On the Evaluation of Three Pre-Injection Analysis Techniques for Model-Implemented Fault- and Attack Injection

Peter Folkesson, Behrooz Sangchoolie, and Pierre Kleberger
Dependable Transport Systems
RISE Research Institutes of Sweden
Borås, Sweden
{peter.folkesson, behrooz.sangchoolie, pierre.kleberger}@ri.se

Nasser Nowdehi
Volvo AB
Gothenburg, Sweden
nasser.nowdehi@volvo.com

***Abstract*—Fault- and attack injection are techniques used to measure dependability attributes of computer systems. An important property of such injectors is their efficiency that deals with the time and effort needed to explore the target system's fault- or attack space. As this space is generally very large, techniques such as pre-injection analyses are used to effectively explore the space. In this paper, we study two such techniques that have been proposed in the past, namely *inject-on-read* and *inject-on-write*. Moreover, we propose a new technique called *error space pruning of signals* and evaluate its efficiency in reducing the space needed to be explored by fault and attack injection experiments. We implemented and integrated these techniques into MODIFI, a model-implemented fault and attack injector, which has been effectively used in the past to evaluate Simulink models in the presence of faults and attacks. To the best of our knowledge, we are the first to integrate these pre-injection analysis techniques into an injector that injects faults and attacks into Simulink models.**

The results of our evaluation on 11 vehicular Simulink models show that the *error space pruning of signals* reduce the attack space by about 30–43%, hence allowing the attack space to be exploited by fewer number of attack injection experiments. Using MODIFI, we then performed attack injection experiments on two of these vehicular Simulink models, *a comfort control model* and *a brake-by-wire model*, while elaborating on the results obtained.

***Index Terms*—fault injection, attack injection, cybersecurity testing, pre-injection analysis.**

I. Introduction

Modern connected vehicles increasingly require communication with the surrounding infrastructure and cloud-based resources. This results in an increasing need for resilience towards cybersecurity attacks to guarantee safety and privacy, among other attributes. Resilience is typically achieved by incorporating detection and handling mechanisms for different types of errors, including intrusions caused by attacks [1]. This is commonly accomplished through the introduction of layers of mechanisms detecting and handling both errors and intrusions caused by cyberattacks. In order to test these detection and handling mechanisms, experimental verification and validation (V&V) methods such as fault- and attack injection can be employed.

Fault- and attack injection are techniques used for measuring the dependability attributes of computer systems in the presence of faults and attacks. These techniques could be performed in real-world or in simulation-based test environments. Using these techniques, faults and attacks can also be injected into systems at different stages of the product development life cycle, e.g., as early as the design stage or as late as when the final product is in use.

An important property of fault- and attack injectors is their *efficiency*, i.e., the time and effort needed to explore the target system's fault- or attack space[1]. The error space is generally very large, which is why in the past, several techniques such as *inject-on-read* and *inject-on-write* [2, 3, 4, 5], *code-slicing* [6], *fault list collapsing* [7, 8], and *error space pruning* [9, 10, 11, 12] have been proposed and used to reduce the error space while preserving the accuracy of the measured dependability metrics. Some of these techniques make use of an analysis called *pre-injection analysis* which requires detailed knowledge of the target system, prior to performing any injection experiments, for reducing the error space; while other techniques rely on *post-injection analysis* which also incorporate the knowledge gained from a set of injection experiments to reduce the error space for future experimentation.

The first goal of this paper is to implement and evaluate two of the above mentioned pre-injection analysis techniques (inject-on-read and inject-on-write) in MODIFI [13], a model-implemented fault- and attack injector that has been used in the past [14, 15, 16] for dependability assessment of software developed as Simulink models. Other tools capable of injecting faults into Simulink models include Sabotage [17], SIMULTATE [18] and FIBlock [19]. Since the inject-on-read and inject-on-write techniques require detailed knowledge of the target system for efficient implementation (i.e., they are suitable for white-box testing), they are particularly suitable for implementation in MODIFI. *To the best of our knowledge,*

[1]In this paper, and to improve the readability, we refer to the target system's fault- or attack space as the *error space*.

we are the first to integrate the inject-on-read and inject-on-write pre-injection analysis techniques into an injector that injects faults and attacks into Simulink models.

As discussed in prior work [5, 20], each of the inject-on-read and inject-on-write techniques are capable of modelling only part of the entire error space. Moreover, to be able to evaluate the entire space, two sets of experiments would need to be performed, one for each technique. This in turn incurs an evaluation time that is similar to when no pre-injection analysis is used. This is why the second goal of this paper is to design, implement, and evaluate a new pre-injection analysis technique that allows us to exploit the entire error space more efficiently. We call this technique *error space pruning of signals* as it may be considered an adaption of the error space pruning technique targeting signals in models. Similar to the inject-on-read and inject-on-write techniques, this technique is particularly suitable for implementation in MODIFI as it requires the detailed knowledge of the target system structure provided by MODIFI. The results of the experiments conducted using this technique show that the entire error space could be exploited by about 30-43% less effort, i.e., fewer number of injection experiments.

The remainder of the paper is organised as follows. Section II describes injection-based V&V methods for experimental verification and validation. Section III describes the experimental setup, followed by our results in Section IV. Section V concludes our paper and presents future work.

II. Background and Related Work

A. Injection-Based Verification and Validation (V&V)

Injection-based V&V methods focus on introducing certain characteristics in a system or triggering certain events, to confirm that the system behaves suitably under the corresponding conditions. There are two main types of injection, namely *fault injection* and *attack injection*.

Using fault injection, artificial faults are inserted in a computer system in order to assess its behaviour in the presence of faults. The technique allows the characterisation of specific dependability measures and/or fault tolerant mechanisms available in the system. According to Avizienis, Laprie, and Randell [21], a fault is the adjudged or hypothesised cause of an error, while an error is the part of the total state of the system that may lead to its subsequent service failure. In other words, the faults injected may lead to errors that may propagate in the system and cause system failures.

Avizienis, Laprie, and Randell [21] define an attack as a special type of fault that is human-made, deliberate and malicious, affecting hardware or software from external system boundaries and occurring during the operational phase. Therefore, attack injection in a system is analogous to fault injection where an attack could result in an intrusion; if the intrusion is not handled properly, it could result in a compromised system. While fault injection can be used to evaluate system safety, attack injection can be used to evaluate a system from the security perspective, as well as to study the interplay between safety and security requirements [14].

B. Previous Work on Pre- and Post Injection Analyses

Techniques that rely on fault- and attack injection, in general, come with significant evaluation time and cost related issues, such as resource costs needed to explore the error space. Therefore, techniques such as pre- and post injection analyses, some of which are listed below, may be used to reduce the space, contributing to a more cost-effective verification and validation of systems.

1) Inject-on-read [2, 3, 4]: Using the inject-on-read technique, faults and attacks are injected in a resource, such as a CPU register, immediately before the resource is read. This way, faults and attacks that have identical activation are grouped into the same equivalence class, thereby reducing the error space when evaluating systems' dependability.

