A Systematic Security Analysis of Real-Time Cyber-Physical Systems*

Arvind Easwaran, Anupam Chattopadhyay

School of Computer Science and Engineering
Nanyang Technological University, Singapore
e-mail: {arvinde,anupam}@ntu.edu.sg

Shivam Bhasin

Temasek Laboratories
Nanyang Technological University, Singapore
e-mail: sbhasin@ntu.edu.sg

Abstract— Security in Cyber-Physical Systems (CPS) has become a serious concern owing to the rapid adoption of technologies such as plug-and-play connectivity, robotics and remote coordination and control. It is well understood that the performance overhead incurred due to security considerations is rather high, which needs to be captured holistically for a real-time CPS with strict timing budget and hard deadlines. Additionally, attacks in real-time CPS may only alter the timing behaviour of system components without any changes in functionality, resulting in serious consequences due to missed deadlines. To address this challenging issue, it is necessary to understand the role of diverse components in a real-time CPS and how those expose the system to a malicious attacker. In this paper, we propose a systematic security analysis flow, using a novel *Attack Sequence Diagram (ASD)*, which links the sources, intermediate components and final manifestations of an attack, thereby clearly delineating the attack surfaces of a complex real-time CPS. Based on the ASD, it is possible to evaluate the complexity of an attack, performance overhead of a countermeasure and explore different design trade-offs for a real-time CPS. With the help of real-world and synthetic examples, we demonstrate that ASD seamlessly enables one to map the existing vulnerabilities and uncover new attack possibilities.

I. Introduction

Cyber Physical Systems (CPS) undoubtedly occupy the most prominent place in today's technology spectrum. The class of CPS which are governed by strict timing deadlines, are referred to as real-time CPS. Depending on the criticality of the application and the task deadlines, it can be further categorized as *hard* or *soft* real-time CPS. Few examples of such real-time CPS are Anti-lock Breaking System (ABS) in automotive, temperature/pressure control systems in manufacturing plants, laboratory robotics, and flight controls in avionics.

In many application scenarios for real-time CPS, security of the system is a growing concern. This is largely driven by the trend in these systems towards large-scale integration and remote co-ordination and control. Although an open operating environment for a hitherto closed system offers potential for performance improvement, it also exposes several new attack surfaces in the system. Therefore, in recent times researchers have taken a strong interest in security analysis of real-time CPS.

A secure CPS aims toward satisfying the fundamental criteria of security, namely, *confidentiality* of data, *integrity* of data/control, *authenticity* of user access, and *availability* of resources. However, the prioritization of these criteria is highly domain-specific. The complexity of a real-time systems' stringent timing constraints adds further to the challenge of considering security as a design metric. Countermeasures to well-known attacks result in performance overhead that can affect timing guarantees due to reduced resource availability. Also, attacks that alter the timing behaviour of system components have a significant impact on system performance even if they do not alter its functional behaviour. Therefore, in such systems, analysis of security issues at an early design phase undoubtedly helps the designer. Along this line, it is suggested in [21] to work on a language, or a feature of it, to let the designers work on the security enhancement in parallel with other requirements of CPS.

In order to define the security issues, the first task faced by a system designer is to identify the possible attack scenarios. An attack scenario consists of two phases. First, it introduces the attack through a component in the system. Second, the attack propagates through the system to manifest itself in a discernible form in the output. Within the first phase, one needs to identify an attack technique, e.g., passive/active attack, and also needs to pinpoint the component that is used to initiate the attack. So far, to the best of our knowledge, there is no formal approach to determine different attack scenarios in real-time CPS, though such methodologies are not uncommon in studying attacks in computers or network systems [1], which forms the key motivation of this work.

In this work, we argue that the most important observable security breach in a real-time CPS is caused by the *malicious* violation of task deadlines. Depending on the application scenario, this can result in seriously damaging consequences. More importantly, this is also a design property that is captured well in the early design phase and therefore, can be analysed well from the perspective of security.

Contribution. The key contribution of this paper is introduction of Attack Sequence Diagrams (ASDs), which is an extension of the sequence diagrams used for describing real-time systems. Using ASDs, it is possible to identify the complete attack scenario in a highly intuitive manner. This is demonstrated using multiple illustrative examples. ASDs describe attack scenarios in the form of task-resource utilization traces. Thus mapping attacks as an external/internal trigger, ASDs have the potential to be automated and further integrated in the verification process at an early design phase.

The rest of the paper is organised as follows. In Section II the generic structure of a real-time CPS is presented. Section B

*This work was funded in part by MoE Tier-1 grant RG21/13 and MoE Tier-2 grant ARC9/14, Singapore.

978-1-5090-1558-0/17/$31.00 ©2017 IEEE

contains our key contribution, the discussions on attack characterization and the proposition of attack sequence diagram, followed by several illustrative examples for the same. State-of-the-art literature in the security of real-time CPS are discussed in Section IV. Section V concludes this paper and identifies several research directions.

II. Real-Time CPS: Generic Structure

Figure 1 shows some of the central components of a real-time CPS. This structure is representative of safety-critical systems such as Integrated Modular Avionics (IMA) [12] and AUTOSAR for automotive [13]. It comprises

1. Real-time applications and tasks at the user level,
2. Real-time scheduler and resource-sharing protocol in the operating system kernel, and
3. Execution platform with peripherals and a communication network at the hardware level.

