains four attack models illustrated as aspect-oriented attack actors in Figure 1. The attack models are not directly connected to any actors in the model but the input port of the following car can be associated with an attack model. In the example, the input port is associated with the attack model *ManInTheMiddleAttack*. The color of the port and a textual description show this association.

4.2 Attack Models

We discuss four different attack models : (a) a man-in-the-middle attack (see Figure 4), also referred to as modification, (b) a fuzz attack (see Figure 2), (c) an interruption attack (see Figure 3), and (d), a replay attack(see Figure 5).

Figure 2 illustrates a fuzz attack, in the literature also referred to as fuzzing. In this attack, random, often invalid data is inserted at random times. The actor *PoissonClock* produces discrete events according to a Poisson process. The *Uniform* actor produces a random sequence with a uniform distribution.

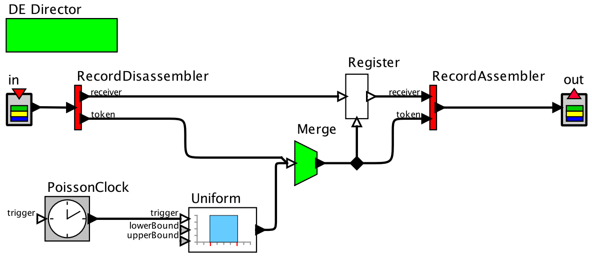

Figure 2: Fuzz attack as an aspect

An interruption of signal transfer is modeled in Figure 3.

[2] http://ptolemy.eecs.berkeley.edu/attackModeling/

Start and end of the attack are configured via the two parameters *attackStart* and *attackEnd* that are used in the actors *AttackStartEvent* and *AttackEndEvent*. These actors create a discrete event at the configured times. Between start and end event, tokens, are inhibited by the *Inhibit* actor.

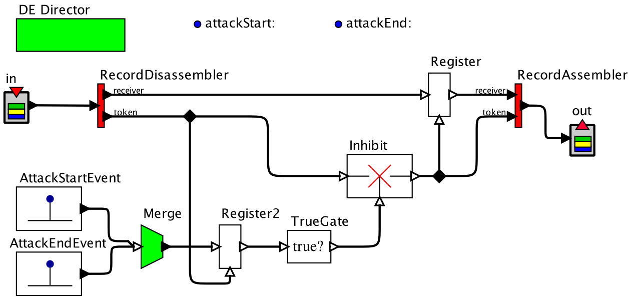

Figure 3: Interruption attack modeled as an aspect

The man-in-the-middle attack model is shown in Figure 4. Attack models are implemented as composite actors. Tokens processed by aspects are received by special input ports (see actor *in* in the attack models in Figures 4, 2 and These ports combine the incoming token into a record containing the target receiver and the token. The actors *RecordDisassenbler* and *RecordAssembler* are used to extract fields out of records and to combine fields into records. The two parameters *attackStart* and *attackEnd* define the temporal context of the attack. The malicious logic is implemented as a modal model with a normal state and an attack state. In the normal state, tokens are forwarded from input to output without modification. In the attack state, tokens are replaced by tokens modified through the *manipulate_expr* parameter. After processing of the incoming tokens by the modal model, the token is reassembled with the receiver that was stored temporarily in a register actor. A special output port, *out*, receives the record, takes the token and sends it to the receiver in the record.

