RESEARCH-ARTICLE

SCVMON: Data-oriented attack recovery for RVs based on safety-critical variable monitoring

SANGBIN PARK, Korea University, Seoul, South Korea

YOUNGJOON KIM, Korea University, Seoul, South Korea

DONGHOON LEE, Korea University, Seoul, South Korea

Published: 16 October 2023

RAID 2023: The 26th International Symposium on Research in Attacks, Intrusions and Defenses
October 16 - 18, 2023
Hong Kong, China

https://doi.org/10.1145/3607199.3607221
ISBN: 9798400707650

SCVMON: Data-oriented attack recovery for RVs based on safety-critical variable monitoring

Sangbin Park
allo120@korea.ac.kr
Korea University
Seoul, Republic of Korea

Youngjoon Kim
acorn421@korea.ac.kr
Korea University
Seoul, Republic of Korea

Dong Hoon Lee
donghlee@korea.ac.kr
Korea University
Seoul, Republic of Korea

ABSTRACT

There are many various data-oriented attacks on robotic vehicles (RVs) that change the inputs of an RV control program. While much research has been dedicated to detecting the attacks, the recovery mechanism has received relatively less attention. Without recovery after detection, an RV cannot continue with its assigned missions. Unfortunately, the existing recovery mechanisms have limitations that make it difficult to deploy these in real RVs, such that they require additional hardware/software or can only recover from the limited types of data-oriented attacks. To overcome these limitations, we propose a framework called SCVMON that detects and helps RVs recover from various data-oriented attacks that generate inappropriate control commands. Based on the observation that data-oriented attacks inevitably change the values of some variables in RV control programs, SCVMON systematically identifies the safety-critical variables (SCVs) that can affect the safety of RVs. For efficient recovery, we extract from SCVs a set of monitored safety-critical variables (mSCVs) that can reflect all input changes, and monitor them to detect and recover from various data-oriented attacks. SCVMON does not depend on the physical nature of a specific sensor or hardware, which is a significant benefit, and it can be applied through a simple software update. Our evaluation shows that SCVMON can quickly detect and recover from 20 types of data-oriented attacks. Also, SCVMON incurs only 0.3% storage overhead and up to 5.1% runtime overhead, proving that it is suitable for RVs.

CCS CONCEPTS

• **Computer systems organization → Embedded and cyber-physical systems**; • **Security and privacy → Software and application security**.

KEYWORDS

CPS security, Robotic vehicle, Attack detection and recovery, Data-oriented attack, Safety-critical variables

ACM Reference Format:

Sangbin Park, Youngjoon Kim, and Dong Hoon Lee. 2023. SCVMON: Data-oriented attack recovery for RVs based on safety-critical variable monitoring. In *The 26th International Symposium on Research in Attacks, Intrusions and Defenses (RAID '23), October 16–18, 2023, Hong Kong, China.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3607199.3607221

RAID '23, October 16–18, 2023, Hong Kong, China

ACM ISBN 979-8-4007-0765-0/23/10…$15.00
https://doi.org/10.1145/3607199.3607221

1 INTRODUCTION

Robotic vehicles (RVs), which have traditionally been used in limited applications such as military missions, are spreading throughout the industrial sector. However, various regulations have been imposed based on safety concerns[17, 49, 53, 56] regarding RVs, hindering the widespread adoption of these vehicles. In the event that there is a safety issue in an RV, collisions or crashes can occur, which can result in physical (property) damage or human casualties[6–9].

There are various methods that detect and prevent attacks[12, 14, 18, 27, 29, 30, 32, 33, 45] to ensure the safety of RVs. However, attack recovery has not received significant attention. This is problematic because an RV without recovery mechanisms cannot continue its assigned missions even if it successfully detects an attack. In previous studies[14, 18, 27, 45], the mission is immediately suspended after an attack is detected, meaning that an attacker can conduct attacks on the target RVs with minimal resources. This ultimately allows attackers to force RVs to abandon urgent missions, which may be the attacker's main goal. In this study, we consider data-oriented attacks that generate inappropriate control commands by changing the values of various inputs of the RV control program without violating its control flow. There are mainly two types of data-oriented attacks against RVs: sensor attacks that change sensor measurements and parameter attacks that change the values of configuration parameters. These data-oriented attacks are common and most frequently performed on RVs[10, 40].