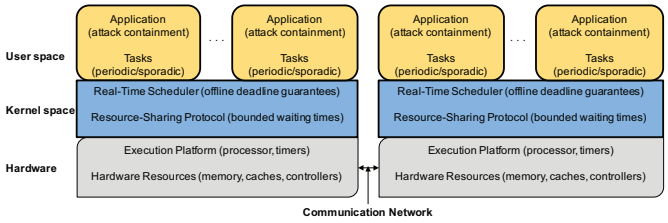


Fig. 1. Generic Structure of a Real-Time CPS

In the following text we discuss in detail the requirements, related to both timing as well as security, for each of these components.

Real-Time Applications. Each real-time application represents a system functionality, and a collection of such applications may be hosted on a hardware platform. These co-hosted applications have stringent *non-interference requirements* to ensure attack and fault containment, because they may represent functionalities having different importance (e.g., Design Assurance Level in avionics [14]) and security (e.g., Multiple Independent Levels of Security or MILS [15]) levels. This means the allocated hardware resources must be appropriately partitioned among them (e.g., time- and space-partitioning in avionics [16]).

Real-Time Tasks. Each real-time application comprises a set of real-time tasks. Each task can be either *periodic* if instances of the task are released for execution periodically using a time-triggered mechanism, or *sporadic* if instances are released using an event-triggered mechanism. In the case of sporadic tasks, it is further assumed that there is a minimum separation between releases of successive instances so that it becomes feasible to provide guarantees on deadlines. Thus, a task is represented using the tuple (T, C, D), where T denotes the period or minimum separation between successive instances, C denotes the worst-case execution time of each instance, and D denotes its relative deadline. Each instance of the task is required to execute on the processor for as much as C time units within D time units from its release. C depends on a lot of factors including state of hardware resources, waiting time to access them and software control- and data-flow. It is very challenging to estimate C accurately because of the complexities involved, and therefore pessimistic upper bounds are often used.

Real-Time Scheduler. The real-time scheduler prioritises allocation of the processor to task instances so as to meet deadlines. Additionally, it is also responsible for partitioning the processor allocations among applications to ensure non-interference. Several algorithms have been developed for both single- and multi-core processors. These can broadly be classified into either fixed-priority scheduling in which priorities are statically allocated to tasks based on their parameters, or dynamic-priority scheduling in which priorities are allocated to task instances at runtime.

An important aspect of real-time schedulers is the offline guarantees that they provide on deadlines using schedulability tests. These tests are used to ascertain whether a given set of tasks and applications will meet their deadlines when scheduled under a given algorithm. The correctness of these tests is fundamentally dependent on the assumed parameters of the tasks and applications as well as hardware state (e.g., clock speed of the processor, state of cache, etc.). Any violation of these parameters or hardware states at runtime has the potential to invalidate the test and consequently the deadline guarantees.

Resource-Sharing Protocol. Access to shared platform resources such as global variables and I/O devices must be regulated using a protocol so as to ensure data consistency and fairness across applications and tasks. Additionally, in real-time systems, these protocols must also ensure that waiting times (i.e., time spent waiting to access a shared resource) are bounded so that deadlines can be guaranteed. Further, it is important that the protocols reduce *priority inversions*, that is time for which a higher-priority task is waiting to access a shared resource, while lower-priority tasks are executing on the processor. With the proliferation of multi-cores, such protocols could also be used to regulate access to shared hardware resources such as the processor interconnect and memory controller so that a bounded waiting time may be achieved.

Execution platform. Important components of the execution platform include the processor (single- or multi-core), the hardware timers and the interrupt handling mechanism. A real-time scheduler uses timers and interrupt handlers to enforce the task parameters at runtime. These timers are used to trigger execution of the scheduler so that periodic tasks can be released as required and task and application processing budgets can be enforced based on their parameters. This ability of the scheduler to interrupt application processing at pre-defined time instants is essential to ensure timing correctness of the system.

Other hardware resources. The communication network in a real-time system is required to provide service with bounded jitter and taking into consideration message deadlines. Since a system functionality may be distributed across several hardware platforms in the network, it is also essential to meet end-to-end latency requirements for applications.

With the growing popularity of multi-core processors, shared hardware resources such as caches, cache and memory controllers and processor interconnect also play a critical role in the design of a real-time CPS. It is essential to ensure either non-interference or bounded interference from tasks and applications executing in the other cores when accessing these

resources. Only then task worst-case execution time and application non-interference requirements can be satisfied.

III. Attack Characterization

In this section we systematically characterise potential attacks in real-time CPS and the impact they have on system requirements. We consider the critical components in kernel space and hardware presented in Section II, and identify potential sources of attacks as well as impact of attacks on task and application deadlines.

A. Attack Considerations

A key parameter which separates real-time CPS from generic systems is the importance of deadlines. Real-time systems are designed in a way that tasks should always be executed within the given deadline, failing which could lead to serious consequences. For the same reason, the worst-case execution time parameter C for tasks is pessimistically computed with huge safety margins to account for execution time variations due to complex software and hardware. Note that a violation of the worst-case execution time estimate at runtime has the potential to lead to a deadline miss, particularly if the system has a high processor utilization. Further, such deadline misses can trigger a chain reaction or overload condition in which several subsequent deadlines are also missed. Given the importance of deadlines and worst-case execution times in real-time CPS, in the following *we consider an attack as any event that results in a missed deadline for a real-time task*, and refer to it as a *deadline attack*.