2) Inject-on-write [4, 5]: In the inject-on-write technique, faults and attacks are injected in a resource immediately after its content is updated. This way, faults and attacks injected into the resource any time after it has been updated and before it is first read, have identical activation and may be grouped into the same equivalence class, thereby reducing the error space.

3) Code-slicing [6]: Code-slicing may be performed at the source code level to determine the statements that a certain criterion, e.g., output signal, depends on. The intersection between these statements and the statements executed at a certain fault/attack injection time are the statements to target when the aim is to affect the criterion.

4) Post-injection Analyses [22, 23, 24, 25]: Post-injection analyses may feed the results obtained from injection experiments into pre-injection analyses that determine what experiments to perform next. For example, post-injection analyses may be used to determine workload input sequences with the lowest error coverage that can be used for fault injection to estimate the real coverage [24].

5) Fault List Collapsing [7, 8]: Faults and attacks that are determined to be equivalent, may be collapsed into the same equivalence class. Apart from the inject-on-read and inject-on-write techniques described above, other techniques may be used for determining equivalence. For example, Smith, Johnson, and Profeta [8] constructed fault trees for each node in a data flow diagram derived from the target system assembler code; faults injected at leaf nodes ending up at the same node along a path are considered equal and the corresponding sub-tree may be collapsed.

6) Error Space Pruning [9, 10, 11, 12]: In error space pruning, pre-injection analyses are performed to prune faults that have a known outcome or are equivalent to other faults, where the equivalence may be determined using either static (pre-injection) or dynamic (post-injection) analyses [9]. For example, locations that would not reveal any system weaknesses when targeted by faults (e.g., those that do not result in an erroneous output) may be pruned using these techniques.

III. Experimental Setup

In this section, we first present MODIFI, the fault- and attack injector used and extended in this study. We then explain the three pre-injection analysis techniques that were added

to MODIFI and evaluated in this paper. Finally, we provide details about the *comfort control model* and the *brake-by-wire model*, which are two of the target systems used for the evaluations. Note that in addition to these two models, we have evaluated 9 other models taken from MathWorks Simulink Automotive Applications (see Section IV-A). These models are not further elaborated in this section as their details are available online by MathWorks [26]. Also note that although all the models under test in this paper are taken from the automotive domain, the pre-injection analysis techniques proposed and evaluated in the paper could be used to evaluate Simulink models from other domains such as the avionics domain.

A. Model-Implemented Fault/Attack Injection Tool (MODIFI)

MODIFI is a model-implemented fault- and attack injector that has been used in over a decade [14, 15, 16, 13] for dependability assessment of software developed as Simulink models in various domains including the automotive domain. The tool enables 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.

MODIFI injects faults and attacks by adding separate blocks modelling the faults and attacks between the connected blocks of the model. The tool supports single or multiple faults and attacks as well as a wide range of fault models, such as bit-flip and stuck-at faults, and attack models, such as replay and jamming attacks [14]. The tool can also easily be extended with additional user-defined fault- and attack model libraries. In this paper, the attack model used for the experiments is the `InvertMessage` attack, which is a variant of the corrupt message attack where all bits in a sent message are inverted. Multiple attacks or sequences of attacks are not considered. Thus, in our experiments, an inverted bit representation of the attacked value is injected for one time sample. Hence, only one message in one signal is affected during the time sample when the attack is injected for each experiment. However, the attacked signal value may propagate through several of the target system's signals to the output signals after the injection. The tool monitors all output signals of the target system until the end of the simulation to determine the outcome of the experiment.

The injection experiments are configured and controlled via a graphical user interface in MODIFI. Each pre-injection analysis technique evaluated in this study is implemented as a function in the MODIFI configuration module. Each function analyses the target system model structure in order to reduce the locations where faults and attacks may be injected. Thus, when a pre-injection analysis technique is applied, the resulting reduced error space is presented to the user as soon as the analysis finishes (typically in the order of seconds). Any number of faults and attacks may then be configured to be injected into the reduced error space.

B. Pre-Injection Analysis Techniques Under Evaluation

In this paper, we investigate three pre-injection analysis techniques, namely, *inject-on-read*, *inject-on-write* and *error space pruning of signals* which is an adaption of static error space pruning that targets signals in models. The code-slicing, post-injection analyses and fault list collapsing techniques mentioned in Section II-B were not considered as suitable or easily adaptable for model-implemented fault- and attack injection, or they rely on previous fault- or attack injection results which are not part of this study. In this section, we present an overview of the three chosen techniques implemented in MODIFI and discuss high-level benefits of using them. We leave the comprehensive efficiency evaluation of these techniques to Section IV, which is where they are applied to the systems under evaluation. Note that, the chosen pre-injection analysis techniques focus on reducing the number of signals that are candidates for the placement of fault/attack injection blocks, thereby reducing the error space. Therefore, as long as the Simulink target system model contains signals between blocks, the pre-injection techniques could be applied for any fault- or attack model targeting signals, such as bit-flips, stuck-at faults, replay or jamming attacks, at least when single faults or attacks are concerned.

Figure 1 illustrates the use of the three techniques, with a complete error space of an example Simulink model (top left) reduced by inject-on-read (top right), inject-on-write (bottom left) and error space pruning of signals (bottom right). In this figure, the locations that are candidates for fault and attack injections are marked with "✘". The figure shows that by integrating the inject-on-read analysis into MODIFI, faults and attacks are injected only into input signals of the blocks used in the Simulink model, while the integration of the inject-on-write analysis into MODIFI results in the injection of faults and attacks only into output signals of the blocks used in the Simulink model.

As shown in Figure 1, integration of the inject-on-read and inject-on-write techniques into this simple Simulink model results in 39% and 61% reduction of the error space, respectively. However, as discussed in prior work [5, 20], these techniques are suitable in targeting different parts of the entire error space. More specifically, the former techniques model faults and attacks in resources, such as CPU registers, immediately before the resource is read, while the latter techniques model faults and attacks in a resource immediately after its content is updated. In the case of injections performed using MODIFI, this translates into the selection of only input or output signals, respectively, for when the inject-on-read and inject-on-write techniques are used. This means that in order to be able to evaluate the target Simulink models according to the faults and attacks modelled by these techniques, two sets of injection campaigns should be conducted, one using inject-on-read and the other one using inject-on-write. This in turn adds to the size of the error space. In the case of the example shown in Figure 1, then 14 and 9 locations should be targeted by faults and attacks for inject-on-read and inject-on-write, respectively,

