A Study of the Interplay Between Safety and Security Using Model-Implemented Fault Injection

Behrooz Sangchoolie, Peter Folkesson, Jonny Vinter
Department of Electronics, RISE Research Institutes of Sweden
Borås, Sweden
Emails: {behrooz.sangchoolie, peter.folkesson, jonny.vinter}@ri.se

***Abstract*—The combination of high mobility and wireless communication in many safety-critical systems have increased their exposure to malicious security threats. Consequently, many works in the past have proposed solutions to ensure *safety* and *security* of these systems. However, not much attention has been given to the interplay between these two groups of non-functional requirements. This is a concern as safety solutions may negatively impact system security and vice versa. This paper addresses the interplay between safety and security by proposing an attack injection framework, based on model-implemented fault injection, suitable for model-based design. The framework enables us to study and evaluate the impact of cybersecurity attacks on system safety early in the development process. To this end, we have implemented six attack injection models and conducted experiments on Simulink models of a CAN bus and a brake-by-wire controller. The results show that the security attacks modeled could successfully impact the system safety by violating our defined safety requirements.**

***Keywords*—fault injection; attack injection; security; safety; cybersecurity attack; model-based design;**

I. Introduction

Embedded electronics have been extensively used in safety-critical systems. Examples of such systems are the ones used in the automotive and avionic domains where failures in them could have severe consequences such as loss of lives or environmental damages. Safety-critical systems are therefore usually equipped with various mechanisms protecting the systems from different types of failures. Traditionally, the main focus of these mechanisms has been to protect the system from hardware and software faults. However, due to the increasing connectivity of embedded electronic systems in recent years, these systems are becoming more exposed and thus vulnerable to other types of failures caused by cybersecurity attacks. Connected (thereby possibly exposed) safety-critical systems must therefore be equipped with security mechanisms for protection against such attacks.

Security mechanisms have been extensively studied in the computer security and network security domains. However, the use of such mechanisms in embedded electronic systems is often hampered by resource and energy constraints. Moreover, when it comes to safety-critical systems, one also needs to address the interplay between safety and security as security mechanisms are specifically not allowed to prevent the intended functionality of the safety mechanisms. Therefore, future testing approaches could benefit from a holistic view on how to design and assess systems from both safety and security perspectives.

Fault injection is a common testing approach for evaluating system safety. Functional safety standards such as IEC 61508 [1] and ISO 26262 [2] recommend, or highly recommend, the use of fault injection to prove that malfunctions in electrical and/or electronic systems will not lead to violations of safety requirements. Analogous to fault injection, *attack injection* may be used to evaluate the impact of cybersecurity attacks on system security. This is due to the fact that cybersecurity attacks may be considered as a special type of faults which are human made, deliberate and malicious, affecting hardware/software from external system boundaries and occurring during the operational phase [3]. In this paper we propose an attack injection framework based on fault injection to address the interplay between safety and security.

Several attack injection tools and frameworks have been proposed in the past [4], [5], [6], [7], [8], which all target the system either in later parts of the system's development process or even after deployment of the system. However, the attack injection framework proposed in this paper simulates cybersecurity attacks in early development phases of systems; thus it is suitable for model-based design and development of, e.g., automotive embedded systems [9]. Using this framework, we perform attack injection experiments on behavior models of systems with the Matlab/Simulink [10] environment. *To the best of our knowledge, we are the first to use model-implemented fault injection to simulate cybersecurity attacks in early development phases of systems.*

The attack models investigated in this paper activate different threats as specified by the Microsoft STRIDE security model [11]. The threats investigated and their corresponding cybersecurity attack models are shown in Table I. This paper also maps the attacks specified in Table I to some of the commonly used fault models recommended by standards such as ISO 26262 [2]. This mapping suggests that the lessons learned from previous fault injection studies could be adopted when evaluating system security.

In summary, the paper makes the following contributions:

- Extends a model-implemented fault injection tool [12] with *(i)* an attack injection framework capable of injecting cybersecurity attacks in early development phases of systems; *(ii)* a fault/attack injection database for storing fault/attack injection data in a generic, tool-independent

Table I
CYBERSECURITY ATTACKS MODELED IN THE PAPER ALONG WITH THE STRIDE THREATS EXPLOITED.

STRIDE Threat	Cybersecurity attack	Description
Repudiation	**Replay a sequence of data**	Targeting freshness and/or non-repudiation of data by resending old captured data transmitted by another entity.
	Replay a random message	
Tampering	**Corrupt data or code**	Targeting integrity of data/code by making it erroneous permanently.
	Corrupt messages	Targeting integrity of data by making the data erroneous.
Denial of service	**Jamming**	Targeting availability of systems by performing denial of service attacks against communication media such as wireless media by inserting random noise.
Information disclosure	**Intercept**	Targeting confidentiality, privacy or availability of data on the communication network by capturing data so that they do not reach the intended destination.

format facilitating comparison of experiments conducted with different fault/attack injection tools and techniques.
- Implements six cybersecurity attack models in the Matlab/Simulink [10] environment.
- Studies the interplay between safety and security by investigating the capability of each of the modeled cybersecurity attacks to violate defined safety requirements.
- Drives insights on how results obtained from traditional fault injection techniques may be reused, or refined, for attack injection by mapping commonly used fault models to cybersecurity attack models.