Deadline attacks are necessarily active attacks because they must alter the system timing behavior. Therefore, in this section we consider attacks that compromise integrity or availability or both. Attacks that compromise the integrity of scheduler or resource-sharing protocol (e.g., changes to task parameters, task priorities, protocol behavior) have a direct impact on resource allocation and waiting times, and hence are important in our study. Similarly, attacks that compromise the availability of hardware and software resources (e.g., I/O and memory controllers, caches, access to global variables) also have an impact on the timing behavior and hence are part of our study.

Deadline attacks could originate either in the user space from a compromised task, or in the kernel space from a compromised operating system, or in hardware. However, given the fact that physical access to these safety-critical systems is not easy (e.g., access to adaptive cruise control system in automotive), attacks originating from a compromised task or application are more likely to occur. This is further supported by the recent trend in such systems wherein certain non-critical applications can be directly accessed by untrusted and unverified systems (e.g., wireless or wired connection of personal devices to the infotainment system in an automotive). Since these non-critical applications have access to some of the resources that are also used by critical applications (e.g., communication network), there is the possibility of an attack path originating in the non-critical application but terminating in a critical application.

B. Attack Sequence Diagram

In this work we present *Attack Sequence Diagrams (ASDs)* as an extension of standard sequence diagrams to capture and illustrate deadline attacks. This is because sequence diagrams are good for modeling resource requests from real-time tasks, resource allocation by scheduler and resource-sharing protocol, and temporal properties such as resource usage time and waiting time.

A standard sequence diagram comprises a set of objects communicating with each other using messages and responses. In the context of an ASD for a real-time system, these objects are applications and tasks (denoted as *users*), scheduler and resource-sharing protocol (denoted as *kernel components*) and hardware and other elements (denoted as *resources*). Resources include global variables, I/O controllers, memory and cache controllers, caches, processing cores, communication network, etc. Users send messages in the form of resource *requests* to the kernel components, and they respond to these requests by allocating resources to the users so as to meet their requirements. Note that hardware resources such as timers and interrupt handlers are internally used by kernel components to manage allocation to the other resources, but are not directly allocated to users. Attacks that can be identified by ASD manifest in the form of delayed access to a resource for a user resulting in a deadline miss eventually. The source of these attacks could be either compromised kernel components or users using resources beyond their budgeted allocation. The kernel components could be compromised either due to direct integrity attacks on their parameters and data or due to indirect attacks through the hardware resources used by them.

Latronico and Koopman [24] have considered the problem of generating sequence diagrams for embedded systems that can be automatically translated to deterministic state charts for further formal analysis. They proposed addition of state information for objects, pre-conditions on message parameters, and timing constraints between successive messages as additional sequence diagram constructs to achieve this determinism. Among these three constructs, we will use state information for kernel components and resources in our ASDs, because their requests and response to requests depend on it. Further, upon receipt of a request this state will be updated immediately (to capture the new pending request), and therefore preconditions and timing constraints between successive requests are not necessary to achieve determinism.

Definition 1 (Attack Sequence Diagram (ASD))**.** *An Attack Sequence Diagram is defined as* $D = \langle \mathcal{O}, \mathcal{R}, \mathcal{S}_r, \mathcal{S}_k, \mathcal{A} \rangle$*, where*

- $\mathcal{O}$ *denotes a set of objects (users, kernel components and resources),*
- $\mathcal{R}$ *denotes a set of requests and response to requests,*
- $\mathcal{S}_r$ *denotes a set of states for each resource* $r \in \mathcal{O}$*,*
- $\mathcal{S}_k$ *denotes a set of states for each kernel component* $k \in \mathcal{O}$*, and*
- $\mathcal{A}$ *denotes a set of deadline attacks.*

Each deadline attack can be one of the following types.

1. ***State attack***: *In this attack, the state of a kernel component or resource is modified at some time instant.*
2. ***Response attack***: *In this attack, the response from a kernel component or resource is modified at some time instant.*

Both these attacks can be defined using a time-dependent mapping function from one state/response to another (e.g., the attack function for resource states would be of the form $\mathcal{F} : \mathcal{S}_r \times \mathcal{T} \rightarrow \mathcal{S}_r$*, where* $\mathcal{T}$ *denotes a set of time instants).*

A state attack can either alter a subsequent request or response from that kernel component/resource, or simply delay the request or response. Both these consequences can eventually lead to a deadline miss for a task. Similarly, in a response attack, although there is no modification to any state, an altered response can change the resource allocation pattern eventually resulting in a deadline miss. In the following subsections, we model example attack scenarios using ASDs.

C. Modeling an Existing Attack

In this section we illustrate a simple attack on the integrity of the priority parameter of a scheduler. Any changes to the assigned priority of tasks has a significant impact on the processor allocation, and hence tasks are very likely to miss their deadlines, especially on systems in which the processor is heavily utilized. This attack was first presented in [18], and here we illustrate the same using an ASD.