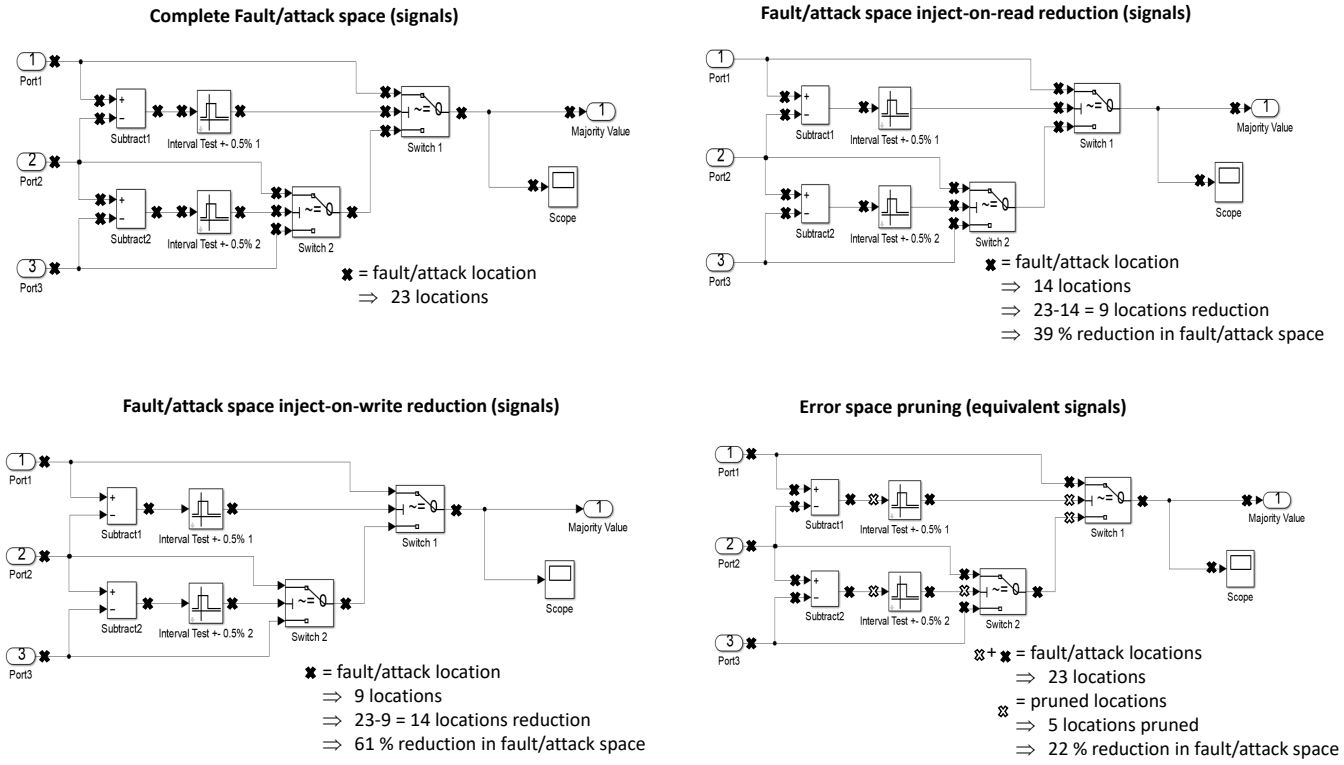

Figure 1. Complete error space of an example Simulink model (top left) reduced by inject-on-read (top right), inject-on-write (bottom left) and error space pruning of signals (bottom right).

resulting in a total of 23 locations, which is identical to the the case where no pre-injection analysis is used (see the top left figure). Therefore, in the error space pruning of signals (see the bottom right figure), *we propose an analysis that combines the benefit of inject-on-read and inject-on-write (reduction of the error space), while assuring that all faults and attacks modelled by these techniques are investigated.*

The error space pruning of signals works by considering *faults and attacks on input signals to be equivalent to those on an output signal if only one propagation path exists between the input and output signal.* A static analysis of the target system structure is performed in order to determine the equivalence. In the case of the simple model shown in the bottom right corner of Figure 1, five input locations are pruned, resulting in a reduction of 22%. Note that, either the input signal or the equivalent output signal can be pruned since they represent equivalent target locations. Also note that, similar to when the inject-on-read or inject-on-write are used, and as discussed in the past work [20, 5, 2], the results obtained from the injection of faults and attacks into the equivalent classes should be extrapolated to the pruned locations.

C. Simulink Models under Evaluation and the Attack Injection Campaign Setup

1) Lock and Comfort Control Model: Figure 2 shows an overview of the LockComfort Simulink model mainly used for comfort control, i.e., controlling the door windows and sun roof, for various vehicle types and models. The model has four main blocks, the `1_InteriorButton`, `2_Remote`, and `3_Keyless` blocks allowing comfort control from buttons inside the vehicle, the remote car key, and the door handles, respectively, and the `4_ComfortRequest` block which produces the required output requests based on the input from the other blocks.

The following input signals are the most relevant for the chosen test case: `CarCfg` denoting the vehicle type, `Lockd_BtnSts` indicating lock button pressed (value 1) or not (value 0), `Lock_KeyId` indicating lock button key identifier, `Unlock_BtnSts` indicating unlock button pressed (value 1) or not (value 0), `Unlock_KeyId` indicating unlock button key identifier, `UsgModSts` indicating current usage mode of the vehicle using the five enumerated values *Abandoned*, *Inactive*, *Convenience*, *Active* and *Driving*, `Pinch protection level` to indicate obstructions when closing the window and `Chassis type` to disable comfort opening/closing for certain types of chassis.

The model produces two output signals for the comfort open/close requests: the `OpenClsCmftReq` signal corresponding to the comfort open/close request using the values 0 (default value), 1 (comfort open request), 2 (comfort close request) and 3 (stop ongoing comfort open/close movement), and the `OpenClsCmftTrigSrc` signal corresponding to the source of the open/close request using the values 0 (default value), 1 (`1_InteriorButton` block), 2 (`2_Remote`

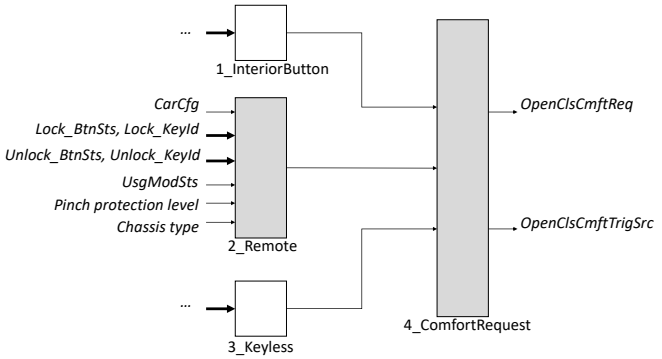

Figure 2. Overview of the Lock and Comfort Control Model (LockComfort)

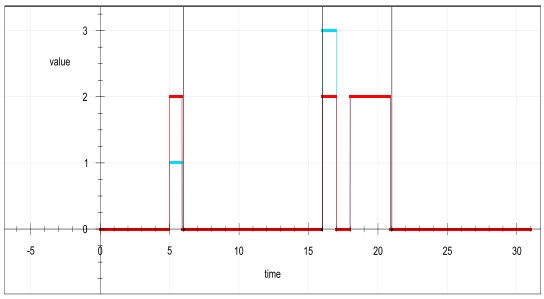

Figure 3. The output signals `OpenClsCmftReq` (light blue) and `OpenClsCmftTrigSrc` (dark red) as functions of time for the chosen test case. Attacks are injected at 6, 16 and 21 seconds, indicated by black vertical lines.

block) and 3 (`3_Keyless` block). `Lockd_BtnSts` must be 1 for more than 2s for the `OpenClsCmftReq` to be set to 2 (close) while `Unlock_BtnSts` must be 1 for more than 4s for the `OpenClsCmftReq` to be set to 1 (open).