Recently, several recovery mechanisms have been proposed, but there are two limitations. First, some of them require additional software/hardware[20, 55]. Although redundancy using additional software/hardware is a common and effective approach for recovery, it may not be appropriate to apply this approach in RVs, which is a real-time system. Operating additional software for recovery separately from existing RV control programs may cause substantial overhead, which is an important factor in RVs. Additionally, approaches that leverage redundant hardware incur additional costs to apply and maintain. Furthermore, hardware support such as a trusted execution environment [5, 35, 42] may not be available in RVs. Therefore, applying this approach to RVs may not be feasible in terms of efficiency and cost[23]. Second, the rest of the previous mechanisms [13, 16, 25, 47, 57] can only recover from one type of data-oriented attacks (that is, sensor attacks), but not parameter attacks. This implies that the previous mechanisms cannot properly recover from parameter attacks or attacks combining sensor attacks and parameter attacks.

One straightforward way to provide recovery may be to simply extend previous detection mechanisms. However, this way may not provide adequate recovery since the factors used in the detection mechanisms are selected without considering recovery. For example, actuator outputs can be utilized as factors in detection mechanisms. However, since actuator outputs are generated through complex calculations involving various inputs, it may not be possible to generate accurate predictions for recovery. Therefore, it may be inappropriate to use these detection factors equally for recovery.

Recovery must be preceded by correct detection. However, some data-oriented attacks may still go undetected, even when mechanisms to defend different types of data-oriented attacks are combined. Data-oriented attacks can be designed in such a way that each type of mechanism fails to detect them. We show an example of such an attack in Section 3.1. This implies that extending the existing defense mechanisms, even through a combination of different types of mechanisms, may not work for proper recovery. So for recovery, we need to take a different approach that analyzes the complex and subtle effects caused by changing the various inputs.

To address these limitations, we propose a framework called SCVMON that detects various types of data-oriented attacks and helps RVs recover from them. SCVMON is hardware-independent and requires no additional hardware. Also, SCVMON is implemented on RV control programs and can be applied through a simple software update. The key insight behind SCVMON is that any change of inputs generating inappropriate control commands inevitably changes the values of certain variables affecting the safety of RVs in a control program. We call these variables safety-critical variables (SCVs). In other words, altering the inputs to crash or destabilize RVs will eventually change the value of SCVs, regardless of the attacker's intention or type of attack. The values of SCVs reflect all the complex and subtle effects caused by the changes in various inputs. Therefore, monitoring these SCVs can detect and recover from various types of data-oriented attacks and their combinations. For efficiency and proper recovery, we evaluate the recovery-suitability of SCVs, and based on this, reduce the set of SCVs into the set of monitored safety-critical variables (mSCVs). Monitoring all sensor measurements and hundreds of configuration parameters is inappropriate in RVs, which is a real-time system, so we only monitor mSCVs that can reflect all changes in these inputs. If an attack is detected on a particular mSCV, SCVMON performs a recovery by changing the value of that mSCV to a normal value. In addition, by implementing flexible switching between normal mode and recovery mode, SCVMON makes it possible for RVs to carry out their assigned mission when an attack is terminated or a false alarm occurs. In summary, SCVMON not only prevents RVs from causing physical damage as a result of various attacks but also provides mission continuity.

The main contributions of this study can be summarized as follows:

- To the best of our knowledge, SCVMON is the first architecture that helps RVs recover from various data-oriented attacks.
- We propose a systematic method for identifying safety-critical variables in RVs beyond an empirical analysis. In addition, we introduce the recovery-suitability concept and propose a systematic method to identify a monitoring set that can effectively detect and recover data-oriented attacks.
- We implement the SCVMON[1] framework on ArduPilot[4], which is a popular RV control program. The results demonstrate that SCVMON can rapidly detect and recover from various data-oriented attacks. SCVMON incurs approximately 5.1% runtime overhead and 0.3% storage overhead, so it is suitable for practical application in RVs.