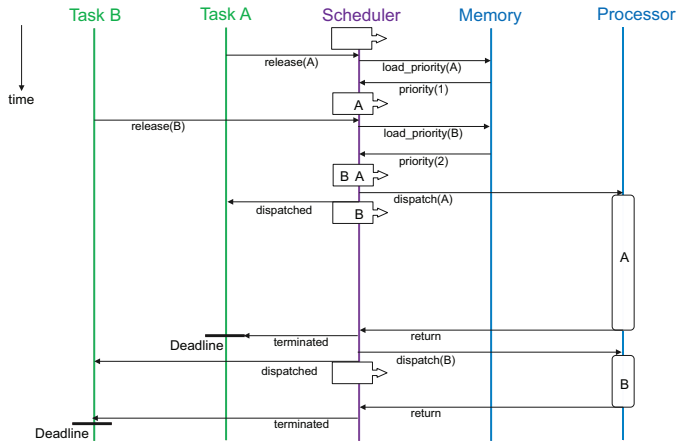


Fig. 2. Scheduler priority management system

Example 1. *Figure 2 shows an ASD comprising two users (tasks A and B), one kernel component (scheduler), and two resources (memory and processor). When a task requests for a release, the scheduler obtains the priority of the task from a pre-determined kernel memory location. In the figure, a lower number denotes higher task priority. It uses this priority to maintain a priority-queue (state of scheduler) and dispatches tasks to the processor accordingly. As shown in the figure, task A has higher priority than task B. Thus, assuming the priority management system is not compromised, task A is first dispatched to the processor followed by task B, and both tasks are able to meet their deadlines. Note that not all component and resource states are shown in the figure; only those that are relevant for the attack under consideration are shown.*

Considering the above example and the ASD from Figure 2, we try to identify some vulnerabilities in the system which a potential attacker can target to force the system to miss deadlines. We show, in particular, one example of a *response* attack and a *state* attack. In the above example, task A requested highest priority (1). The priority value is retrieved by a call to a predefined memory location. An attack to this system can be realized by disturbing the priority load instruction. There are numerous methods which an attacker could use to achieve this. Some common methods are overriding the memory bus [28], or forcing memory address change [25], or flipping bits in memory location [27]. While the former two are examples of response attacks, the latter is an example of state attack. Note that the attack reported in [27] only requires user-level privileges in the OS to affect the memory locations. Depending on the technique and expertise at hand, the precision of the attack will be different however, all of these methods can disturb the priority load instruction. In the present case, if the priority of task A is lowered from 1 to 3, it will be executed after task B, leading to a deadline miss as shown in Figure 3(a).

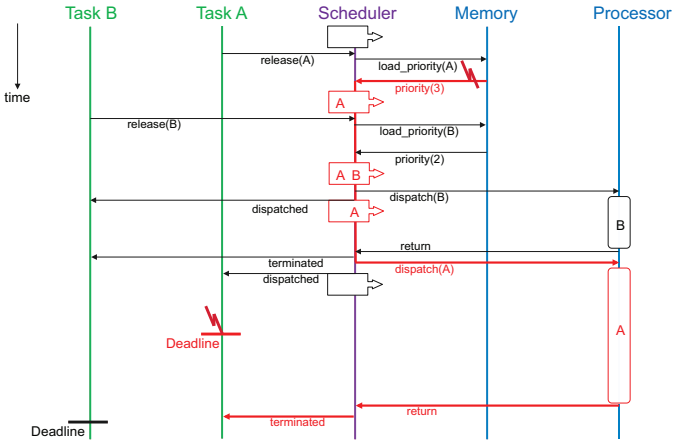
Continuing towards identifying other vulnerabilities from the ASD, one can also spot a potential *state* attack. If the *return* signal after completion of task A is delayed by the processor (a state attack), the scheduler will not free the processor leading to a delay in the execution of subsequent tasks, resulting in a deadline miss. This can be achieved by manipulating the clock input via electromagnetic (EM) injection [29]. An EM injection can stall the oscillator or also temporarily modify clock paths. The attack scenario is illustrated in Figure 3(b).

D. New Attack Propositions

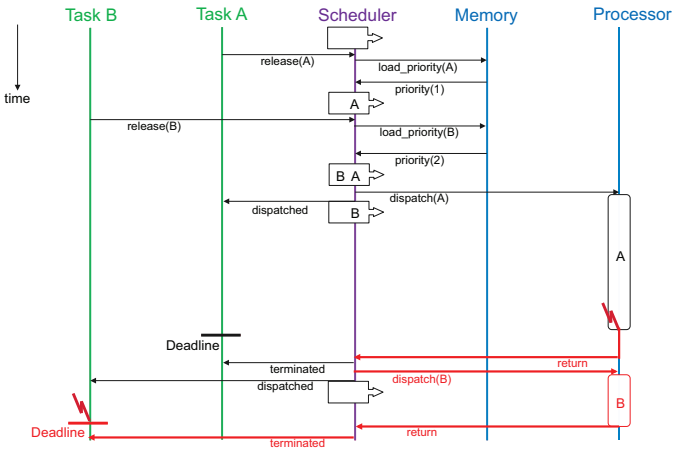
In this section we illustrate a few non-trivial attacks targeting both integrity as well as availability using a typical real-time system comprising different applications. One of the applications implements a critical functionality, whereas the other implements a less-critical functionality. We further assume that the less-critical application is relatively easy to compromise compared to the critical application, due to reduced rigor in its verification process.

D.1 Representative Real-Time CPS

The system shown in Figure 4 comprises two tasks belonging to different applications and executing functions with varying criticalities. It also comprises two kernel components (scheduler and resource-sharing protocol) and four resources (timer, two cores on a multiprocessor platform, and memory hierarchy). Memory hierarchy can be viewed as an abstraction for all the hardware resources involved in a memory transaction; namely caches, cache and memory controllers and the main memory itself. On a multiprocessor platform, it is important to note that these hardware resources in the memory hierarchy are shared among the two cores, so that only one core may access each such resource at a single time instant.