II. BACKGROUND

A. Threats to Dependability and Security

The threats to dependability comprise of faults, errors and failures [3], [13]. A system failure occurs when the delivered service deviates from the correct intended service. The failure is caused by an error, which is a deviation from the correct system state, while the cause of the error is called a fault. In this paper, we map these threats to concepts that are commonly used in the security domain, such as attack, intrusion and compromised system (see Fig. 1).

Avizienis et al. [3] define an *attack* as a special type of fault which is human made, deliberate and malicious, affecting hardware or software from external system boundaries and occurring during the operational phase. An *intrusion*, however, refers to when an unauthorized entity gains access to system resources by circumventing the system's security protections [14]. Thus, as shown in Fig. 1, *an attack (fault) may lead to an intrusion (error) which may result in a compromised system (failure of the intended service).*

When discussing threats to dependability and security, one could also encounter concepts such as *threat* and *vulnerability*. The former may be defined as a potential cause of unwanted incidents that may result in harm to a system or organization; while the latter may be defined as a weakness of an asset or control that can be actively exploited by one or more threats in a *threat action* (attack) [14], [15].

B. Related Work

In connected safety-critical systems, safety and security are intertwined. This is due to the fact that security threats to a system could potentially affect the system safety and vice versa. Therefore, this paper focuses on the use of fault injection techniques for simulating cybersecurity attacks to evaluate the impact of these attacks on system safety and to identify security vulnerabilities of target systems.

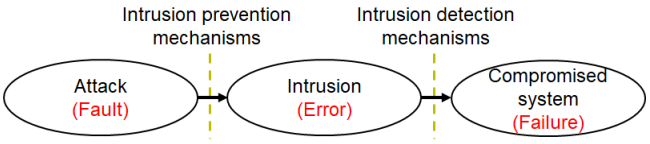


Figure 1. Threats to dependability and security.

Van Woudenberg et al. [16] and Jeong et al. [17] use optical fault injection and power supply disturbances, respectively, on protected smart cards to identify and exploit security vulnerabilities to recover secret keys and access decrypted data. Ujcich et al. [7] use the ATTAIN framework, Fonseca et al. [6] use the VAIT tool and Antunes et al. [18] and Neves et al. [8] use the AJECT tool to perform attack injection to identify vulnerabilities. ATTAIN is developed to inject attacks on software-defined networking (SDN) architectures; VAIT is used to evaluate web security mechanisms; and AJECT is developed to inject attacks on network-related servers, specifically IMAP servers, where it uses a specification of the server's communication protocol and predefined test case generation algorithms to automatically generate attacks.

A recent attack uses the "Row hammer" exploit in DDR3 DRAM [19] which causes bit-flip faults in memory to gain kernel privileges on x86-64 Linux [20]. Aliabadi and Pattabiraman [21] use a compiler-level software fault injection tool as well as an aspect-oriented programming language to simulate security attacks. Authors in [4], [5], [6] inject vulnerabilities, to analyze security mechanisms of web applications. This is done by performing malicious activities that can be used to exploit the given vulnerabilities later on. Therefore, the attacks used are specific to the vulnerabilities injected. However, in this paper, we do not inject any vulnerabilities as we would like to exploit the ones that already exist in the systems under development.

All above mentioned work target the system either in later parts of the system's development process or even after deployment of the system to identify different vulnerabilities. However, our attack injection technique is well-suited for

evaluating security in early development phases before a physical system or prototype is available. To this end, similar to past work [22], [23], [24], [25] where *faults* are injected on behavior models of the system using Matlab/Simulink [10], we use the MODIFI tool [12] to inject *attacks* into Matlab/Simulink models.

C. Cybersecurity Modeling

There are several ways to perform security modeling. Examples of models, methods, and projects that are used for this purpose are CIA (Confidentiality, Integrity, and Availability), STRIDE [11], CVSS [26], HAZOP [27], HEAVENS [28] and EVITA [29]. For this paper, we base our classification of cybersecurity attacks on the well-known STRIDE model [11]. This model is proposed by Microsoft to identify threats, which can be categorized based on the goals and purposes of the attacks. The term STRIDE is an acronym based on the initial letter of the following possible threats:

- *Spoofing* - Attackers pretend to be someone or something else.
- *Tampering* - Attackers change data in transit or in a data store.
- *Repudiation* - Attackers perform actions that cannot be traced back to them.
- *Information disclosure* - Attackers get access to data in transit or in a data store.
- *Denial of service (DoS)* - Attackers interrupt a system's legitimate operation.
- *Elevation of privilege* - Attackers perform actions they are not authorized to perform.

III. Attack Injection Framework

A. Attack Injection

In this paper, we have extended the MODIFI [12] fault injection tool to support attack injection. This is done by adding attack models, simulating the six cybersecurity attacks presented in §III-B, to the fault model library of the tool. MODIFI has previously been used to inject faults for dependability assessment of software developed as Simulink models [22]. It enables both functional and non-functional testing to be performed in early phases of the software development life cycle within the same environment typically used for model-based development.

Fig. 2 shows an overview of MODIFI. The user interface is implemented using Java, and the injection engine is developed in MATLAB `m-code`. For this study, we have developed a MySQL database for storing configuration data and results of injection experiments. The database, called FIND (Fault/attack INjection Database) has been developed for storing both fault- and attack injection data in a generic, tool-independent format to facilitate comparison of experimental results from multiple tools and injection techniques.