2 BACKGROUND

2.1 RV control program

An RV control program is a cyber element of RVs that controls physical elements such as the motor to ensure the proper and safe operation of RVs. The program consists of a main loop and several tasks for performing specific functions, such as ground control station (GCS) communication and log writing. Each task is repeated at predefined intervals. The main loop operates based on a feedback loop (see Figure 1) to generate appropriate control commands.

An RV control program generates appropriate control commands through complex calculations of sensor measurements and configuration parameters, to perform assigned tasks. Sensors are used to measure the physical state of and external environmental factors surrounding RVs. Additionally, an RV control program utilizes a number of configuration parameters that are compatible with various RV types and models[1]. These configuration parameters are used in various locations within an RV control program and can affect the generation of control commands.

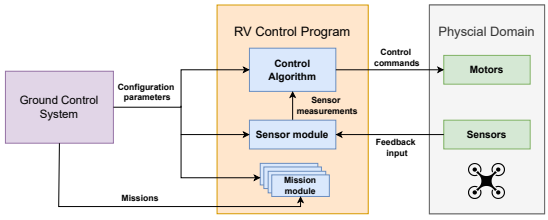


Figure 1: An RV control program

2.2 Previous Works

Choi *et al.* proposed a control-invariant approach[14] that extracts the invariant values from the RV and detects physical attacks based on changes in the corresponding invariant values. Quinonez *et al.* proposed SAVIOR[45], which defends against stealthy attacks[15, 51, 54]. SAVIOR is very similar to [14], save for two main differences: First, SAVIOR uses a nonlinear model to predict invariant values rather than a linear model, and second, it uses the CUSUM algorithm to detect attacks rather than the time window method. Ding *et al.* proposed Mini-Me[18], which is an online monitoring framework that detects data-oriented attacks against RVs by modeling program-level control state dynamics. Mini-Me utilizes the internal data flow information and control variable dependencies of RV controller functions to create a neural network-based approximate model to

[1]SCVMON is available at https://github.com/SCVMON/SCVMON

Table 1: Summary of previous mechanisms for RVs

No	Mechanism	Sensor attack	Parameter attack	No additional HW/SW	Recovery	Applied to real RV
1	CI[14]	✓	△	✓		✓
2	SAVIOR[45]	✓	△	✓		✓
3	Mini-Me[18]	✓	✓	✓		✓
4	VirtualDrone[55]		✓		✓	✓
5	BlueBox[20]	✓	✓		✓	✓
6	LA[57]	✓		✓	✓	
7	SSR[13]	✓		✓	✓	✓
8	PID-piper[16]	✓		✓	✓	✓
9	SCVMON	✓	✓	✓	✓	✓

△: Able to defend against certain types of attacks

detect various attacks. These approaches detect attacks as they occur, but unfortunately, they cannot recover from them.

Yoon *et al.* proposed VirtualDrone[55], which performs recovery through virtualization by sandboxing untrusted operations, including sensors, actuators, and communication channels. However, virtualization environment can incur a relatively large overhead, and some hardware may not support the configuration of such a virtualization environment. Additionally, VirtualDrone is mainly focused on defending against cyber attacks. Fei *et al.* proposed BlueBox[20], which is a framework that can detect and recover from cyber-physical attacks by combining software and hardware redundancy. BlueBox extracts the vehicle's control logic and implements the control and sensing logic independently in external hardware to detect and recover from attacks. The main limitation of these approaches is that they require additional hardware or software to perform the recovery, which can incur increased overhead and cost.

Zhang *et al.* proposed LA[57], which synthesizes a recovery controller that infers the current state when a sensor attack is performed and returns to the target state set before a safety deadline. However, this approach has not been implemented in any actual RV control programs. Choi *et al.* proposed SSR[13], which uses modeling to predict sensor values rather than control invariants. If these values are different from the measured values, SSR isolates the compromised sensors and utilizes the estimated sensor values to perform recovery. Dash *et al.* proposed PID-piper[16] as a method for recovering from sensor attacks. This method uses machine learning to predict the output value of the controller, and it detects and recovers attacks if the predicted value differs from the value generated by the actual controller. While these three approaches do not require additional hardware or software, they can only recover from limited types of attack. The characteristics of the proposed approaches and our SCVMON are summarized in Table 1.