(a)



(b)

Fig. 3. Examples of *response attack* (a) and *state attack* (b) on the priority management system

Upon release of a task, the scheduler uses the timer to monitor the budget allocation for that task. Once the allocated budget is consumed, the timer is triggered and the task is either terminated (if it has completed) or an exception handler may be invoked otherwise. The scheduler then dispatches the tasks to the two cores depending on its scheduling policy and priority assignment.

While executing on the processor, the tasks request for access to data in memory in parallel; this means the two memory requests are pending in the various hardware resources of the memory hierarchy at the same time. In particular, it is possible that the request from the critical task may be delayed due to the pending request from the less-critical task.

A little while later, both the tasks request for access to the same shared resource (e.g., I/O controller) almost at the same time instant. Based on the protocol and relative task priorities, the less-critical task is given first access to the shared resource. It is important to note that while this task is accessing the shared resource, the critical task is blocked because it is waiting to get access to the same resource.

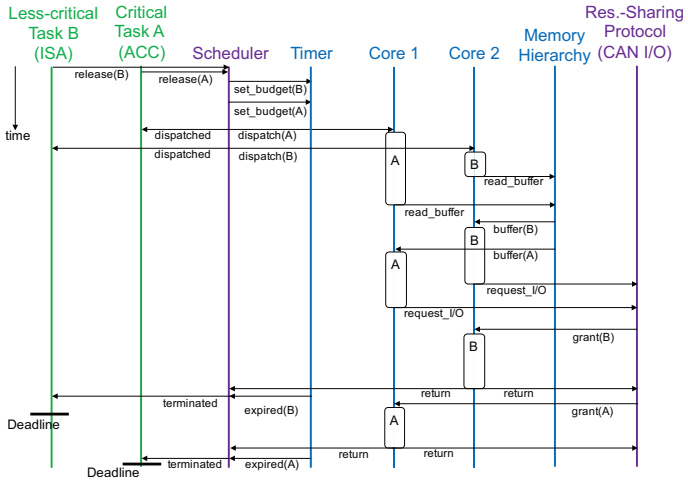


Fig. 4. A representative real-time system comprising two applications

Example 2. *As a representative example, we can consider the critical application Adaptive Cruise Control (ACC) and the relatively less-critical application Intelligent Speed Advisory (ISA) in automotive CPS. ACC receives position and velocity inputs from the lead as well as controlled vehicle, and determines a safe acceleration/braking value to be sent to the actuators. ISA only receives position and velocity input from the controlled vehicle, and issues speed warning messages to prevent over-speeding. In the system shown in Figure 4, the critical task belongs to ACC, the less-critical task belongs to ISA and the resource-sharing protocol manages access to the Controller Area Network (CAN) device. Initially, both these tasks read inputs from independent buffers in memory. Once the output is determined, they seek access to the CAN device via the protocol. Actuation commands are then issued by the tasks once they have access to the CAN device.*

D.2 Attack Scenarios

A possible attack exploiting the physical layer is shown in Figure 6. This attack shows how a real-time system can be forced to miss a deadline of a critical task by exploiting a less critical task. It targets physical layer by gaining access to physical parameters like power supply or clock. Note that access to physical parameters does not mean physical access to the system. In various systems, these parameters can be accessed remotely, originally to allow better and enhanced control.

If the power line or clock to a processor is disturbed, it will have a direct impact on the execution of the task as demonstrated with physical access to the system [25]. Corresponding OS-level techniques are discussed here [26]. A straightforward scenario is to drive a processor at lower frequency/voltage through a malware, which will increase the signal delays, leading to increased execution time. Since a system on chip isolates power and clock lines for various components, it is possible to affect only a single core of a multi-core processor while having no impact on other components. This is illustrated by an abstract system in Figure 5.

Following Figure 4, once scheduled, tasks A and B are dispatched to different cores for execution and the processor execution is independent. However, during the execution, the

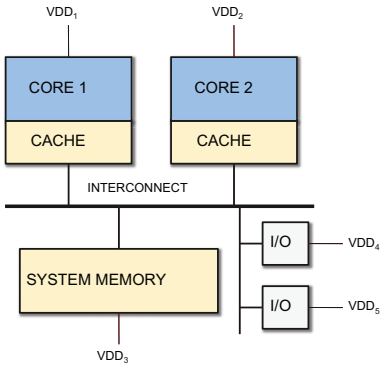


Fig. 5. Multi-core system with separated power lines

tasks need access to shared resources. We demonstrate an active attack on this system by lowering the power supply (under-powering) to the processor core executing task B. If the voltage is not lowered drastically, the core can still operate but will be slower, resulting in higher execution time. This can be realized in Figure 5 by lowering VDD_2. The execution time of task B is increased as shown by the shaded box in Figure 6. This delay is not accounted as worst-case execution time estimates are done at nominal operating conditions. The impact of the increased execution time is as follows. At some point of time, both tasks A and B need access to a shared resource, in this case the I/O device. Since task B already has a handle on the device, task A must wait for the resource. Since the execution time of task B has increased, the I/O device will be available to task A after an unaccounted delay. This delay will force task A to miss its deadline. A smart attack can adjust the delay in a way that task B still makes the deadline while compromising task A, and thus avoiding health checks.

