

Design of Reactive Security Mechanisms in Time-Triggered Embedded Systems

Dawid Trawczynski
AMD

Janusz Zalewski
Florida Gulf Coast University

Janusz Sosnowski
Warsaw University of Technology

ABSTRACT

In the paper we discuss how a single node communication interface failure in a time-triggered system can be used to model a DoS-type attack. More so, we present a design approach based on active detection of common DoS characteristics, which can serve as a template for attack detection. This approach is feasible in time-triggered systems because of the periodic and deterministic characteristics either at the fieldbus communication or application level. We support our discussion with an example case study of a vehicle braking system implementing time-triggered messages disturbed by fault injection.

CITATION: Trawczynski, D., Zalewski, J., and Sosnowski, J., "Design of Reactive Security Mechanisms in Time-Triggered Embedded Systems," *SAE Int. J. Passeng. Cars – Electron. Electr. Syst.* 7(2):2014, doi:10.4271/2014-01-0341.

INTRODUCTION

Due to a rapid increase in connectivity of automotive electronics, issues with security in automotive vehicles have been known for more than a decade [1]. This is paralleled by security concerns in aircraft databuses. Even though no serious accident has been reported yet, addressing related problems is a matter of critical importance to the safety of passengers and the general public. A recent study by the U.S. Transportation Research Board [2], in one of its findings, made the following suggestion to researches:

"Examine security vulnerabilities arising from the increase in remote access to and interconnectivity of electronics systems that can compromise safety-critical vehicle capabilities such as braking, exterior lighting, speed control, and steering."

Research in this area has been progressing steadily, with more and more publications appearing every year [3, 4, 5, 6, 7, 8, 9]. One of the problems with addressing bus security issues in automobiles is that there is a wide diversity of bus technologies used in modern cars. The report [2] lists over a dozen bus related standards, from CAN to SMARTwireX. It is therefore difficult to address in a meaningful way general problems,

across the board. Thus, this paper concentrates on improving security mechanisms in time-triggered systems, such as TTP or FlexRay.

The rest of the paper is structured as follows. The next section outlines the background of the problem, including TTP protocol, fault injection and threat modeling. Section Fault Injection Supported Threat Model and Attack Detection present the basics of the proposed solution, section Braking System Case Study discusses the case study of a braking system, and final section concludes the paper.

BACKGROUND

In this section, we present some background information relevant to the solution. Specifically, we discuss time-triggered embedded and distributed system protocols with focus on Time-Triggered Protocol (TTP), tested in the case-study. We also discuss a testing technique known as fault injection that is used in robustness testing of system error handling abilities; we used this technique to model a security threat in the TTP network. Finally, we present the security threat model and review related research in the domain of time-triggered system's security.

Time-Triggered System Design

The time-triggered computing paradigm has been introduced relatively recently in industrial settings, and brings many potential benefits to the design of safety related features in the transportation domain (avionics, automotive, rail). The main benefit of this paradigm stems from allowing an embedded system designer to establish apriori (pre-run-time), highly deterministic communication schedule that eliminates medium access (i.e., communication channel) conflicts for high priority network messages [10].

Other essential design properties of time-triggered systems are: composability (maintaining of sub-system properties at a system scale allowing consistent testability), use of apriori knowledge (system information and its exchange time intervals are known at design time), naming (no explicit naming required), acknowledgment (correct reception acknowledgment only sent when all group members correctly receive information), and fail-silence in time domain (network messages are either delivered at correct time intervals or not at all).

Multiple standards and protocols support deterministic communication. The main ones used in automotive and avionics applications are:

- TTP [11], [13]
- TTEthernet [12], [13]
- TTCAN [14], [15]
- FlexRay [16], [17].

In this paper, we focus mainly on TTP and present a respective case study discussed in Section 4. This protocol supports a Time-Triggered Architecture (TTA) and its basic research and development has been done at Vienna University of Technology, while the commercial product development is being supported by TTTech, Inc. [13].

The TTP protocol supports a number of logical and physical layers that are either implemented at the host or communication controller. The main layers of the protocol are shown in Fig. 1, and their details are described in [10]. Our security analysis is based on *Application Software in Host* and *Communication Network Interface* (CNI) layers.

Application Software in Host Application Membership
Fault Tolerant Unit Membership Permanence of Messages
Communication Network Interface
Cold and Warm Startup Redundancy Management
Smallest Replaceable Unit Membership Clock Synchronization Implicit Acknowledgment
Media Access: TDMA Bit Synchronization Bit Encoding/Decoding

Figure 1. TTP Protocol Layers [10]

The host layer implements application functionality (or part of it) and maps that functionality to target code (e.g., PowerPC). The CNI layer is essentially a reserved memory area that stores host application messages to be transmitted or received over a network. These messages are processed together with a Message Descriptor List (MEDL) data structure that sets up and monitors correct and timely communication over CNI. More details on CNI can be found in [10]. In our security analysis and design approach, we disturb application messages exchanged between hosts over the CNI layer by means of fault injection. The next subsection describes some essential aspects of this testing methodology.