For the LockComfort model test case, the `CarCfg` is set to a typical road vehicle and the `Chassis type` default value is used to allow full comfort control functionality. The `UsgModSts` is set to Convenience.

The following sequence is generated by using the input signals indicated below as stimuli for the model for the chosen test case:

1) Press the unlock button with `Unlock_BtnSts`=1 and `Unlock_KeyId`=1 during time 1s-5.9s.
2) Wait 10s.
3) Press the lock button with `Lock_BtnSts`=1 and `Lock_KeyId`=1 during time 16s-20.9s.
4) Wait 10s.

A security violation occurs if the window does not close during steps 3 to 4.

Figure 3 shows the two output signals `OpenClsCmftReq` (light blue) and `OpenClsCmftTrigSrc` (dark red) as functions of time for the chosen test case. Due to the input sequence described above, the `OpenClsCmftReq` is equal to 1 (i.e., open) between 5s and 5.9s, since the unlock button is pressed between 1s and 5.9s; and then `OpenClsCmftReq` is equal to 3 (i.e., stop) at 16s and ending at 17s. Finally, `OpenClsCmftReq` is equal to 2 (i.e., close) between 18s and 20.9s, since the lock button is pressed between 16s and 20.9s. `OpenClsCmftTrigSrc` is equal to 2 (corresponding to `2_Remote` block) whenever the `OpenClsCmftReq` signal is activated.

Three attack injection campaigns have been chosen for each of the three evaluated pre-injection analyses to be injected after 6, 16 and 21 seconds from the beginning of the simulation. For comparison, an initial set of campaigns has also been conducted without the use of any pre-injection analyses at each of the chosen time points, resulting in a total of 12 attack injection campaigns. In each of the campaigns, attack injection is performed exhaustively on all possible locations of the security-critical parts of the model featuring comparatively large attack surfaces, i.e., the `2_Remote` and `4_ComfortRequest` blocks (highlighted in gray in Figure 2).

The pre-injection techniques evaluated can be applied for any fault- or attack model as long as they target single signal locations. However, as different fault- or attack models may target different types of signals, the error space may vary depending on fault- or attack model used. The `InvertMessage` attack model, see Section III-A, was used in these experiments since it may have significant impact on the targeted signals and there are no random parameters controlling the behaviour of this model which could skew the results when comparing the techniques. The error space reduction achieved using this model is representative since the types of signals targeted by this model is identical to the types targeted by most other fault- or attack models (bit-flips, stuck-at faults, replay attacks, jamming attacks etc.). However, the user may also select a subset of the the complete error space depending on the fault- or attack model employed. This was not considered in our experiments where all possible signals were targeted.

2) Brake-By-Wire Model (BBW): Figure 4 shows a structural overview of the BBW target system Simulink model which has been used extensively in previous research [15, 5, 27, 28, 14]. The model implements a four-wheel BBW application developed using five ECUs (Electronic Control Units) connected to a bus. 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 the 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.

The output with the chosen test case for the BBW model is `VehSpdEst` signal that reports the estimated vehicle speed. Figure 5 shows the nominal (attack-free) value of `VehSpdEst` during the simulated time. The test case simu-

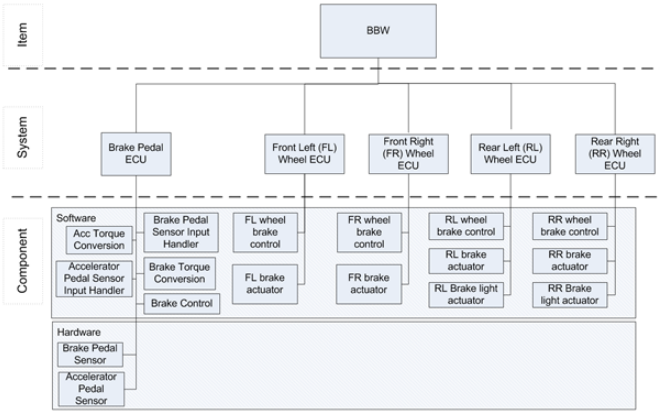

Figure 4. Structural overview of the BBW target system.

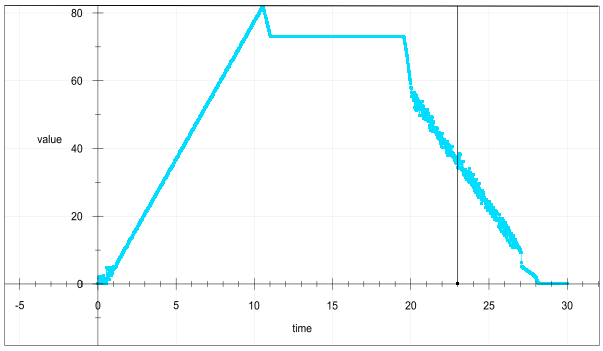

Figure 5. The nominal (attack-free) speed (`VehSpdEst` output signal) as a function of time for BBW. Attacks are injected at 23 seconds as indicated by the vertical line.

lates acceleration of the vehicle between 0s and 10s followed by a constant speed between 10s and 20s. Braking of the vehicle is performed from 20s until it stops after 28s. Previous experiments (e.g., see [15, 14]) have focused on injecting faults and attacks targeting one of the wheel nodes at multiple control loop iterations around the simulation time point of 23s. However, in our experiments, attack injection has been performed exhaustively on *all* possible locations of the target system. To reduce the number of experiments, a single control loop corresponding to the simulation time point at exactly 23s (see vertical line in Figure 5) was injected.

One attack injection campaign was conducted for each of the three evaluated pre-injection analysis techniques. For comparison, one campaign has also been conducted without the use of any pre-injection analysis. The `InvertMessage` attack model is used for the same reason as for the LockComfort model, see Section III-C1.