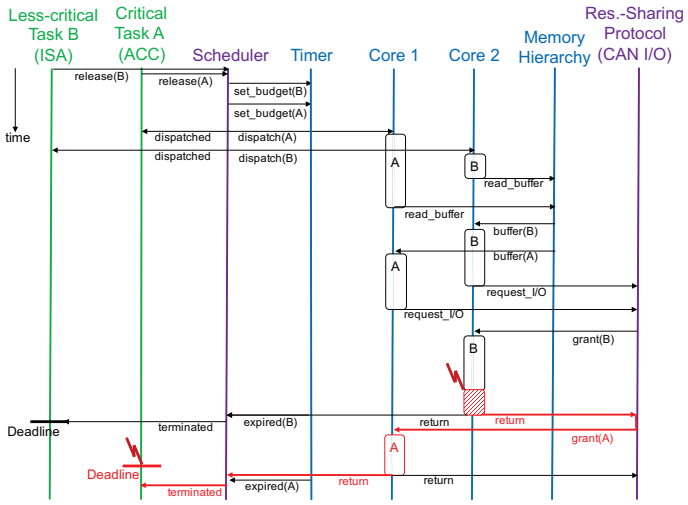


Fig. 6. Example state attack by under-powering task B's execution

An integrity attack on the task budget enforcement mechanism is shown in Figure 7. The scheduler assigns a dedicated budget to each task based on its state parameters. In the proposed attack, the budget assignment of task B (ISA) is altered to a lower value through a response attack on the "set_budget(B)" instruction. Since the budget expires before the task execution has terminated, an exception handler (E) is launched. Due to the exception, task B occupies the I/O device longer, because the device is only released after the exception is processed. This delay leads to a deadline miss. This attack can be realized by techniques like in [27, 25, 28]; the active manipulation can alter the assigned budget value.

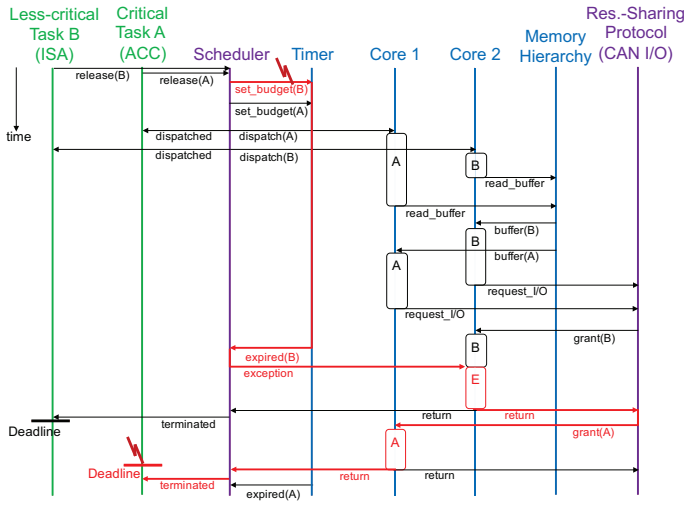


Fig. 7. Example response attack targeting task B's budget enforcement

Unlike previous examples, a deadline attack can also be performed using purely software techniques. An example of such an attack on availability is shown in Figure 8. It is a *state* attack realized by malicious memory access by the less-critical task B infected with a malware. This malware triggers unaccounted memory accesses for the task and occupies the shared resources in the memory hierarchy. The critical task A is consequently delayed and would have to wait for a longer time to get memory access. However, since this additional memory occupation is unaccounted, it would not be considered when determining the worst-case execution time of task A, leading to a deadline miss.

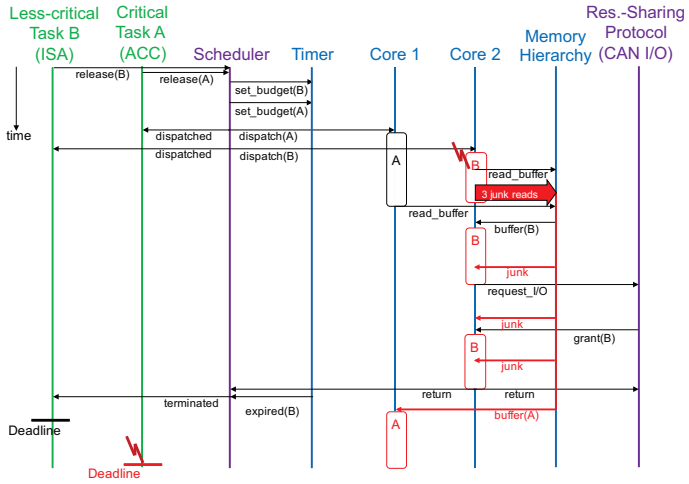


Fig. 8. Example state attack targeting memory through a malware in Task B

These examples illustrate that ASD offers an intuitive approach at the early design phase for security analysis. This can be aided with an automated tool-flow to identify all the potential attack sequences and vulnerable components.

IV. Related Work

Addressing security issues in the context of real-time CPS is now gaining relevance in the research community. There have been several studies in the recent past [11, 10, 5, 6, 2, 3, 4, 19, 20, 22, 17], all of which focussed on issues related to *confidentiality*. They either developed countermeasures while ensuring satisfaction of timing requirements or reduced the impact of countermeasures on resource utilization through modifications in the design.