Fault Injection

Fault injection techniques have been developed by academia and industry for system dependability evaluation identifying fault effects and improving system fault robustness (see [18, 19, 20] and references therein). Taking into account the objective of fault injection we can distinguish two general classes of fault injectors:

FIR - Fault injectors dealing with real systems. Faults are injected by simulating physical disturbances (hardware implemented fault injectors) in the analyzed system or by emulating fault effects at the logical level within the analyzed system resources (accessible from the programming level: software implemented fault injection - SWIFI)

FIM - Fault injectors dealing with system models. In this approach we deal with a model of the analyzed system and simulate faults within the model. Here we can consider various abstract levels of modeling: fine grained (e.g., electrical or RTL models) or course grained (e.g., VHDL, SystemC functional models). In practice, there are also mixed approaches combining FIR and FIM features. This is especially interesting for reactive (or real-time) systems which cooperate with an active environment [20, 21].

The fault injection campaign is composed of fault injection experiments related to specific system resources, fault classes, their triggering time and location. In fault injection experiments we have to specify not only the fault load but also workload in accordance with the analysis goal, e.g., checking fault susceptibility, error detection mechanisms, integrity/security violation, etc. An important issue is to trace fault effect propagation from its source via logical effects (errors) to application level consequences (failures). In this process we can check the effectiveness of various error detectors. They may relate to low level hardware mechanisms (e.g., segmentation faults, access violation, invalid opcode, parity error - signaled as system exceptions, watchdog timer alarms, etc.) or microkernel mechanisms (alarms related to detected deadline misses, abnormal termination of a system call, exceptions).

Dealing with security or DoS problems we should select appropriate fault models directly (e.g., transmitting wrong simulated attacks), or indirectly (simulating internal system

errors which can be attributed to external violations). In this process an important issue is to identify objects susceptible to threats (e.g., access ports), sensitive information within the system and possible paths of accessing or modifying it [22]. We should also take into account the problem of security compromising in consequence of other faults in the system (e.g., transient disturbances) and checking if they do not increase the probability of security threats, data leakage, open access paths to sensitive data, reveal passwords, create side channels for attacks, etc.

Beyond the classical error detection mechanisms available in commercial systems (invalid opcode, access violation, memory parity, etc.) we may include special software assertions checking program control flow, etc. Moreover, these mechanisms can be supported with on-line monitoring of event or performance logs provided by the system or applications [23]. These logs characterize normal operations so many security DoS attacks may disturb the image of logs (operational profile) and facilitate their detection or threat prediction. In the next section, we discuss in more detail relevant security threat models for embedded systems.

Security Threat Models

One important problem in security of embedded systems is the definition of their vulnerabilities, which can affect the controller. In order to define them, one first needs to differentiate an embedded system from a traditional computing system. Among typical characteristic of networked embedded systems, one can find [24]:

- multiple interacting nodes
- incorporation into control systems operating without human intervention
- purpose other than general computing and communications
- natural or engineered contexts.

Fig. 2 shows an example high-level model that takes into account above system features. This model consists of the following main subsystem models [25]:

- Plant (the controlled object)
- Controller (main computing device issuing control commands to the Plant)
- Observer (implicit and not shown, used to measure plant's response via its output variables).

The *Plant* inputs are *Controller* outputs and environment disturbances (non-malicious and malicious). The malicious disturbances can be modeled as security threats and they affect mainly a controller or plant operation. In modern embedded systems, the controller often interfaces with a network (real-time and non-real-time), database, and a user. These interfaces are not shown in Fig. 2 but are implicitly included in the controller.

Security threat models are mostly concerned with network, database and user interfaces because a breach in those interfaces can reduce confidentiality, availability and integrity of exchanged information in an embedded system.

We are mainly concerned with network interface security threats. Specifically, in our modeling approach, after [26], we consider:

- introduction of spurious network messages
- message deletion
- message corruption
- message flooding.

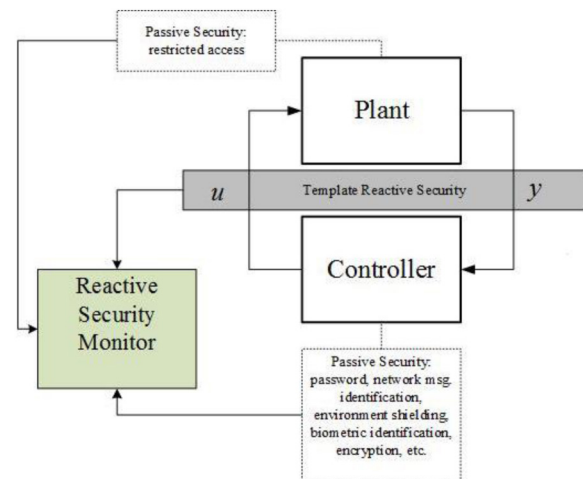


Figure 2. Template Security - System View

In the paper, we are focused on deleted (or missing) message threat model, which is discussed in detail in the Fault Injection Supported Threat Model and Attack Detection section. Other threat models can be also investigated. In the next section, we describe some related work on the security of time-triggered protocol.

Related Work

It's been widely agreed upon that the security aspects of time-triggered systems have not been extensively covered or explicitly included in the protocol specification, although its dependability aspects help in securing the protocol [27]. Given this fact, there are essentially two approaches to address related problems. First, as discussed in [27] and [28], a new architecture may be proposed, which would take care of securing communication and, at the same time, make sure that the safety concerns are addressed. Such an approach addresses security at the design level and has been discussed in publications of the Vienna group [27], [28], [29].

The second approach is to investigate vulnerabilities of the existing time-triggered protocols and address them without developing a new architecture or modifying an existing one, but by proposing respective extensions that adequately react to threats and develop respective responses. This approach