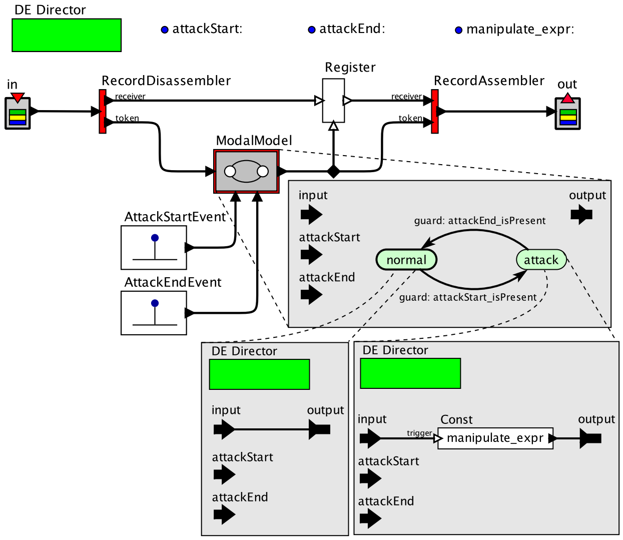

Figure 4: Man-in-the-middle attack modeled as an aspect

A model of a replay attack is illustrated in Figure 5. Similar to other attack models, the time this attack is active can be configured. The attack is implemented as a modal model with three states. In the *idling* state the array of recordedTokens is reset. In the *recording* state, incoming tokens are recorded but tokens are also forwarded to the receiver. In the *replaying* state, the array of recorded tokens is sent to the receiver instead of the incoming tokens.

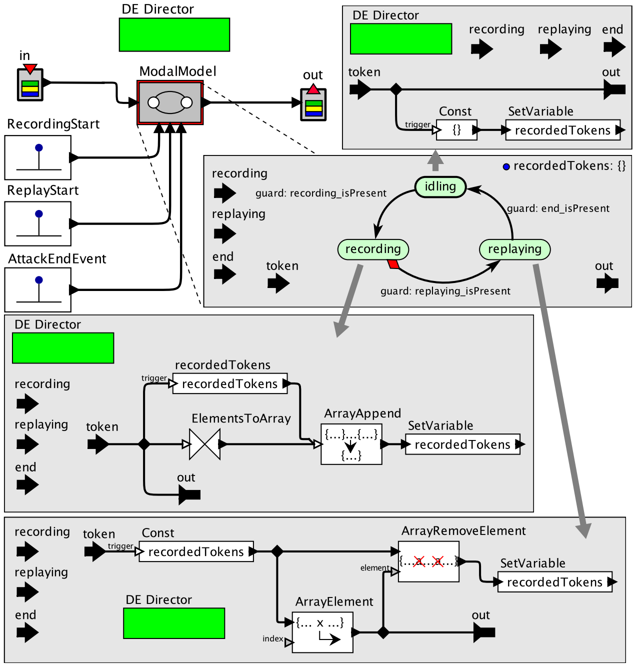

Figure 5: Replay attack as an aspect

4.3 Results

A comparison of the controller behavior is shown in Figure 6. Lines (a) and (b) are the reference trajectories. In (c), the fuzz attack speeds up the following car to potentially go below a safe distance. Attacks in (d) and (e) accelerate the following car such that it crashes into the leading car in front. Finally, (f) slows down the second car, which will consequently leave the platoon.

Aspect-oriented modeling is a very powerful way of performing design space exploration in a minimally intrusive ways. Special attention has to be paid to the mechanisms of linking aspects to system models and to the model of computation of the aspect. For instance, an aspect that handles delays and resource contention needs a timed model of computation and cannot be embedded into an untimed model of computation. Special attention has to be paid, if MoCss in the system model and aspect model implement different notions of time which can potentially introduce discontinuities and disrupt control algorithms.

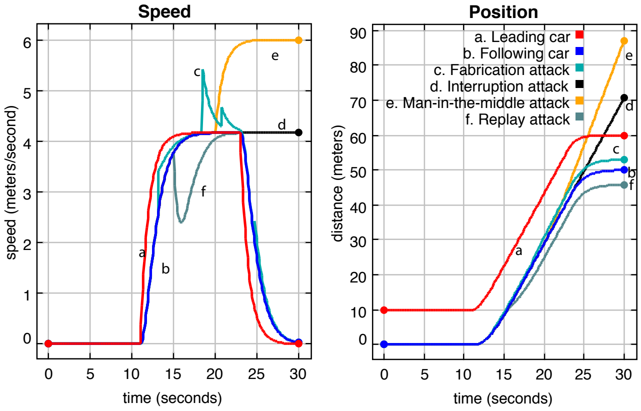

Figure 6: Comparison of Controller behavior under individual attacks

5. CONCLUSION

In this work, we presented aspect-oriented modeling as a powerful modeling technique to assess a CPS' security under specified attack models. It enables assessing system models and associated attacks within the same model environment, but using potentially distinct MoC. Using this technique, future work will be dedicated to researching appropriate executable attack models for CPS and generating instances of attacks from general attack patterns.

6. REFERENCES

[1] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, January 2004.