Compromised Confidentiality. Völp et al. [10] considered the problem of information leakage in the form of state-dependent access times for shared resources such as disks. They proposed techniques to reduce variability in the access times for such resources through modifications in the resource-sharing protocols, while still preserving the required worst-case waiting time bounds to guarantee application deadlines. Pellizoni et al. [6, 2] considered the problem of information leakage through shared hardware resources such as caches, memory controllers and I/O interfaces. They focused on a generic vendor-driven security model in which real-time applications developed by different vendors had constraints on information sharing represented in the form of an asymmetric binary relationship. Considering a simple countermeasure in the form of state clean-up (e.g., cache flushing, delay in I/O interface, etc.), these studies presented techniques to analyse the impact on schedulability of the real-time applications due to these countermeasures. They also proposed design methodologies for allocating priorities to the real-time applications so as to minimize this impact on schedulability, while still preventing information leakage.

Side Channel Attacks and Schedule Randomization. The other group of studies focus on side-channel attacks (e.g., differential power analysis), and present countermeasures to reduce the predictability of the system schedule [11, 3, 4, 5]. The objective there is to sufficiently randomize the schedule so that side-channel attacks on specific applications become difficult, while at the same time ensuring that all application deadlines are met. These studies quantify the robustness of a schedule to such attacks using metrics that measure predictability of the schedule. They also design runtime scheduling techniques that meet application deadlines and at the same time increase robustness based on these metrics. These methodologies basically deploy a shuffling countermeasure to reduce leakage [23]. This would lead to reduction in correlation between sensitive data and leakage by a factor of $\sqrt{n}$, where n is the shuffling order or number of different shuffled task schedules possible. Note that this lowered correlation does not necessarily prevent an attack, but only increases the complexity by a polynomial order. Thus, the practical viability of these approaches remains questionable. There are also some related studies that utilize the inherent predictability of real-time systems to detect anomalous behaviors (e.g., see [7, 8, 9]).

Countermeasures. There have also been studies on the design of resource-efficient cryptographic techniques for secure communication in real-time systems [19, 20, 22, 17]. While [19, 22] proposed software-based techniques with guarantees on the satisfaction of task deadlines, the work in [20] proposed a hardware/software co-design approach to minimize the impact on hardware resources. Wen et al. [17] considered fault injection attacks on such cryptographic algorithms, and evaluated different fault detection schemes for Advanced Encryption Standards (AES) in terms of their timing overheads. Moro et al. [18] have evaluated software-based countermeasures to attacks on a real-time operating system using an electromagnetic fault injection technique. Specifically, they consider instruction skip attack on an instruction that changes the execution mode from privileged to non-privileged and load instruction corruption attack on instructions that load the task priorities. Their focus was on evaluating the ability of the countermeasures to prevent faults, but not on the impact these faults have on system timing requirements.

Specifically within the context of real-time CPS, the works done so far concentrated on particular attack scenarios and countermeasures. Clearly, there is a pressing need of research attention towards detecting real-time system security vulnerabilities in the early design phase. This is the key idea presented in this work.

V. Conclusion and Road map

Real-time cyber-physical systems govern a large part of our everyday lives, ranging from (semi-)autonomous vehicles to robotics and avionics. Security considerations are taking a prime spot in the design and execution of real-time CPS. Given the variety of the application scenarios and attack models, it is becoming an increasingly harder job to address the security concerns. In this paper, we described a generic real-time CPS structure and subsequently, introduced the concept of Attack Sequence Diagrams (ASDs). ASD provides a high-level and yet, accurate description of potential attack surfaces, which we show by mapping existing attacks as well as uncovering new attack possibilities.

This work can be extended in many important directions, as described in the following.

- From the formal ASD notation, it is straightforward to derive a graph-based representation with the dependencies between tasks, components and schedules. This can be subjected to automated analysis for discovering potential new attacks, relative vulnerability of the components as well as low-overhead countermeasures.
- The new attack propositions pointed out in this paper can be demonstrated with practical systems.
- While we discussed the attack sequences, the initiation of an attack is not described in much detail. Various attacking techniques, ranging from malware to Trojan hardware can be integrated to this flow to provide a more rigorous analysis of the attack complexity. Even, hybrid and distributed attack sequences can be developed on that basis.
- Our current study focussed on violation of *availability* and *integrity* as long as it violates the task deadline. However, a well-achieved schedule of events may still be dangerous if the computed values are erroneous. This direction of research is already well-studied in the context of fault-tolerant computing. It remains an interesting proposition

on how to combine ASD with attacks, which intend to corrupt the computation.

REFERENCES

[1] C. Salter, O. S. Saydjari, B. Schneier, and J. Wallner. Toward a secure system engineering methodology. In Proceedings of the 1998 workshop on New security paradigms (NSPW '98). pp. 2–10, 1998.

[2] R. Pellizzoni, N. Paryab, M. K. Yoon, S. Bak, S. Mohan and R. B. Bobba. A Generalized Model for Preventing Information Leakage in Hard Real-Time Systems. In Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), pp. 271–282, 2015.

[3] M. K. Yoon, S. Mohan, C. Y. Chen and L. Sha. TaskShuffler: A Schedule Randomization Protocol for Obfuscation against Timing Inference Attacks in Real-Time Systems. In Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), pp. 1–12, 2016.

[4] K. Jiang, P. Eles, Z. Peng, S. Chattopadhyay and L. Batina. SPARTA: A scheduling policy for thwarting differential power analysis attacks. In Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 667–672, 2016.

[5] K. Jiang, L. Batina, P. Eles and Z. Peng. Robustness Analysis of Real-Time Scheduling Against Differential Power Analysis Attacks. In Proceedings of the IEEE Computer Society Annual Symposium on VLSI, pp. 450–455, 2014.