3 MOTIVATION AND ATTACK MODELS

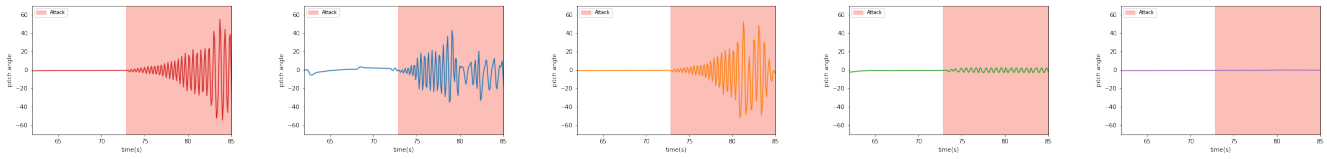
3.1 Motivation

Previous recovery mechanisms[13, 16, 25, 47, 57] focused on recovering only from sensor attacks, hence cannot recover from parameter attacks. There are studies [29, 32] to defend against parameter attacks by setting the normal range of each parameter. Therefore, attacks that significantly change sensor measurements or set parameter values out of the normal range can be detected and prevented through the previous mechanisms. At this time, an attacker can bypass the previous mechanisms and destabilize RVs by changing the parameter values within the normal range and slightly changing sensor measurements. This is because the possible effects incurred by simultaneous changes in two different types of inputs are not considered.

We show in an example that previous recovery mechanisms cannot adequately defend against this type of attack. In the example, an attacker slightly changes the gyroscope measurement which was originally close to zero into 0.1. At the same time, the attacker also slightly changes the configuration parameters *ATC_RAT_PIT_P* and *ATC_RAT_YAW_P* which correct the difference between the desired and actual rate on the pitch axis and yaw axis, respectively. QGroundControl[44], a widely used GCS program, limits the ranges of *ATC_RAT_PIT_P* and *ATC_RAT_YAW_P* to (0.08, 0.35) and (0.18, 0.6), so the attacker changes those values to 0.35 and 0.6, i.e., the maximum values within the normal range.

We implement SSR, PID-Piper, and CI-based recovery mechanisms to evaluate flight when the above attack is performed. The selection and implementation of previous mechanisms and their detailed evaluation results are described in Section 5. Figure 2 depicts the pitch angle over time when the above attack was performed. As shown in Figure 2a to Figure 2c, when this attack is performed, the RVs become unstable and eventually crash.

To account for these results, we additionally evaluate flight when only one type of data-oriented attack is performed. If only the gyroscope measurement is altered, the RV operates safely as shown in Figure 2d without any defense mechanism applied. Since these subtle changes in sensor measurements do not affect the flight of the RV, the existing mechanisms naturally determine them as measurement errors. Also, as shown in Figure 2e, if only the values of *ATC_RAT_PIT_P* and *ATC_RAT_YAW_P* are changed within the normal range, the RV's flight is not affected. In this case, the parameters are changed to values within the normal range, so the RV control programs and mechanisms do not block these changes and no alarm is set off. However, since control commands that control physical elements are generated through complex calculations between different sensor measurements and various configuration parameters, a simultaneous attack, should one occur, can destabilize RVs. For this reason, previous recovery mechanisms are unable to quickly detect a simultaneous attack and cannot determine where the attack was conducted, making it impossible to recover properly.



(a) CI (b) Pid-Piper (c) SSR (d) Gyroscope spoofing (e) Parameters change

Figure 2: Flight evaluation during a data-oriented attack

To properly recover from these attacks, changes in various inputs must be comprehensively considered.