IV. Experimental Results

This section presents the results of our evaluations when it comes to the measurement of the efficiency of the three pre-injection analyses presented in Section III-B. To this end, the pre-injection analyses were conducted on 11 vehicular Simulink models and the results are presented in Section IV-A. Moreover, to provide additional insights and details about the pre-injection analysis techniques, we performed attack injection experiments on the two target systems presented in Section III-C. The results of the experiments are presented in Table I and Table III classified by the following columns:

- **Campaign**: The name of the campaign indicating *target system model*, *point in time* for attack injection and *type of pre-injection analysis* performed.
- **Signals**: The total number of injectable locations (signals) in the target system blocks.
- **Input signals**: The total number of injectable input signals in the target system blocks.
- **Output signals**: The total number of injectable output signals in the target system blocks.
- **Attacks performed**: The total number of attacks injected (for exhaustive campaigns, this should correspond to the number of injectable locations).
- **Attacks logged**: The total number of attacks logged. Note that, when using the error space pruning techniques, the total number of attacks logged could be larger than the total number of attacks performed. This is due to the fact that the result obtained for each injected attack is also considered for the corresponding equivalent attacks.
- **SDC**: The total number of attacks leading to Silent Data Corruption, i.e., erroneous values on any of the output signals of the model.
- **Detected**: The total number of attacks leading to errors being detected. Since there are no error detection mechanisms implemented in the target system models, only errors detected by the MATLAB/Simulink environment are included.

Note that, an injected attack could also have no impact on a Simulink model's output nor trigger any detection mechanism. For these attacks, a classification category that is known as *No Impact* [11] could be calculated as $Attackslogged - (SDC + Detected)$. The result classification categories used in this paper is similar to those used in the past work such as [5, 29, 11]. Moreover, the SDC, Detected, and No Impact results obtained could be used to measure target system dependability metrics such as error coverage [30, 31], error resiliency [32, 33], and error sensitivity [34, 29]. However, we do not measure these metrics in this paper as we focus on the evaluation of the pre-injection analysis techniques.

A. Comparison of Error Space Reduction

The pre-injection techniques implemented in MODIFI can be applied regardless of target system and fault/attack model as long as the system contains signals which can be targeted by fault- or attack injection. Figure 6 shows a comparison of the reduction of the error space using each of the implemented pre-injection analysis techniques for nine of the MathWorks Simulink Examples Automotive Applications models available at [26] as well as the two vehicular Simulink models presented in Section III-C. The MathWorks models targeted include Anti-Lock Braking System (`absbrake`), Automatic Climate Control System (`auto_climatecontrol`), Automatic Transmission Controller (`autotrans`), Clutch

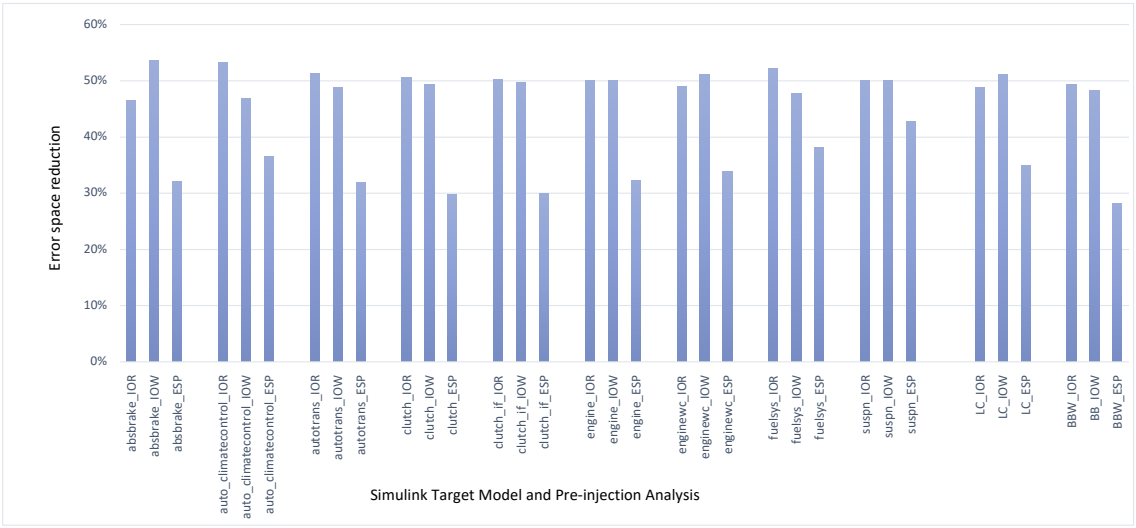

Figure 6. Comparison of error space reduction using inject-on-read (IOR), inject-on-write (IOW) and error space pruning of signals (ESP) pre-injection techniques for nine MathWorks Simulink automotive example models together with the LockComfort (LC) and Brake-By-Wire (BBW) models.

Lock-Up Model (`clutch`), Clutch Lock-Up Model Using If Blocks (`clutch_if`), Engine Timing Using Triggered Subsystems (`engine`), Engine Timing Model with Closed Loop Control (`engine_wc`), Fault-Tolerant Fuel Control System (`fuelsys`), and Automotive Suspension (`suspn`). The other models in the MathWorks Automotive Applications repository rely on software components which were not included in the experimental set up used for the comparison.

The results presented in Figure 6 show a reduction of the error space of between 46–53% for the inject-on-read technique, 47–54% for the inject-on-write technique and 30–43% for the error space pruning of signals technique. In the remainder of this section, we use MODIFI to inject attacks into the *comfort control* and *brake-by-wire* models and provide additional details about the results obtained while comparing the pre-injection analysis techniques.

B. Results Obtained for the Lock and Comfort Control Model

Table I shows the results of attack injection experiments conducted for the `LockComfort` test case. Rows 1 to 3 of the table show the results of exhaustive attack injection campaigns at the simulation time points 6s, 16s and 21s, respectively, without any pre-injection analyses performed. For each set of experiments (i.e., campaigns), 457 signals were targeted by the `InvertMessage` attacks where 126 to 186 of them were classified as SDCs in these campaigns.

Rows 4 to 6 of Table I show the results of exhaustive attack injection campaigns using the inject-on-read technique. The number of signals targeted is 234, corresponding to 51% of the error attack space (consisting of 457 signals). Due to the exploitation of the partial error space using the inject-on-read technique, signals which result in errors being detected by the simulation are never attacked in any of the campaigns performed with this technique, and thus, never tested. This anomaly is highlighted in the Detected column of the table.

Rows 7 to 9 of Table I show the results of exhaustive attack injection campaigns using the inject-on-write technique. The number of injected signals is 223, corresponding to 49% of the entire error space. Based on the results presented in rows 4–9, we can also observe that the error space exploited using inject-on-write results in a higher number of SDCs compared to that exploited by the inject-on-read technique.

The results obtained for the inject-on-read and inject-on-write pre-injection analyses suggest that in order to have more complete data about the sensitivity of the target system to attacks on all locations, the results obtained for inject-on-read should be accompanied with those obtained for the inject-on-write technique.

Rows 10 to 12 of Table I show the results of exhaustive attack injection campaigns using the error space pruning of signals technique. The number of signals targeted by attacks is reduced from 457 to 297, corresponding to a 35% reduction of the error space. For each injected attack, the corresponding equivalent attacks are logged and the result of each injected attack is also considered for the corresponding equivalent attacks.