[6] S. Mohan, M. K. Yoon, R. Pellizzoni and R. Bobba. Real-Time Systems Security through Scheduler Constraints. In Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS), pp. 129–140, 2014.

[7] S. Mohan, S. Bak, E. Betti, H. Yun, L. Sha, and M. Caccamo. S3A: Secure system simplex architecture for enhanced security and robustness of cyber-physical systems. In Proceedings of the ACM Conference on High Confidence Networked Systems, 2013.

[8] M.-K. Yoon, S. Mohan, J. Choi, and L. Sha. Memory Heat Map: Anomaly Detection in Real-Time Embedded Systems using Memory Behavior. In Proceedings of the ACM/EDAC/IEEE Design Automation Conference (DAC), 2015.

[9] M. M. Z. Zadeh, M. Salem, N. Kumar, G. Cutulenco, and S. Fischmeister. SiPTA: Signal Processing for Trace-based Anomaly Detection. In Proceedings of the International Conference on Embedded Software (EMSOFT), 2014.

[10] M. Völp, B. Engel, C., J. Hamann, and H. Härtig. On Confidentiality Preserving Real-Time Locking Protocols. In Proceedings of the IEEE Real-Time Embedded Technology and Applications Symposium (RTAS), 2013.

[11] M. Völp, C., J. Hamann, and H. Härtig. Avoiding Timing Channels in Fixed-Priority Schedulers. In Proceedings of the ACM Symposium on Information, Computer and Communication Security, pp. 44–55, 2008.

[12] J.-B. Itier. A380 Integrated Modular Avionics. Available at http://www.artist-embedded.org/docs/Events/2007/IMA/Slides/ARTIST2_IMA_Itier.pdf.

[13] AUTOSAR consortium. Automotive Open System Architecture. Available at https://www.autosar.org.

[14] RTCA. DO-178C (Software Considerations in Airborne Systems and Equipment Certification). Available at http://www.rtca.org/store_product.asp?prodid=803.

[15] J. Alves-Foss. Multiple Independent Levels of Security. In Springer Encyclopaedia of Cryptography and Security, pp. 815–818, 2011.

[16] J. Windsor. Time and Space Partitioning in Spacecraft Avionics. In Proceedings of the IEEE International Conference on Space Mission Challenges for Information Technology, pp.13–20, 2009.

[17] L. Wen, W. Jiang, K. Jiang, X. Zhang, X. Pan and K. Zhou. Detecting Fault Injection Attacks on Embedded Real-Time Applications: A System-Level Perspective. In Proceedings of the IEEE High Performance Computing and Communications (HPCC), 2015.

[18] N. Moro, K. Heydemann, A. Dehbaoui, B. Robisson, and E. Encrenaz. Experimental Evaluation of Two Software Countermeasures Against Fault Attacks. In Proceedings of the IEEE Conference on Hardware-Oriented Security and Trust (HOST), pp. 112–117, 2014.

[19] K. Jiang, P. Eles and Z. Peng. Optimization of Message Encryption for Distributed Embedded Systems with Real-Time Constraints. In Proceedings of the IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems, pp. 243–248, 2011.

[20] K. Jiang, P. Eles and Z. Peng. Co-design Techniques for Distributed Real-Time Embedded Systems with Communication Security Constraints. In Proceedings of the Design Automation and Test in Europe (DATE), pp. 947–952, 2012.

[21] S. Peisert, J. Margulies, D. M. Nicol, H. Khurana and C. Sawall. Designed-in Security for Cyber-Physical Systems, IEEE Security & Privacy, vol. 12, no. , pp. 9–12, Sept.-Oct. 2014.

[22] C. W. Lin, Q. Zhu and A. Sangiovanni-Vincentelli. Security-Aware Modelling and Efficient Mapping for CAN-Based Real-Time Distributed Automotive Systems. In IEEE Embedded Systems Letters, vol. 7, no. 1, pp. 11–14, 2015.

[23] N. Veyrat-Charvillon, M. Medwed, S. Kerckhof and F-X. Standaert. Shuffling against side-channel attacks: A comprehensive study with cautionary note. In International Conference on the Theory and Application of Cryptology and Information Security, pp. 740–757, Springer, 2012.

[24] E. Latronico and P. Koopman. Representing Embedded System Sequence Diagrams as a Formal Language. In Proceedings of the International Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools (UML), pp. 302–316, 2001.

[25] A. Barenghi, G. Bertoni, L. Breveglieri, M. Pellicioli, and G. Pelosi. Low voltage fault attacks to AES and RSA on general purpose processors, Cryptology ePrint Archive, Report 2010/130, 2010.

[26] IBM Blueprints. Using the Linux CPUFreq Subsystem for Energy Management. Available at https://www.ibm.com/support/knowledgecenter/linuxonibm/liaai.cpufreq/liaai-cpufreq_pdf.pdf.

[27] S. Bhattacharya and D. Mukhopadhyay. Curious case of Rowhammer: Flipping Secret Exponent Bits using Timing Analysis, Cryptology ePrint Archive, Report 2016/618.

[28] S. Skorobogatov. Flash memory 'bumping' attacks. In Proceedings of the 12th international conference on Cryptographic hardware and embedded systems (CHES), pp. 158–172, 2010.

[29] N. Miura, Z. Najm, W. He, S. Bhasin, X.-T. Ngo, M. Nagata, and J.-L. Danger. Pll to the Rescue: A Novel EM Fault Countermeasure. In Proceedings of the ACM Design Automation Conference (DAC), 2016.