B. Attack Modeling

The general structure of the implemented attacks is shown in Fig. 3. Here `In` is the input signal targeted by the attack resulting in the `Out` output signal. The `DOC Text` file describes the

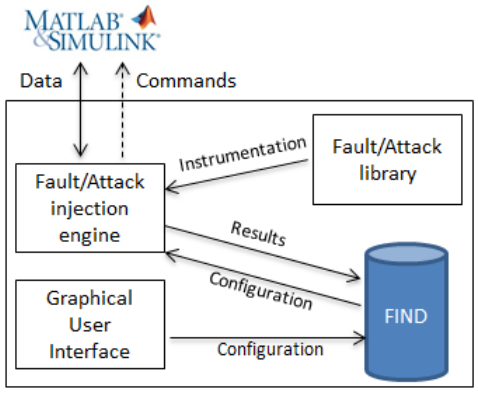


Figure 2. Overview of MODIFI fault and attack injection tool.

datatypes and the names of parameters controlling the attack for each datatype. `ModelId` is the identification number of the attack model used for retrieving the correct parameters for it (if any), while `Clock` is the current timestamp used for controlling the point in time and duration of the attack injection. The point in time for attack injection may be set to random or specific time points. The main function of the attack model can be implemented as either Simulink models or Matlab code (as indicated by the `fcn` block). The remainder of this section presents the cybersecurity attack models implemented in different `fcn` blocks.

1) `Replay(n, m)`: A replay attack collects and buffers communication data on the network to be replayed at a later time [30]. We have modeled the replay attack as a parameterized replay attack (see Fig. 4a) capable of replaying `n` messages ($M_1, M_2, ...M_n$) from consecutive time steps which are `m` time steps old ($n \leq m$), where the parameters `n` and `m` can be statically set or randomly generated within a range of values for each injected attack.

2) `ReplayRandomMessage`: We have implemented a variant of the replay attack described in §III-B1 in the form of a replay random message attack. The attack replays one message that is randomly selected from the ones captured before the injection time (see Fig. 4b).

3) `CorruptDataOrCode(x, y)`: The corrupt data or code attack makes data or code in a system erroneous. On the software side, it may be used as content pollution, which is the modification of personalized or requested content; and on the hardware side it may be used to inject bit-flips in DRAM cells for example by intensive write operations in

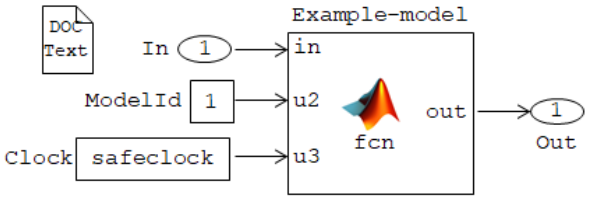
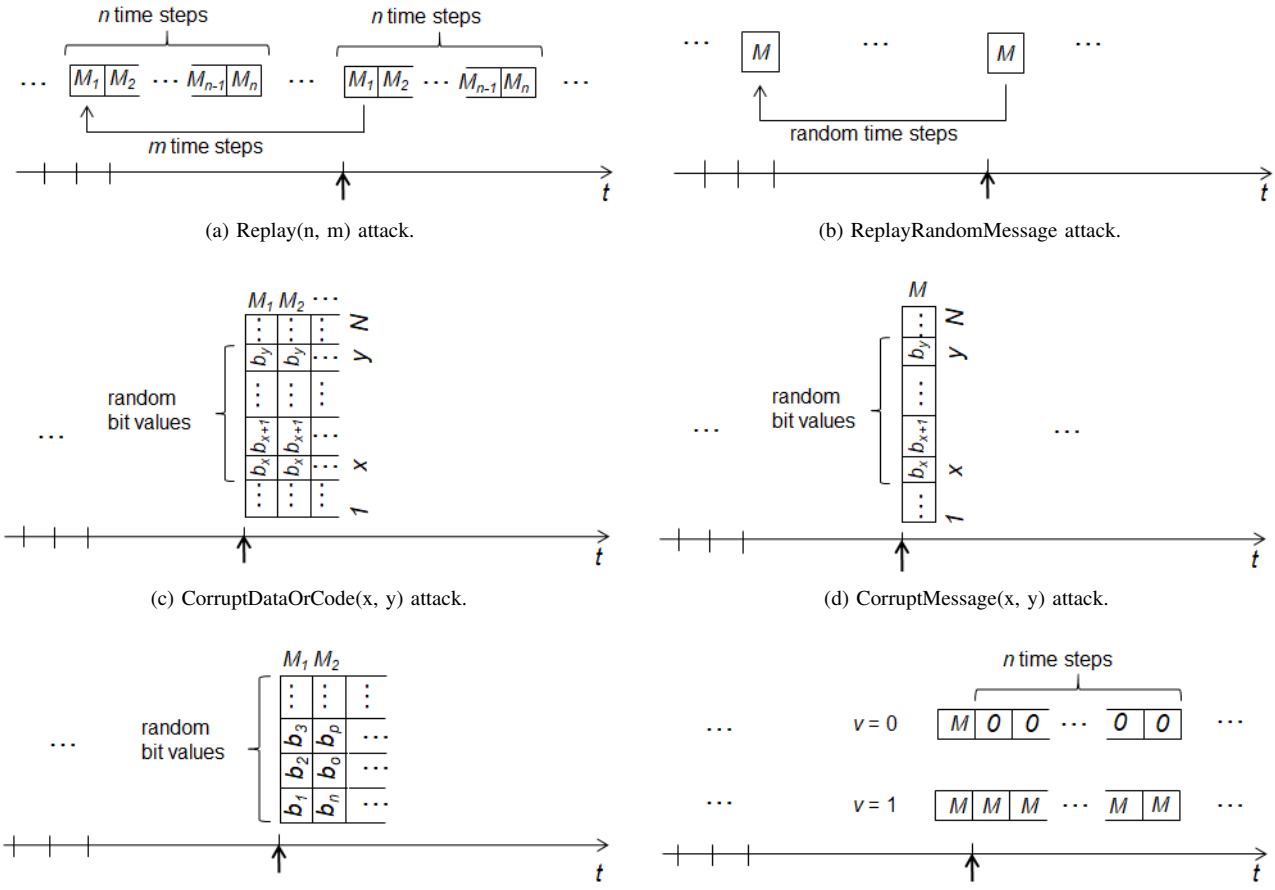