[2] T. J. Barlow, S. Latham, I. S. McCrae, and P. G. Boulter. A reference book of driving cycles for use in the measurement of road vehicle emissions. Published Project Report PPR354, TRL Limited, June 2009.

[3] M. Broy, M. Feilkas, M. Herrmannsdoerfer, S. Merenda, and D. Ratiu. Seamless Model-Based Development: From Isolated Tools to Integrated Model Engineering Environments. *Proceedings of the IEEE*, 98(4):526 – 545, 2010.

[4] A. A. Cárdenas, S. Amin, Z.-S. Lin, Y.-L. Huang, C.-Y. Huang, and S. Sastry. Attacks Against Process Control Systems: Risk Assessment, Detection, and Response. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, March 2011.

[5] A. A. Cárdenas, S. Amin, and S. Sastry. Secure control: Towards survivable cyber-physical systems. In *Proceedings of the First International Workshop on Cyber-Physical Systems (WCPS)*, June 2008.

[6] J. Cardoso, P. Derler, J. C. Eidson, E. A. Lee, S. Matic, Y. Zhao, and J. Zou. Modeling timed systems. In C. Ptolemaeus, editor, *System Design, Modeling, and Simulation using Ptolemy II*, page 355. Ptolemy.org, 2014.

[7] D. Dolev and A. C. Yao. On the security of public key protocols. In *Proceedings of the IEEE 22nd Annual Symposium on Foundations of Computer Science*, pages 350–357, 1981.

[8] J. Eker, J. W. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Sachs, Y. Xiong, and S. Neuendorffer. Taming heterogeneity - the ptolemy approach. *Proceedings of the IEEE*, 91(1):127–144, 2003.

[9] N. Falliere, L. O. Murchu, and E. Chien. W32.stuxnet dossier. Security Response v1.4, Symantec, Feb 2011.

[10] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[11] G. Georg, I. Ray, K. Anastasakis, B. Bordbar, M. Toahchoodee, and S. H. Houmb. An aspect-oriented methodology for designing secure applications. *Information and Software Technology*, 51(5):846 – 864, 2009.

[12] J. D. Howard and T. A. Longstaff. A common language for computer security incidents. Report SAND98–8667, Sandia National Laboratories, 1998.

[13] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. marc Loingtier, and J. Irwin. Aspect-oriented programming. In *ECOOP*. SpringerVerlag, 1997.

[14] B. Kordy, L. Pietre-Cambacedes, and P. Schweitzer. Dag-based attack and defense modeling: Don't miss the forest for the attack trees. *arXiv*, 1303.7397, 2013.

[15] E. Lee. Cyber physical systems: Design challenges. In *International Symposium on Object Oriented Real-Time Distributed Computing (ISORC)*, 2008.

[16] E. A. Lee. CPS foundations. In *Proceedings of the 47th Design Automation Conference*, DAC '10, pages 737–742, New York, NY, USA, 2010. ACM.

[17] J. M.-K. Leung, T. Mandl, E. A. Lee, E. Latronico, C. Shelton, S. Tripakis, and B. Lickly. Scalable semantic annotation using lattice-based ontologies. In *12th International Conference on Model Driven Engineering Languages and Systems*, pages 393–407. ACM/IEEE, October 2009. (recipient of the MODELS 2009 Distinguished Paper Award).

[18] Y. Liu, P. Ning, and M. K. Reiter. False data injection attacks against state estimation in electric power grids. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, 2009.

[19] F. Pasqualetti, F. Dorfler, and F. Bullo. Attack detection and identification in cyber-physical systems. *Automatic Control, IEEE Transactions on*, 58(11):2715–2729, 2013.

[20] C. Ptolemaeus, editor. *System Design, Modeling, and Simulation using Ptolemy II*. Ptolemy.org, 2014.

[21] G. Tassey. The economic impacts of inadequate infrastructure for software testing. RTI Project Number 7007.011, NIST, 2002.

[22] J. D. Weiss. A system security engineering process. In *Proceedings of the 14th National Computer Security Conference*, 1991.

[23] D. Xu and K. Nygard. Threat-driven modeling and verification of secure software using aspect-oriented petri nets. *Software Engineering, IEEE Transactions on*, 32(4):265–278, 2006.