The time measured for doing the pre-injection analysis is negligible (in the order of just a few seconds) compared to the execution time of the attack injection campaigns, which went down from about 40 minutes in the case of no pre-injection analysis to 26 minutes for the error space pruning of signals technique. This results in being able to evaluate target systems when using the error space pruning of signals technique with about 35% less effort.

We also conducted an additional classification of the SDC errors in order to investigate the differences in the behaviour of the `LockComfort` test case when attacks are injected with and without pre-injection analyses. Two classes were used, corresponding to errors in each of the output signals listed in Section III-C1. The classification of errors on

Table I
RESULTS OF ATTACK INJECTION FOR LOCKCOMFORT TEST CASE.

Campaign	Signals	Input signals	Output signals	Attacks performed	Attacks logged	SDC	Detected
LC_6s	457	234	223	457	457	180	2
LC_16s	457	234	223	457	457	126	2
LC_21s	457	234	223	457	457	186	2
LC_6s_IOR	234	234	0	234	234	84	0
LC_16s_IOR	234	234	0	234	234	59	0
LC_21s_IOR	234	234	0	234	234	86	0
LC_6s_IOW	223	0	223	223	223	96	2
LC_16s_IOW	223	0	223	223	223	67	2
LC_21s_IOW	223	0	223	223	223	100	2
LC_6s_ESP	297	74	223	297	457	180	2
LC_16s_ESP	297	74	223	297	457	126	2
LC_21s_ESP	297	74	223	297	457	186	2

Table II
RESULTS ACCORDING TO ERROR CLASSIFICATION FOR LOCKCOMFORT TEST CASE. NOTE THAT, *n* IN A*n* AND B*n* INDICATE THE ERRONEOUS VALUE OF THE OPENCLSCMFTREQ AND OPENCLSCMFTTRIGSRC OUTPUT SIGNALS, RESPECTIVELY.

Campaign	Class A0	Class A1	Class A2	Class A3	Class B0	Class B1	Class B2	Class B3
LC_6s	0	57	41	54	0	20	4	4
LC_16s	87	3	0	1	31	4	0	0
LC_21s	0	57	44	57	0	20	4	4
LC_6s_IOR	0	25	20	27	0	8	2	2
LC_16s_IOR	40	1	0	1	15	2	0	0
LC_21s_IOR	0	25	21	28	0	8	2	2
LC_6s_IOW	0	32	21	27	0	12	2	2
LC_16s_IOW	47	2	0	0	16	2	0	0
LC_21s_IOW	0	32	23	29	0	12	2	2
LC_6s_ESP	0	57	41	54	0	20	4	4
LC_16s_ESP	87	3	0	1	31	4	0	0
LC_21s_ESP	0	57	44	57	0	20	4	4

the `OpenClsCmftReq` output signal are denoted as A0 to A3 (or A*n* where *n* indicates the erroneous value of the `OpenClsCmftReq` output signal). The classification of errors on the `OpenClsCmftTrigSrc` output signal are denoted as B0 to B3 (or B*n* where *n* indicates the erroneous value of the `OpenClsCmftTrigSrc` output signal). Table II shows the number of SDC errors for each of the error classes defined.

Table II shows that the errors belonging to each class vary significantly depending on the point in time for attack injection. For example, when no pre-injection analysis is used, 0 errors belong to Class A0 for the campaigns performed at 6s and 21s, compared to 87 errors for the campaign performed at 16s. This shows the significance of the test scenario selection as well as the fault injection campaign setup when evaluating target systems, as the sensitivity of the target system to attacks could vary significantly throughout time. The table also shows that the number of SDC errors belonging to the classes could vary significantly. For example, the errors belonging to Class A1 and B3 for when no pre-injection is used are 117 and 8, respectively. This could be a valuable piece of information for system testers that are after identification of parts of the system or system configurations that are more sensitive to attacks.

Looking at the inject-on-write results in Table II, we observe that no errors belong to Class A3 for the injections performed at 16s although there is 1 error classified in this class when no pre-injection was used. This anomaly is highlighted in the table and the reason for it, as discussed before, is that for inject-on-read and inject-on-write only a subset of possible attacks is targeted. The table also shows that the error space pruning technique is successful in classifying all the errors in their respective classes, as the results obtained are identical to when no pre-injection analysis is used.

C. Results Obtained for the Brake-By-Wire Model (BBW)

Table III presents the attack injection results obtained for the BBW test case. The first row in the table shows the results of an exhaustive attack injection campaign when no pre-injection analysis was used. The injections are conducted using the `InvertMessage` attack model at simulation time point 23s. In this test case, 691 signals were targeted by attacks resulting in a total of 525 SDC errors.

The second and third rows of the table show the results of exhaustive attack injection campaigns when using the inject-on-read and inject-on-write techniques, respectively. The number of signals targeted are 344 and 347, respectively, each corresponding to about 50% of the entire error space that contains 691 signals. This reduction in the error space for each of the techniques is comparable to that caused by these techniques in the `LockComfort` test case. Similar to the results presented for the `LockComfort` test case, attacks injected using the inject-on-write technique result in a higher number of SDC errors.

The last row of the table shows the results of exhaustive attack injection when using the error space pruning of signals technique. The number of signals targeted by attacks is

Table III
RESULTS OF ATTACK INJECTION FOR BBW TEST CASE.

Campaign	Signals	Input signals	Output signals	Attacks performed	Attacks logged	SDC	Detected
BBW_23s	691	344	347	691	691	525	0
BBW_23s_IOR	344	344	0	344	344	248	0
BBW_23s_IOW	347	0	347	347	347	277	0
BBW_23s_ESP	486	139	347	486	691	525	0

Table IV
RESULTS ACCORDING TO ERROR CLASSIFICATION FOR BBW TEST CASE.

Campaign	Class A	Class B	Class C
BBW_23s	65	451	9
BBW_23s_IOR	30	215	3
BBW_23s_IOW	35	236	6
BBW_23s_ESP	65	451	9

reduced from 691 to 486 corresponding to a 30% reduction of the error space. For each injected attack, the corresponding equivalent attacks are logged and the result of the injected attack is also considered for the corresponding equivalent attacks.

Similar to what is observed for the `LockComfort` test case, the time measured for pre-injection analysis of the BBW system is negligible compared to the execution time of the attack injection campaigns. In this case, this results in being able to evaluate target systems with respect to both inject-on-read and inject-on-write techniques with about 30% less effort.

Further classification of the SDC errors, corresponding to the `VehSpdEst` output signal, has been made in order to investigate any behavioral differences of the BBW target system model between the attack injection campaigns. To conduct this classification, we define the following three error classes, where the first two are inspired from the result classifications used in the past [15, 14] in which SDC errors are further classified into *severe* and *benign* classes, respectively:

- Class A: `VehSpdEst` $<$ (nominal value $-$ 10 km/h) or `VehSpdEst` $>$ (nominal value $+$ 10 km/h).
- Class B: (nominal value $-$ 10 km/h) $\leq$ `VehSpdEst` $\leq$ (nominal value $+$ 10 km/h).
- Class C: `VehSpdEst` $=$ nominal value, but one or more of the other output signals are affected.

