Addressing Safety and Security Contradictions in Cyber-Physical Systems

Mu Sun, Sibin Mohan, Lui Sha and Carl Gunter
Dept. of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61802,
[musun,sibin,lrs,cgunter]@illinois.edu

Abstract

Modern cyber-physical systems are found in important domains such as automobiles, medical devices, building automation, avionics, etc.*. Hence, they are increasingly prone to security violations. Often such vulnerabilities occur as a result of contradictory requirements between the safety/real-time properties and the security needs of the system. In this paper we propose a formal framework that assists designers in detecting such conflicts early, thus increasing both, the safety and the security of the overall system.*

1. Introduction

Embedded systems have permeated into every aspect of day-to-day life, ranging from non-critical systems (televisions or toasters), moderately critical systems (stop lights), to highly critical ones (anti-lock breaks, hydro-electric dam controls, flight control systems, etc.). The latter two categories are examples of cyber-physical systems (CPS) where system control affects human lives or interacts with the environment in general. CPS system requirements have always been required to maintain the highest caliber in terms of timing, reliability, fault-tolerance, *etc.* With the ever-increasing use of such systems, ensuring that CPS are secure from intrusion and tampering by adversaries is the next important design challenge. However, with the many strict design requirements in place adding security into existing systems requires additional thought [8] and can easily create conflicts. These conflicts could result in either *(a)* overly secure systems that compromise the reliability of critical operations or *(b)* create insecure systems where back-doors are easily found.

Consider the following incident (obtained by us as an anecdotal verbal report):

A European luxury car manufacturer noticed that one of its models was a disproportionately likely target for theft. The mystery was solved when an apprehended thief revealed that this model of car easily opened even when its doors were locked. When jumping on the roof of the car, doors would unlock. The designers of the car included a safety feature whereby the doors of the car would unlock if the car was involved in an accident and rolled over. To check for roll-over situations they verified if enough pressure/weight was being applied on the roof.

Hence, *the process of increasing the safety of the system actually decreased the security of the same system.* By providing the ability to get out of a car easily in the case of a catastrophic event the designers also made it easier to break in when there was no accident. Such problems can be avoided by a novel co-design process that we propose here. We need techniques to integrate safety and security as well as the necessary trade-offs during the design phase. Some of these trade-offs may not be evident up front but we need an analysis technique that can alert us to their existence.

In this paper we propose techniques to alert us to *contradictions* between the requirements for safety and security. Typically, various domain experts must come together to define a comprehensive set of requirements for any system. However, often-times experts from differing domains, such as safety and security, are relatively isolated from each other. *E.g.*, safety experts view the system in terms of fault trees, (safety) risk analysis, reliability modeling, *etc.* while security experts will view the system in terms of vulnerability trees, attacker models, *their* version of (security) risk analysis, *etc.* While developing complex systems such as automobiles, avionics and healthcare it is not hard to lose sight of the larger system. It is quite clear that human limitations restrict normal designers from having a comprehensive view of the system. What is needed then is an automated tool to detect when local decisions made at the requirements level adversely impact other parts of the system.

We propose a framework for detecting such conflicts that includes:

1. extensible global language to specify the system and environment
2. mechanisms to specify domain requirements
3. mechanisms to relate requirements across classes
4. find conflicts between requirement classes

We have created a prototype for the underlying formalisms using the Maude, rewriting logic, language [1].

2. Resolving Safety and Security Requirement Conflicts

Consider a building automation system[1]. The problem is setup as having an exit door from some building or room (perhaps a museum or even some top secret facility) that has a status *openable* detailing whether the door can be opened while exiting the building. In Figure 1, this is captured in the *Basic World* block defining a model of the world that is given. Let us now consider this problem from two the different viewpoints of safety and security. A safety designer defines a set of environmental hazards that may influence the system and associate potential risk values to them. Suppose that the safety designer identifies that *fire hazards* must be addressed. This is shown in Figure 1 that contains a *hazard model* in the *Safety World* with one attribute that can be in one of two states, fire or none (*i.e.*, no hazards). Similarly, the security designer will have a different set of goals in mind. Perhaps access across the door must be restricted. Hence, a model of a *lock* must be added to *Security World* as well as a model of a *person gaining access* through the door. Notice that although the safety and security designers do not have a complete view of the system (the "world"), they share a view of the basic world in their separate world views. This is the link between them, since after all they are designing for the same system.