Figure 3. Attack model example.



(a) Replay(n, m) attack.

(b) ReplayRandomMessage attack.

(c) CorruptDataOrCode(x, y) attack.

(d) CorruptMessage(x, y) attack.

(e) Jamming attack.

(f) Intercept(n, v) attack.

Figure 4. Cybersecurity attack models. The initial injection times are specified by upwards arrows (↑).

adjacent memory cells [19]. Our implementation simulates this attack by permanently changing the bits between position `x` and `y` (added as parameters to the attack model) of a bit representation of the attacked message to a random string of zeros and ones (see Fig. 4c). The parameters `x` and `y` can be statically set or randomly generated within a range of values for each injected attack.

4) `CorruptMessage(x, y)`: The corrupt message attack corrupts messages that are sent on the network. Our implementation is similar to the corrupt data or code attack. However, the bits between position `x` and `y` of a bit representation of the attacked value is only corrupted during one time sample, thus only affecting one message (see Fig. 4d).

5) `Jamming`: Jamming attacks are a type of denial of service attacks which may affect communication equipment such as wireless devices [31]. Diverse methods may be used to perform jamming. Here, we have implemented a method simulating *white noise* in which the jammed messages $(M_1, M_2, ...)$ are transformed by changing the bit representation of each transmitted message $(b_1, b_2, ...)$ to a random string of zeros and ones (see Fig. 4e).

6) `Intercept(n, v)`: The intercept attack captures communication data to prevent them from reaching the intended destination. An example of this attack is to trick a legitimate client into making a connection to a compromised entity, e.g., by routing the traffic via a compromised router. We have modelled the intercept attack as a parameterized attack where the intercepted value is replaced by either `null` (parameter `v` = 0) or the value before the attack (parameter `v` = 1) during `n` consecutive time samples (see Fig. 4f). The parameters `n` and `v` can be statically set or randomly generated within a range of values for each injected attack.

IV. Evaluation

A. Target Systems

We evaluate our attack models using two target systems from the automotive domain, namely *CAN bus* and *brake-by-wire (BBW)* controller. The CAN bus model is developed in our research group, while the brake-by-wire controller model is developed by AB Volvo for research purposes.

1) CAN Bus Model: The CAN bus standard is designed to allow microcontrollers and devices to communicate with each other in applications without a host computer. It is used in industrial, automotive and medical fields e.g., to exchange information between different electronic components. CAN bus does not support any security features intrinsically, thus,

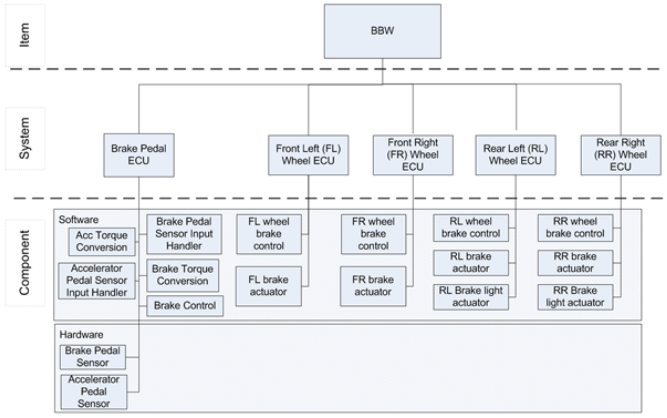


Figure 5. Overview of the brake-by-wire (BBW) system.

it is prone to many security attacks. In most implementations, applications are expected to deploy their own security mechanisms, which is why several authentication solutions have already been proposed, however, such solutions have not yet been widely adopted by e.g., the automotive industry [32].

The CAN bus model used in this paper is created using the components and blocks available in MATLAB R2014b. However, note that the CAN bus could also be modeled using the *vehicle network toolbox* available in MATLAB R2017b [33]. Also note that this paper uses a CAN bus that is not equipped with any integrity checksum such as CRC. This is because (i) here, we aim to study the effectiveness of different attack models and (ii) as there is no built-in message authentication, an attacker can edit a CAN message and recompute the CRC without the substitution being detected.

2) Brake-by-wire (BBW) Model: A four-wheel BBW application, used in several previous work [22], [34] and research projects such as TIMMO [35] and VeTeSS [36], is used in this paper as a control application example for our attack injection experiments. An overview of the BBW system developed using five ECUs (Electronic Control Units) connected to a bus is given in Fig. 5. Each wheel has one corresponding ECU while the central brake controller is located on the brake pedal ECU. The BBW application is distributed across the brake pedal ECU and the four wheel ECUs. The input sensors to the system are the *brake pedal* (for driver input of requested brake torque); *vehicle speed sensor* (for measuring longitudinal speed of the vehicle); and four *wheel speed sensors* (for measuring angular speed of each wheel). There are also four brake actuators used for controlling the brake torque of each wheel.