Table IV shows the number of SDC errors for each of the error classes defined. Similar to the results presented in Table II, here we see that the error space pruning technique is successful in classifying all the errors in their respective classes, as the results obtained are identical to when no pre-injection analysis is used. The table also shows that between 12–13% of the errors are those that are severe, belonging to Class A. Note that, as discussed in prior research [14], the choice of 10 km/h in the above classes 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.

V. CONCLUSIONS AND FUTURE WORK

This paper evaluates three pre-injection analysis techniques facilitating effective testing of systems modelled in Simulink in the presence of faults and attacks. Two of these analysis techniques (*inject-on-read* and *inject-on-write*) are inspired by the previous work (although we are the first to integrate them into a model-implemented attack injector) and one is a new analysis technique (*error space pruning of signals*) proposed and evaluated for the first time in this paper. The main idea behind the pre-injection analyses is to reduce the error space in order to focus on injections that are known to be potentially effective, e.g., in penetrating the systems' resilience. An experimental evaluation of the pre-injection analyses is presented by analysing results from model-implemented attack injection campaigns performed for the three pre-injection analysis techniques with two vehicular Simulink target system models, a *comfort control model* and a *brake-by-wire (BBW) model*.

The results for the comfort control model show that the error space is reduced to 51% of the entire error space for inject-on-read analysis and to 49% for the inject-on-write analysis. However, due to the exploitation of the partial error space using these pre-injection techniques, several anomalies were observed. For inject-on-read, signals which result in errors being detected by the simulation are never attacked and thus, never tested. For inject-on-write, errors belonging to certain classes are sometimes never triggered although they are triggered when no pre-injection analysis is performed. The results obtained for the inject-on-read and inject-on write pre-injection analyses suggest that in order to have more complete data about the sensitivity of the target system to attacks on all locations, the results obtained for inject-on-read should be accompanied with those obtained for the inject-on-write technique. However, as this would result in an error space identical to the case when no pre-injection analysis is done, for the target systems and attack injection technique used in this study, a pre-injection analysis techniques called error space pruning of signals has been designed, implemented and evaluated.

When using the error space pruning of signals with the comfort control model, a reduction of the error space of 35% could be observed. This technique also logs the results of the injected attacks for the corresponding equivalent attacks pruned. Results identical to the results without any pre-injection analysis were obtained for this technique. Thus, error space pruning of signals allows the target systems to be

evaluated with respect to using both inject-on-read and inject-on-write techniques with about 35% less effort.

The results for the comfort control model are corroborated by the experiments performed on the BBW model. The results for the BBW model show a 50% reduction of the error space for both inject-on-read and inject-on-write analyses and a 30% reduction for error space pruning of signals. The results also indicate that a lower percentage of attacks may have the potential of testing the resilience mechanisms using inject-on-read than without any pre-injection analyses, while the opposite is true for the inject-on-write analysis. No anomalies were observed for the BBW model when comparing the results using pre-injection analysis with the results obtained without pre-injection analysis. However, deeper analysis may reveal the existence of such anomalies for the BBW model as well.

Comparison with nine of the automotive example models available on the Mathworks web site shows a similar reduction in the error space as for the comfort control and BBW models. In fact, the results of our analyses show that the entire error space could be efficiently explored using the error space pruning of signals with about 30% to 43% reduction in the number of experiments needed to be conducted. This optimisation comes with a minimal analysis overhead of just a few seconds, which is negligible compared to the time it takes to conduct the injection experiments. Note that although all the models under test in this paper are taken from the automotive domain, the pre-injection analysis techniques proposed and evaluated in the paper could be used to evaluate Simulink models from other relevant domains. In fact, as part of our future work, we plan on evaluating the proposed pre-injection analysis techniques using Simulink models of an engine controller from the avionics domain.

As part of the future work, we plan on investigating pre-injection analysis techniques exploring the use of other equivalent fault- and attack classes. Moreover, post-injection analyses performed after fault- and attack injection campaigns to predict or determine relevant injection experiments were not considered but should also be investigated, e.g., based on sensitivity profiling or error propagation analyses identifying the parts of the error space with the highest probability of penetrating the resilience layers. Also, only single faults and attacks have been considered so far, the usefulness of the proposed techniques for multiple faults and attacks or sequences of attacks have not yet been investigated.

Acknowledgement

This work was partly funded by the CyReV project, which is a Swedish VINNOVA FFI project (Diary number: 2018-05013, 2019-03071); and the VALU3S project, which has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 876852. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Austria, Czech Republic, Germany, Ireland, Italy, Portugal, Spain, Sweden, Turkey.

References

[1] P. Kleberger, P. Folkesson, and B. Sangchoolie. "An Integrated Safety and Cybersecurity Resilience Framework for the Automotive Domain". In: *CARS - 7th International Workshop on Critical Automotive Applications: Robustness & Safety*. 2022.

[2] R. Barbosa, J. Vinter, P. Folkesson, and J. Karlsson. "Assembly-Level Pre-injection Analysis for Improving Fault Injection Efficiency". In: *Dependable Computing - EDCC 5*. Ed. by M. Dal Cin, M. Kaâniche, and A. Pataricza. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 246–262. ISBN: 978-3-540-32019-7.

[3] G. Munkby and S. Schupp. "Improving Fault Injection of Soft Errors Using Program Dependencies". In: *Testing: Academic Industrial Conference - Practice and Research Techniques (taic part 2008)*. 2008, pp. 77–81.

[4] J. Grinschgl, A. Krieg, C. Steger, R. Weiss, H. Bock, and J. Haid. "Efficient fault emulation using automatic pre-injection memory access analysis". In: *2012 IEEE International SOC Conference*. 2012, pp. 277–282. DOI: 10.1109/SOCC.2012.6398361.

[5] 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. DOI: 10.1109/EDCC.2015.24.

[6] A. C. Bagbaba, M. Jenihhin, J. Raik, and C. Sauer. "Efficient Fault Injection based on Dynamic HDL Slicing Technique". In: *CoRR* abs/2002.00787 (2020). arXiv: 2002.00787.

[7] L. Berrojo, I. Gonzalez, F. Corno, M. Reorda, G. Squillero, L. Entrena, and C. Lopez. "New techniques for speeding-up fault-injection campaigns". In: *Proceedings 2002 Design, Automation and Test in Europe Conference and Exhibition*. 2002, pp. 847–852. DOI: 10.1109/DATE.2002.998398.

[8] D. Smith, B. Johnson, and J. Profeta. "System dependability evaluation via a fault list generation algorithm". In: *IEEE Transactions on Computers* 45.8 (1996), pp. 974–979. DOI: 10.1109/12.536240.