Aside from the modeling objects in a world, certain propositions must be defined in each of the various worlds. Figure 1 illustrates a set of propositions defined in each view of the world. In the *Basic World* the proposition "valid" defines which world configurations are valid so we can ignore unnecessary or impossible models of the world. The proposition "can-open" defines when the door in the system is openable. As an illustrating example, in Maude, we would define this predicate to be true whenever the door object has "openable" as true:

```
eq {C:Configuration < door : Door | openable : true >}
      |= can-open = true .
```

Similarly, for the *Safety World*, we define the proposition "haz-fire" for when a fire hazard exists, and for the *Security World*, we define the propositions "locked" and "authorized" for when the door is locked and when the person is authorized respectively.

The propositions themselves are not interesting aside from labeling the states in the world. However, with these propositions we can now define interesting properties about the world (or a "view" of the world). For each world view there exist a set of assumptions and requirements. Assumptions specify a set of assumed relations between different

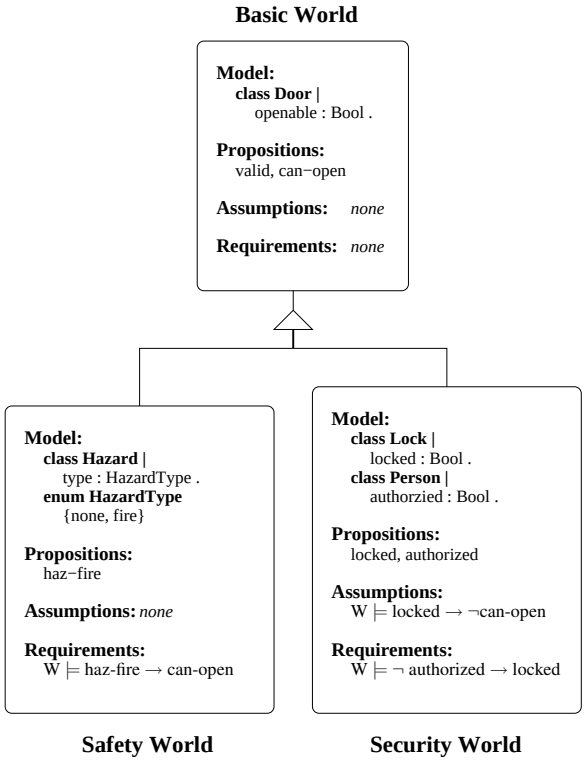

Figure 1: Different Domain Models of the World

propositions so that any world not satisfying these assumptions will not be considered. The world view in our example that lists an assumption is the *Security World*. Naturally, security mechanisms are added to a system with an assumed behavior. *E.g.*, a lock object is added into the security model but it is only useful because of the assumption that a locked door is an unopenable door (specified formally in the assumptions part of the security model). After the assumptions are handled, the most interesting and non-trivial part of design is specifying the requirements. The safety model requirement specifies that in the event of a fire the door should be open. The security model requirement specifies that whenever a person is not authorized the doors should remain locked.

What happens when there is a fire and an unauthorized person is trying to escape the building? It should now be apparent that these requirements conflict. Hence, to automate the process of conflict detection, we must capture this notion of conflict more formally. For short hand, we denote the "Basic World," "Safety World," and "Security World" models as W_0, W_{safety}, and $W_{security}$ respectively. By definition, all models of the world in different domains includes W_0 (*I.e.* $W_0 \subseteq W_{safety}$, and $W_0 \subseteq W_{security}$). Now, we define a combined version of the world that joins all domains: $\widetilde{W} = W_{safety} \uplus W_{security}$ (which just performs the disjoint union on components not common in W_0 with renaming if there are name conflicts). We also define the fol-

1 This problem is similar to that of the car door example mentioned in section 1 and can be easily transposed to the same issue of doors being "open" or "closed," thus leading to the same contradictions between safety and security

lowing propositions for $\widetilde{W}$ ($\models$ is the symbol for *satisfies*):

$$\widetilde{W} \models \text{all-premise} \Leftrightarrow \widetilde{W} \models \text{locked} \rightarrow \neg\text{can-open}$$
$$\widetilde{W} \models \text{safe} \Leftrightarrow \widetilde{W} \models \text{haz-fir} \rightarrow \text{can-open}$$
$$\widetilde{W} \models \text{secure} \Leftrightarrow \widetilde{W} \models \neg\text{authorized} \rightarrow \text{locked}$$

As a sanity check there should be a common notion of normal operation. In our case this would be when all users are authorized and no hazards are present. If defined properly, normal operation should satisfy both safety and security, *i.e.*, the configurations satisfying both safety and security should not be empty.

```
search {choose(init)} =>* W:World
   such that
   W:World |= valid and W:World |= all-premise and
   W:World |= safe and W:World |= secure .
```

If the above search returns no solutions then it means that the safety and security designers had completely different ideas about system operation. Fortunately, in our case, there are five configurations common to safety and security. We present the first solution below:

```
Solution 1 (state 67)
states: 68  rewrites: 2569 ... (12ms real)
W -->
{
< door : Door | openable : true >
< hazard : Hazard | type : none >
< lock : Lock | locked : false >
< person : Person | authorized : true >
}
```