B. Attack Injection Outcome Classification

We conduct different attack injection *campaigns* on the two target systems presented in §IV-A. An attack injection campaign refers to a set of attack injection experiments using the same attack model on a given *workload*; a workload is a target system running with a given input stimuli. In this paper, we classify the outcome of each experiment according to the

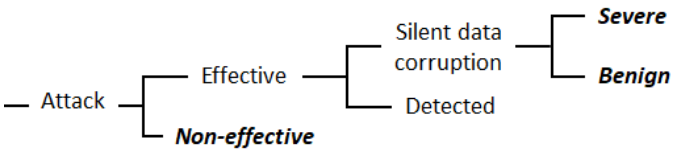


Figure 6. Impact of injected attacks on safety.

impact of the injected attack on the target system's different output signals. Note that one could also analyze and evaluate the attack models with respect to other dependability metrics such as mean time to failure (MTTF) or the extent in which the system is compromised. However, to facilitate the study of the interplay between safety and security, we chose to evaluate the impact of an attack on the output signals since attacks may violate system safety requirements by causing erroneous output. Therefore, the outcome of each experiment is classified into one of the following categories (see Fig. 6):

- *Non-effective.* The system terminates its execution normally without causing either a detectable failure or a *silent data corruption* (*SDC*), also known as undetected erroneous output.
- *Benign.* The injected attack causes an SDC in at least one of the output signals, however the deviation of the output from the nominal value is within an acceptable range, which is defined according to the target system requirements.
- *Severe.* The injected attack causes an SDC with an output deviation from the nominal value that is outside an acceptable range, which is defined according to the target system requirements.

The impact of injected attacks on a system may also be detected or handled by various security/safety mechanisms, resulting in another outcome classification known as *Detected*. However, as the current versions of our target systems are not equipped with any security/safety mechanisms, none of the conducted experiments resulted in this category. We choose the unprotected version of the target systems as we are interested in studying the effectiveness of the attack models as well as to find attack models and attack locations that are more successful in violating safety requirements. However, in the future, we plan to use the results of this study to equip our target systems with cost-effective safety/security mechanisms.

C. Experimental Results

In this section, we report the results obtained from the attack injection experiments. The results *(i)* demonstrate the effectiveness of attack models in causing erroneous outputs (see §IV-C1); and *(ii)* reveal useful information about the different attack models and the extent in which they violate safety requirements (see §IV-C2).

1) Effectiveness of the Attack Models in Causing Erroneous Outputs: The effectiveness of the attack models are evaluated by targeting the communication link of a CAN bus model. This link consists of four signals, namely `Can ID`, `DLC`,

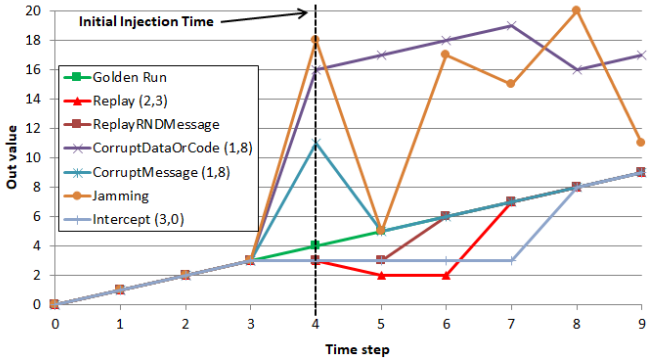


Figure 7. Results for attacks injected on a CAN bus model.

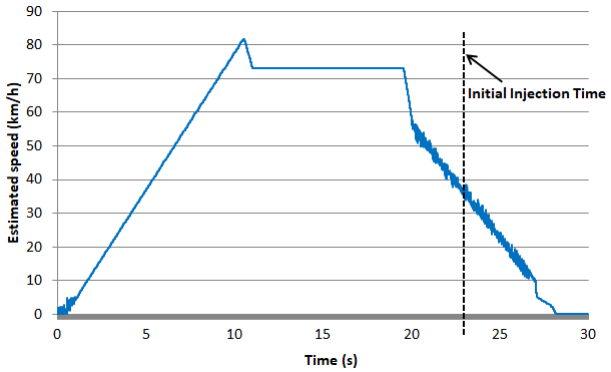


Figure 8. Nominal estimated speed as a function of time for the BBW model.