3.2 Attack Model

In this paper, we consider various data-oriented attacks. Data-oriented attacks include (1) sensor attacks that change sensor measurements, (2) parameter attacks that change configuration parameters used in various RV control program modules, such as attitude control, sensors, mission, etc., and (3) combinations of these attacks. Attackers aim to cause physical damage or interfere with RV's missions through such attacks. Considering various existing attack cases[10, 40], these attack goals are the most common and reasonable. Some data-oriented attacks exploit vulnerabilities caused by improper implementation of the RV control program. However, we do not consider these attacks as they can be mitigated through existing mechanisms that are able to identify[11, 29, 32] and patch[30] vulnerable codes.

We assume attackers have expertise in RV and control programs. These attackers can easily identify the inputs of RV control programs through open RV control programs as well as official documentation, as described in Section 4.2.1. Also, the attackers can use various resources to maliciously change the identified inputs. Previous studies have already proposed various methods for changing sensor measurements and configuration parameter values. Attackers can change sensor measurement values by utilizing acoustic noise[25, 50] or sensor-spoofing tools[24, 38, 48]. In addition, [29, 32] introduced the method of manipulating RV configuration parameters, and these studies also argue that such attacks are realistic. Therefore, we start with the conservative assumption that attackers can utilize these methods to inflict change on inputs.

Also, similar to previous studies[13, 14, 16, 18, 45, 57], we do not consider control-flow attacks. As opposed to simply changing inputs, an attacker performing a control-flow attack must identify additional vulnerabilities. Additionally, these attacks, such as buffer overflow and return-oriented attacks, can be prevented through secure coding practices and memory protection mechanisms[28, 34, 41]. Therefore, we assume that an attacker cannot bypass or corrupt SCVMON. In conclusion, we focus on data-oriented attacks that generate inappropriate control commands, through which they can change various inputs of an RV control program.

4 SCVMON ARCHITECTURE

In this section, we detail the design of our framework. We present an overview of the SCVMON framework in Section 4.1 and then describe the details of SCVMON from Section 4.2 to Section 4.4.

4.1 SCVMON Framework Overview

A key concept of our approach is to identify and monitor SCVs suitable for recovery in RV control programs. However, there are two main challenges to this approach. First, it is difficult to identify variables that are related to safety and are suitable for recovery. RV control program developers or security experts may implicitly determine whether certain variables are safety-critical or not, but this method cannot guarantee accurate results. To overcome this challenge, we propose a systematic approach that performs a static analysis on RV control programs and uses an RV simulator to identify SCVs. We also introduce a recovery-suitability evaluation to further identify variables that are suitable for recovery among the identified SCVs.

Second, even if SCVs are identified using the proposed methods, it is impossible to monitor all SCVs due to real-time constraints. Similarly, the RV control program uses several sensor measurements and hundreds of configuration parameters as inputs, so monitoring all of these input changes can cause significant overhead[11]. Therefore, in order to effectively defend against various data-oriented attacks that change inputs, it is necessary to extract a small mSCVs set that can reflect all these changes. We identify an mSCV set by performing graph cuts using source-sink analysis results and the results of safety-criticality and recovery-suitability evaluations. Subsequently, monitoring only the identified mSCVs enables SCVMON to detect any attack that changes sensor measurements and configuration parameters without violating real-time constraints. Figure 3 presents an overview of the SCVMON framework, which consists of three main steps: (1) pre-processing, (2) identifying mSCVs, and (3) monitoring mSCVs.

4.2 Pre-processing

The pre-processing step first identifies the inputs and outputs of the RV control program. The step then converts the RV control program into an LLVM IR[37] bitcode to generate a data flow graph (DFG) of the RV control program. Finally, a source-sink analysis is performed based on the generated DFG and the identified inputs and outputs.