Now, the conflict detection formula for safety and security is just as simple. We search for all configuration that are safe but insecure and also we search for all the configurations that are secure but unsafe. This is done using the following queries:

```
search {choose(init)} =>* W:World
   such that
   W:World |= valid and W:World |= all-premise and
   W:World |= safe and not (W:World |= secure) .

search {choose(init)} =>* W:World
   such that
   W:World |= valid and W:World |= all-premise and
   not (W:World |= safe) and W:World |= secure .
```

The solutions returned from such searches can be of two forms, one more easier to resolve than the other. The easier case is that one domain did not know all the factors in the global configuration. *E.g.*, in the absence of fires the safety experts would agree that a person needs to be authorized before the door is openable. The other case is a *true conflict* an example of which is shown below:

```
Solution 2 (state 72)
states: 73  rewrites: 2949 ... (17ms real)
W -->
{
< door : Door | openable : true >
< hazard : Hazard | type : fire >
< lock : Lock | locked : false >
< person : Person | authorized : false >
}
```

This shows the situation when there exists a fire and an unauthorized person gains access. Neither the safety nor the security designers can yield immediately without more detailed information. *E.g.*, for a school building, the safety requirement would dominate but for a top secret facility, the security requirement would dominate.

The current conflict detection technique is by brute force search through all possible valid models of the world that satisfy the initial assumptions. Although inefficient, this provides a systematic and exhaustive way of reminding different design domains about potential conflicts and, ultimately, serving its purpose as a communication mechanism between different designers to detect impacts of their requirements (and related changes) on the rest of the system.

There is the potential concern that too many conflicts will be detected that overwhelm the different designers beyond comprehension. On a pragmatic level, we believe that any kind of assistance is better than no assistance. On a technical level, using a compiler analogy, a programmer never really reads through all the potentially daunting list of compile-time errors – he/she just looks at the first error, fixes it and then recompiles; most of the time this removes multiple bugs (*e.g.* in the case of missing semicolon). We believe that requirement conflicts should be treated in the same way. Many requirement conflicts may be detected but when the designer addresses a specific conflict the solution may resolve a whole *class* of conflicts.

2.1. Proposed Framework

From the above example, we can hypothesize a general framework for specifying domain models and requirements and finding conflicts. We define a $world$ as a 4-tuple consisting of the model (classes and objects), propositions (and definitions), assumptions, and requirements: $W = (M, P, A, R)$. We start with an initial, common world and each new domain adds its view to the world. Thus, if the initial world is W_0 then each new domain D_i extends the initial world W_0 into W_{D_i} (where $W_0 \subseteq W_{D_i}$). For analysis we consider a joined world $\widetilde{W}$ where $\widetilde{W} = W_{D_1} \bowtie W_{D_2} \bowtie ... \bowtie W_{D_n}$ is the join of n different domain views of the world. At this point, we speculate that the general method of obtaining a joined world is the disjoint union of the domain specific model components as in the example (*I.e* $\bowtie = \uplus$). However, regardless of the definition for $\widetilde{W}$ it is necessary that a family of unique projection functions π_{D_i} exist such that $\pi_{D_i}(\widetilde{W}) = W_{D_i}$ (*i.e.* we do not loose domain information when merging domains).

With these definitions the process of detecting conflicts between two domains D_i and D_j is reduced to finding any world models $w \in W$ such that $w \models \widetilde{A} \wedge R_{D_i} \wedge \neg R_{D_j}$ or $w \models \widetilde{A} \wedge \neg R_{D_i} \wedge R_{D_j}$ where $\widetilde{A}$ denotes all assumptions in $\widetilde{W}$. When all possible conflicts between two domains have been resolved by removing some requirements

and/or adding other requirements to the world with models $M_{D_i} \bowtie M_{D_j}$, we would have created a merged world $W_{D_{i,j}}$ where all requirements are consistent. Furthermore, since the process of resolving conflicts merges two domains it suggests an algorithm for removing all inconsistencies by analyzing domains in a pairwise manner by merging two domains at a time.

Currently, our framework identifies conflicts for designers who may then proceed to resolve them. Identifying a problem is a necessary and important step before solving it. However, we have not considered how to categorize such conflicts in terms of importance and criticality. Furthermore, it may also be possible to have automated tools that handle the conflicts. These are a few of the directions we plan to explore in the future.