`Payload`, and `Toggle`. Here, the input payload is from 0 to 9. Therefore, the expected output of the model (`Out`) should also be values from 0 to 9. We have conducted experiments using all six attack models and confirmed their effectiveness as they all resulted in erroneous outputs in one or more time steps (see Fig. 7). The initial injection time of all attacks is time step 4. A short description of each attack is as follows:

- The `Replay(2, 3)` attack replays two messages at time steps four and five. The replayed messages are taken starting from three time steps earlier, i.e., the messages that were sent at time steps one and two.
- The `ReplayRandomMessage` attack replays a message randomly from the messages sent before time step 4. Here the message selected is from time step 3.
- The `CorruptDataOrCode(1, 8)` corrupts bits of the 8-bit `Payload` signal as this signal is of higher importance to the attacker. The bits are randomly selected and are affected at all time steps starting from time step 4. Therefore, it simulates the case where certain data or code is targeted by an attacker.
- The `CorruptMessage(1, 8)` attack is similar to the `CorruptDataOrCode(1, 8)`, however, bits of the `Payload` signal are only affected at time step 4.
- The `Jamming` attack implemented here simulates white noise on all CAN bus signals. Note that the value of `out` at time step 5 is the same as the one obtained for the golden run. This is due to the fact that in this time step, the generated white noise only impacted `Can ID`, which does not impact the value of the `out` signal.
- The `Intercept(3, 0)` attack captures the data sent within three time steps (4, 5, and 6) so that it does not reach the receiver. Here, the CAN receiver continues to use the last value received before the start of the attack.

2) Evaluation of Attacks w.r.t. the Violation of Safety Requirements: In this section, we evaluate and compare the attack models with respect to their violation of safety requirements by targeting the brake-by-wire (BBW) model. Fig. 8 shows the estimated vehicle speed signal of the model during a 30-second time interval in the attack-free scenario. The figure shows that the vehicle is accelerated to a speed of around 75 km/h and at approximately 19.5 seconds of simulated time the braking of the vehicle starts. Two different time steps, selected randomly within an interval of 10 time steps, were chosen for attack injection. These time steps correspond to 5 control loops iterations in the model, or a simulated time between 23.0 and 23.05 seconds marked by the vertical line. Here we inject attacks on all 347 available locations in the BBW model. As we use two randomly selected parameter settings for each attack, a total number of 1388 attacks are injected for each of the attack models. Table II shows the results of the campaigns conducted, which were configured as follows:

- The `Replay(n, m)` attack replays `n` consecutive values `m` time steps old, where both `n` and `m` are randomly selected between 1 and 10 for each attack.
- The `ReplayRandomMessage` attack replays a message that is randomly selected from the previous messages sent on the link.
- The `CorruptDataOrCode(x, y)` attack selects values for `x` and `y` randomly between the start and the end bits of each targeted location and corrupt them at all time steps starting from the initial injection time.
- The `CorruptMessage(x, y)` attack selects values for `x` and `y` randomly between the start and the end bit of each targeted location and target them only once.
- The `Jamming` attack injects random bit values on all bits of the selected location for all time steps starting from the initial injection time.
- The `Intercept(n, 0)` attack intercepts `n` consecutive values, where `n` is randomly selected between 1 to 10 for each attack, replacing the intercepted value with 0 to model capturing data so that it does not reach the intended destination.

There are eleven output signals observed for the BBW model corresponding to the brake torque, rotation of each wheel, estimated vehicle speed and driver requested brake torque. In Table II, the result of an attack is classified as *Non-effective* when no differences are observed between the output signals of the attacked system and the output signals of the attack-free system. Table II shows that a great number of attacks injected into the BBW model are non-effective.

Table II
ATTACK INJECTION RESULTS FOR THE BBW MODEL.

Attack model	Number of attacks	Non-effective	Benign	Severe
Replay(n, m)	1388	712	676	0
ReplayRandomMessage	1388	586	768	34
CorruptDataOrCode(x, y)	1388	457	629	302
CorruptMessage(x, y)	1388	784	537	67
Jamming	1388	97	294	997
Intercept(n, 0)	1388	540	797	51

This could be as a result of inherent robustness of this model to attacks, especially since the value of many signals are overwritten in each control loop iteration. The *effective* attacks are however further classified according to how severely they affect the system behaviour.

Similar to our previous work [22], the estimated vehicle speed output signal is used for classifying the severity of an attack. An attack is classified as *Severe* when the estimated speed of the vehicle has a difference of more than $\pm$10 km/h from the attack-free value. However, if the deviation is within the acceptable range (less than $\pm$10 km/h), the experiment is classified as *Benign*. Here the choice of $\pm$10 km/h is only to investigate an example of a violation of a safety requirement, and in reality, this number should be selected from the safety requirements of the system under test. Note that the deviation in the estimated vehicle speed could be temporarily, however, its consequence may nevertheless be critical depending on e.g., the operational environment (ice, obstacles and etc.).

Table II shows that the `Jamming` attack causes the highest number of safety violations (997 Severe failures), followed by the `CorruptDataOrCode(x, y)` attack (302). The high effectiveness of these attacks in violating safety requirements could be due to two reasons. First, both of these attacks corrupt multiple bits resulting in values that are significantly different than the attack-free values. Secondly, due to the nature of these attacks, the exposure time of the BBW system to these attacks are considerably higher than the other attacks.

In order to study the impact of exposure time in violating safety requirements, we have conducted two more campaigns. The first campaign is conducted for the `Replay(n, m)` attack which instead of using random values between 1-10 for its parameters, uses random values selected between 1-1000. The second campaign selects the same value range (1-1000) for the `n` parameter of the `Intercept(n, 0)` attack. The results are presented in Table III. Here we can see that the number of Severe experiments for the `Replay(n, m)` and `Intercept(n, 0)` attacks are 433 and 847, respectively, which are considerably higher than the results presented in Table II for these attack models. Note that the impact of exposure time on the results may be application-dependent. However, looking at the results presented in Table II and Table III, we confirm the effectiveness of our attack models and framework in violating our safety requirement.