[9] S. K. S. Hari, S. V. Adve, H. Naeimi, and P. Ramachandran. "Relyzer: Exploiting Application-Level Fault Equivalence to Analyze Application Resiliency to Transient Faults". In: *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS XVII. London, England, UK: Association for Computing Machinery, 2012, pp. 123–134. ISBN: 9781450307598. DOI: 10.1145/2150976.2150990.

[10] F. Ayatolahi, B. Sangchoolie, R. Johansson, and J. Karlsson. "A Study of the Impact of Single Bit-Flip and Double Bit-Flip Errors on Program Execution". In: *Computer Safety, Reliability, and Security*. Ed. by F. Bitsch, J. Guiochet, and M. Kaâniche. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 265–276. ISBN: 978-3-642-40793-2.

[11] B. Sangchoolie, K. Pattabiraman, and J. Karlsson. "An Empirical Study of the Impact of Single and Multiple Bit-Flip Errors in Programs". In: *IEEE Transactions on Dependable and Secure Computing* 19.3 (2022), pp. 1988–2006.

[12] I. Tuzov, D. de Andres, and J.-C. Ruiz. "Reversing FPGA architectures for speeding up fault injection: does it pay?" In: *18th European Dependable Computing Conf.* 2022.

[13] R. Svenningsson, J. Vinter, H. Eriksson, and M. Törngren. "MODIFI: A MODel-Implemented Fault Injection Tool". In: *Computer Safety, Reliability, and Security*. Ed. by E. Schoitsch. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 210–222. ISBN: 978-3-642-15651-9.

[14] B. Sangchoolie, P. Folkesson, and J. Vinter. "A study of the interplay between safety and security using model-implemented fault injection". In: *2018 14th European Dependable Computing Conference (EDCC)*. IEEE. 2018, pp. 41–48.

[15] 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. ISBN: 978-3-319-24255-2.

[16] D. Skarin, J. Vinter, and R. Svenningsson. "Visualization of Model-Implemented Fault Injection Experiments". In: *Computer Safety, Reliability, and Security*. Ed. by A. Bondavalli, A. Ceccarelli, and F. Ortmeier. Cham: Springer International Publishing, 2014, pp. 219–230. ISBN: 978-3-319-10557-4.

[17] G. Juez, E. Amparan, R. Lattarulo, A. Ruíz, J. Pérez, and H. Espinoza. "Early Safety Assessment of Automotive Systems Using Sabotage Simulation-Based Fault Injection Framework". In: *Computer Safety, Reliability, and Security*. Ed. by S. Tonetta, E. Schoitsch, and F. Bitsch.

Cham: Springer International Publishing, 2017, pp. 255–269. ISBN: 978-3-319-66266-4.

[18] I. Pill, I. Rubil, F. Wotawa, and M. Nica. “SIMULTATE: A Toolset for Fault Injection and Mutation Testing of Simulink Models”. In: *2016 IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. 2016, pp. 168–173. DOI: 10.1109/ICSTW.2016.21.

[19] *Fault Injection Block (FIBlock)*. https://se.mathworks.com/matlabcentral/fileexchange/75539-fault-injection-block-fiblock. Accessed: 2022-06-15.

[20] H. Schirmeier, C. Borchert, and O. Spinczyk. “Avoiding Pitfalls in Fault-Injection Based Comparison of Program Susceptibility to Soft Errors”. In: *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. 2015, pp. 319–330. DOI: 10.1109/DSN.2015.44.

[21] A. Avizienis, J.-C. Laprie, and B. Randell. *Fundamental Concepts of Dependability*. 739. Newcastle upon Tyne: Department of Computing Science, University of Newcastle upon Tyne, 2001, p. 21.

[22] E. W. Czeck and D. P. Siewiorek. “Observations on the Effects of Fault Manifestation as a Function of Workload”. In: *IEEE Trans. Comput.* 41.5 (May 1992), pp. 559–566. DOI: 10.1109/12.142682.

[23] J. Aidemark, P. Folkesson, and J. Karlsson. “Path-based error coverage prediction”. In: *Proceedings Seventh International On-Line Testing Workshop*. 2001, pp. 14–20. DOI: 10.1109/OLT.2001.937811.

[24] P. Folkesson and J. Karlsson. “Considering Workload Input Variations in Error Coverage Estimation”. In: *Proceedings of the Third European Dependable Computing Conference on Dependable Computing*. EDCC-3. Berlin, Heidelberg: Springer-Verlag, 1999, pp. 171–190. ISBN: 3540664831.

[25] B. Sangchoolie, K. Pattabiraman, and J. Karlsson. “One Bit is (Not) Enough: An Empirical Study of the Impact of Single and Multiple Bit-Flip Errors”. In: *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 2017, pp. 97–108. DOI: 10.1109/DSN.2017.30.

[26] *Simulink - Examples Automotive Applications*. https://se.mathworks.com/help/simulink/examples.html. Accessed: 2022-06-14.

[27] *TIMMO - Timing Model project*. http://adt.cs.upb.de/timmo-2-use/timmo/index.htm. Accessed: 2018-03-21.

[28] *VeTeSS - Verification and Testing to Support Functional Safety Standards project*. https://artemis-ia.eu/project/43-vetess.html. Accessed: 2018-03-21.

[29] B. Sangchoolie, R. Johansson, and J. Karlsson. “Light-Weight Techniques for Improving the Controllability and Efficiency of ISA-Level Fault Injection Tools”. In: *2017 IEEE 22nd Pacific Rim International Symposium on Dependable Computing (PRDC)*. 2017, pp. 68–77. DOI: 10.1109/PRDC.2017.18.

[30] W. G. Bouricius, W. C. Carter, and P. R. Schneider. “Reliability Modeling Techniques for Self-Repairing Computer Systems”. In: *Proceedings of the 1969 24th National Conference*. ACM ’69. New York, NY, USA: Association for Computing Machinery, 1969, pp. 295–309. ISBN: 9781450374934.

[31] T. Arnold. “The Concept of Coverage and Its Effect on the Reliability Model of a Repairable System”. In: *IEEE Transactions on Computers* C-22.3 (1973), pp. 251–254. DOI: 10.1109/T-C.1973.223703.

[32] B. Fang, Q. Lu, K. Pattabiraman, M. Ripeanu, and S. Gurumurthi. “ePVF: An Enhanced Program Vulnerability Factor Methodology for Cross-Layer Resilience Analysis”. In: *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 2016, pp. 168–179.

[33] Q. Lu, M. Farahani, J. Wei, A. Thomas, and K. Pattabiraman. “LLFI: An Intermediate Code-Level Fault Injection Tool for Hardware Faults”. In: *2015 IEEE International Conference on Software Quality, Reliability and Security*. 2015, pp. 11–16. DOI: 10.1109/QRS.2015.13.

[34] B. Sangchoolie, F. Ayatolahi, R. Johansson, and J. Karlsson. “A Study of the Impact of Bit-Flip Errors on Programs Compiled with Different Optimization Levels”. In: *2014 Tenth European Dependable Computing Conference*. 2014, pp. 146–157. DOI: 10.1109/EDCC.2014.30.