3. Related Work

The closest work in integrating security into safety-critical CPS is our previous work that proposes solutions on integrating security policies into the real-time task framework [8]. Other existing work has explored the duality between safety and security. Simpson [10] applied the security concept of noninterference to analyze safety properties of communication networks. Stavridou [12] also found similar relations by applying noninterference and well known safety techniques of fault tolerance for intrusion detection. Given the similarities between safety and security, Eames [3] and Stoneburner [13] have presented methodologies for analyzing safety and security concurrently by proposing a merged risk analysis process. Novak [9] analyzed these methodologies in even more detail by presenting an integrated safety and security life cycle process for design of building automation systems. Smith [11] has also presented similar integration ideas in rail communications systems. All of these techniques are complementary to the ideas proposed in this paper since our framework analyzes *conflicts* in the *requirements* using a formal process. Other techniques can then be applied to improve the overall safety/security of the system.

We used the example of building automation to illustrate our proposed methods since it a relevant problem. However, it will be in new applications domains where safety and security conflicts are still uncertain that our techniques will be most useful. Representative publications [6, 7] have brought to light many of the security factors in domains such as power grids and smart vehicles. Recently, the security risks of implanted medical devices [5], especially pacemakers [4], were brought to the attention of the community. Denning [2], has proposed solutions in resolving the specific safety and security conflicts such as information access in medical devices. As more security issues are identified for medical devices, many new and subtle requirement conflicts may result and resolving these will be key towards safe and secure medical devices in the future. This is where our work will be able to provide key insights to the security/safety requirements in the domain.

4. Conclusion

In this paper we propose techniques that can be used to detect conflicts between requirements in differing design domains. We especially care about safety and security as they are vital to most modern CPS domains. We propose a framework that helps alleviate contradictory requirements between these two domains by allowing safety and security specifications to remain decoupled initially. An automated tool then detects the truly coupled requirements among the two domains, thus, providing system designers information on problems that would have been difficult to detect by manual processes alone.

References

[1] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. The Maude 2.0 system. In R. Nieuwenhuis, editor, *Rewriting Techniques and Applications (RTA 2003)*, number 2706 in Lecture Notes in Computer Science, pages 76–87. Springer-Verlag, June 2003.

[2] T. Denning, K. Fu, and T. Kohno. Absence makes the heart grow fonder: New directions for implantable medical device security. *3rd USENIX Workshop on Hot Topics in Security (HotSec '08)*, July 2008.

[3] D. P. Eames and J. D. Moffett. The integration of safety and security requirements. In *SAFECOMP '99: Proceedings of the 18th International Conference on Computer Computer Safety, Reliability and Security*, pages 468–480, London, UK, 1999. Springer-Verlag.

[4] D. Halperin, T. Heydt-Benjamin, B. Ransford, S. Clark, B. Defend, W. Morgan, K. Fu, T. Kohno, and W. Maisel. Pacemakers and implantable cardiac defibrillators: Software radio attacks and zero-power defenses. In *Security and Privacy, 2008. SP 2008. IEEE Symposium on*, pages 129–142, May 2008.

[5] D. Halperin, T. Kohno, T. Heydt-Benjamin, K. Fu, and W. Maisel. Security and privacy for implantable medical devices. *Pervasive Computing, IEEE*, 7(1):30–39, Jan.-March 2008.

[6] J. Hubaux, S. Capkun, and J. Luo. The security and privacy of smart vehicles. *Security and Privacy, IEEE*, 2(3):49–55, May-June 2004.

[7] W. Johnston, D. Gannon, and B. Nitzberg. Grids as production computing environments: the engineering aspects of nasa's information power grid. In *High Performance Distributed Computing, 1999. Proceedings. The Eighth International Symposium on*, pages 197–204, 1999.

[8] S. Mohan. Worst-case execution time analysis of security policies for deeply embedded real-time systems. *SIGBED Rev.*, 5(1):1–2, 2008.

[9] T. Novak, A. Treytl, and P. Palensky. Common approach to functional safety and system security in building automation and control systems. pages 1141–1148, Sept. 2007.

[10] A. Simpson, J. Woodcock, and J. Davies. Safety through security. In *IWSSD '98: Proceedings of the 9th international workshop on Software specification and design*, page 18, Washington, DC, USA, 1998. IEEE Computer Society.

[11] J. Smith, S. Russell, and M. Looi. Security as a safety issue in rail communications. In *SCS '03: Proceedings of the 8th Australian workshop on Safety critical systems and software*, pages 79–88, Darlinghurst, Australia, Australia, 2003. Australian Computer Society, Inc.

[12] V. Stavridou and B. Dutertre. From security to safety and back. In *CSDA '98: Proceedings of the Conference on Computer Security, Dependability, and Assurance*, page 182, Washington, DC, USA, 1998. IEEE Computer Society.

[13] G. Stoneburner. Toward a unified security-safety model. *Computer*, 39(8):96–97, Aug. 2006.