Table III
RESULTS FOR THE BBW MODEL WHEN EXPOSED TO THE `Replay(n, m)` AND `Intercept(n, 0)` ATTACKS FOR A LONG PERIOD OF TIME.

Attack model	Number of attacks	Non-effective	Benign	Severe
Replay$_{long-exposure}$ (n, m)	1388	398	557	433
Intercept$_{long-exposure}$ (n, 0)	1388	249	292	847

D. Mapping of Cybersecurity Attack Models to Commonly Used Fault Models

Some of the cybersecurity attack models implemented in this study could be mapped to fault models commonly used in the safety domain. Table IV shows the mapping between these models. According to the table, for example, the way we model the `CorruptDataOrCode(x, y)` attack in the security domain is similar to the way *multiple stuck-at* faults are modeled in the safety domain. This means that some attacks affect the system in the same way as faults do, which implies that the lessons learned from past fault injection studies could be reused when studying cybersecurity of safety-critical systems. In fact safety mechanisms implemented in the past may already protect systems against certain types of cybersecurity attacks. Therefore, we suggest that safety and security analysis of computer systems should be done by a mixed group of safety and security experts. This could result in the implementation of light-weight and cost-effective mechanisms to achieve both safety and security combined.

V. SUMMARY AND IMPLICATIONS

In this paper, our goal was to study the interplay between safety and security. To this end, we proposed and evaluated an attack injection framework that is capable of modeling different cybersecurity attacks and identifying the ones that violate safety requirements early in the development process. This is important as embedded electronic systems in addition to being used in safety-critical domains, are becoming increasingly connected, hence more exposed to cybersecurity attacks.

The results of the evaluation indicate that traditional fault injection could be effectively used to simulate cybersecurity attacks in early development phases. In fact, all attack models implemented in this study could effectively result in erroneous

Table IV
MAPPING THE CYBERSECURITY ATTACK MODELS IMPLEMENTED IN THIS PAPER WITH COMMONLY USED FAULT MODELS.

Cybersecurity Attack Models	Fault Models
Replay(n, m)	-
ReplayRandomMessage	-
CorruptDataOrCode(x, y)	Multiple stuck-at
CorruptMessage(x, y)	Multiple bit-flips
Jamming	Oscillations
Intercept(n, v)	Stuck-at-zero or Stuck-at-value

outputs or violate our defined safety requirement. However, the attacks that resulted in a higher exposure time were more effective in violating safety requirements.

The attack injection framework could also be used to compare different system designs as well as to evaluate different safety and security mechanisms. The latter is of high importance as safety and security mechanisms could have negative impacts on system security and safety, respectively. For example, *majority voting* is a safety mechanism suggested by the IEC 61508 standard [1] that uses a hardware voter for error recovery. However, this mechanism could negatively impact system security since the voter may increase the opportunities for possible cybersecurity attacks. Therefore, as part of the future work, we plan to use our framework and further study the interplay between safety and security by implementing some of the safety and security mechanisms proposed by standards such as ISO 26262 [2], IEC 62443 [37], and ISO/SAE AWI 21434 [38]. Moreover, in this paper, we only focused on the injection of one attack in each experiment; however, an attacker might perform a sequence of attacks to impact the system. Therefore, as part of the future work, we also plan to enhance our framework by combining different attacks to simulate attack sequences performed by an attacker.

ACKNOWLEDGMENT

We would like to thank AB Volvo for providing us with the prototype brake-by-wire controller. This research was partially supported by the Swedish VINNOVA FFI project "HoliSec: Holistic Approach to Improve Data Security".

REFERENCES

[1] "IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems," Int. Electrotechnical Commission, Standard, 2010.

[2] "ISO 26262:2011 Road vehicles-functional safety," ISO, Standard, 2011.

[3] A. Avizienis, J.-C. Laprie, B. Randell *et al.*, *Fundamental Concepts of Dependability*. University of Newcastle, Computing Science, 2001.

[4] J. Fonseca, M. Vieira, and H. Madeira, "Vulnerability & attack injection for web applications," in *2009 IEEE/IFIP Int. Conf. on Dependable Systems Networks*, 2009, pp. 93–102.

[5] R. V. Bhor and H. K. Khanuja, "Analysis of web application security mechanism and attack detection using vulnerability injection technique," in *2016 Int. Conf. on Computing Communication Control and Automation (ICCUBEA)*, Aug 2016, pp. 1–6.

[6] J. Fonseca, M. Vieira, and H. Madeira, "Evaluation of web security mechanisms using vulnerability & attack injection," *IEEE Trans. on Dependable and Secure Computing*, vol. 11, no. 5, pp. 440–453, 2014.

[7] B. E. Ujcich, U. Thakore, and W. H. Sanders, "ATTAIN: An attack injection framework for software-defined networking," in *2017 47th Annual IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN)*, 2017, pp. 567–578.

[8] N. Neves, J. Antunes, M. Correia, P. Verissimo, and R. Neves, "Using attack injection to discover new vulnerabilities," in *Int. Conf. on Dependable Systems and Networks.* IEEE, 2006, pp. 457–466.

[9] M. Törngren, D. Chen, D. Malvius, and J. Axelsson, *Automotive Embedded Systems Handbook, chapter Model Based Development of Automotive Embedded Systems*. Taylor and Francis CRC press, 2008.

[10] "Matlab/Simulink," https://mathworks.com/products/simulink.html, accessed: 2018-03-21.

[11] F. Swiderski and W. Snyder, *Threat Modeling*. Redmond, WA, USA: Microsoft Press, 2004.

[12] R. Svenningsson, J. Vinter, H. Eriksson, and M. Törngren, "Modifi: A model-implemented fault injection tool," in *Proc. of the 29th Int. Conf. on Computer Safety, Reliability, and Security*, ser. SAFECOMP'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 210–222.

[13] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Trans. on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, 2004.

[14] R. Shirey, "Internet security glossary," Internet Requests for Comments, RFC Editor, RFC 2828, 2000.

[15] "ISO 27000: Information technology – security techniques – information security management systems – overview and vocabulary, third edition," Int. Standard ISO/IEC, Standard, 2016.

[16] J. G. J. van Woudenberg, M. F. Witteman, and F. Menarini, "Practical optical fault injection on secure microcontrollers," in *2011 Workshop on Fault Diagnosis and Tolerance in Cryptography*, 2011, pp. 91–99.

[17] K. Jeong, Y. Lee, J. Sung, and S. Hong, "Security analysis of hmac/nmac by using fault injection," *Journal of Applied Mathematics*, 2013.

[18] J. Antunes, N. Neves, M. Correia, P. Verissimo, and R. Neves, "Vulnerability discovery with attack injection," *IEEE Trans. on Software Engineering*, vol. 36, no. 3, pp. 357–370, 2010.

[19] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them: An experimental study of dram disturbance errors," in *2014 ACM/IEEE 41st Int. Symp. on Computer Architecture (ISCA)*, 2014, pp. 361–372.

[20] "Row hammer privilege escalation vulnerability," https://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-20150309-rowhammer/, accessed: 2018-03-21.

[21] M. R. Aliabadi and K. Pattabiraman, "FIDL: A Fault Injection Description Language for Compiler-Based SFI Tools," in *Int. Conf. on Computer Safety, Reliability, and Security*. Springer, 2016, pp. 12–23.

[22] P. Folkesson, F. Ayatolahi, B. Sangchoolie, J. Vinter, M. Islam, and J. Karlsson, "Back-to-back fault injection testing in model-based development," in *Computer Safety, Reliability, and Security*, 2015.

[23] J. Nilsson, N. Strand, P. Falcone, and J. Vinter, "Driver performance in the presence of adaptive cruise control related failures: Implications for safety analysis and fault tolerance," in *43rd Annual IEEE/IFIP Conf. on Dependable Systems and Networks Workshop (DSN-W)*, 2013, pp. 1–10.

[24] M. Saraoğlu, A. Morozov, M. T. Söylemez, and K. Janschek, "ErrorSim: A tool for error propagation analysis of simulink models," in *Computer Safety, Reliability, and Security*, 2017, pp. 245–254.

[25] G. Juez, E. Amparan, R. Lattarulo, J. P. Rastelli, A. Ruiz, and H. Espinoza, "Safety assessment of automated vehicle functions by simulation-based fault injection," in *2017 IEEE Int. Conf. on Vehicular Electronics and Safety (ICVES)*, 2017, pp. 214–219.

[26] "Common vulnerability scoring system," https://www.first.org/cvss/, accessed: 2018-03-21.

[27] T. Srivatanakul, J. A. Clark, and F. Polack, *Effective Security Requirements Analysis: HAZOP and Use Cases*, Berlin, 2004, pp. 416–427.

[28] "HEAling Vulnerabilities to ENhance Software Security and Safety (HEAVENS)," https://www.sp.se/en/index/research/dependable_systems/heavens/sidor/default.aspx, accessed: 2018-03-21.

[29] "E-Safety Vehicle Intrusion Protected Applications (EVITA)," https://www.evita-project.org/, accessed: 2018-03-21.

[30] P. Syverson, "A taxonomy of replay attacks [cryptographic protocols]," in *Proc. The Computer Security Foundations Workshop VII*, 1994.

[31] K. Pelechrinis, M. Iliofotou, and S. V. Krishnamurthy, "Denial of service attacks in wireless networks: The case of jammers," *IEEE Communications Surveys Tutorials*, vol. 13, no. 2, pp. 245–257, 2011.

[32] N. Nowdehi, A. Lautenbach, and T. Olovsson, "In-vehicle CAN message authentication: An evaluation based on industrial criteria," in *Proc. of the IEEE 86th Vehicular Technology Conf. (VTC)*, 2017.

[33] "Build CAN communication simulink models," https://se.mathworks.com/help/vnt/ug/build-can-communication-simulink-models.html, accessed: 2018-03-21.

[34] B. Sangchoolie, F. Ayatolahi, R. Johansson, and J. Karlsson, "A comparison of inject-on-read and inject-on-write in ISA-level fault injection," in *11th European Dependable Computing Conf.*, 2015, pp. 178–189.

[35] "TIMMO - Timing Model project," http://adt.cs.upb.de/timmo-2-use/timmo/index.htm, accessed: 2018-03-21.

[36] "VeTeSS - Verification and Testing to Support Functional Safety Standards project," https://artemis-ia.eu/project/43-vetess.html, accessed: 2018-03-21.

[37] "IEC/TR 62443-3-1: Industrial communication networks- network and system security- Part 3-1: Security technologies for industrial automation and control systems," Int. Electrotechnical Commission, Standard, 2009.

[38] "ISO/SAE AWI 21434: Road Vehicles-Cybersecurity engineering," ISO/SAE, Standard, Under development.