4.2.1 RV inputs and outputs. The inputs of the RV control program are collected from official manuals[1, 3] and various references[29, 32]. To identify the sensor-related variables in the RV control program, all physical sensors used by the target RV are first identified. Then, the variables storing the measurements of these sensors are compiled into a list. All configuration parameters are outlined in official documentation[1], which also provides detailed information about each parameter. Additionally, GCS program

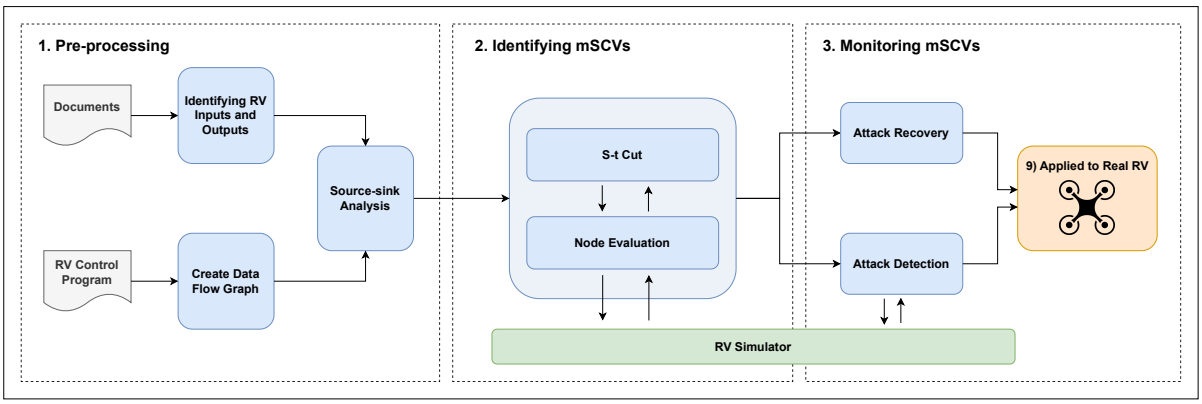


Figure 3: Overview of the SCVMON framework

QgroundControl[44] provides a complete set of configuration parameters that the target RV control program uses while connected to the RV. Configuration parameters that do not clearly affect control command generation are removed from the list to reduce input space. For example, *CAM_MIN_INTERVAL* and *LOG_FILE_BUFSIZE* are configuration parameters that set the minimum time between photos and the buffer size of log files, respectively. Both clearly do not affect the control commands, so they can be removed. We use the official description of configuration parameters[1]. The outputs of the RV control program are variables that store the final values transmitted to the motors. Through this process, a group of variables corresponding to the full list of inputs and outputs—and their locations in the source code—are identified. The corresponding input and output variable lists are described in Appendix B.

4.2.2 Data flow graph for an RV control program. Static analysis of the RV control program identifies relationships and interdependencies between variables in the program. Changes in the input values of the RV control program are propagated to various variables in the program[26, 31]. We perform a source-sink analysis with the generated DFG to identify the variables that can be affected by any input changes during the control command generation process. By monitoring these variables, SCVMON can detect simultaneous changes in various inputs, and solve the interdependency issue caused by input changes. For efficient monitoring, we reduce the size of the variables set to be monitored in Section 4.3.

The DFG of the RV control program is constructed using the static value flow analysis tool (SVF)[52]. However, due to some characteristics, including indirect value flows, the generated DFG has errors that need to be fixed. First, several intrinsic functions in the LLVM IR, such as `fabs`, do not generate proper edges in the DFG. In addition, proper edges are also missing in the vector-related LLVM IR instructions, like `extractelement`, `insertelement`[36]. To fix these errors, we identify corresponding intrinsic functions and instructions on the LLVM IR code and express these as regular expressions to describe where the edge should be added. In SVF[52], if another `load` or `store` instruction references the memory adjacent to the memory referenced by the previous `store` instruction, SVF generates an indirect edge between the two instructions. However, we do not consider buffer overflow attacks that overwrite the adjacent memory, so an edge should be generated only if an instruction refers to the exact same memory address. The `getelementptr` instruction is used to compute the memory address, so we generated an edge between `store` and `getelementptr`, which refers to the same memory address.

In order to perform further analysis with the generated DFG and be able to interpret the results, it is necessary to connect the variables in the C++ code of the RV control program to the corresponding LLVM IR variables. Most of the variables in the control program are instance variables or static variables. Therefore, we create a mapping table that connects the input and output variables as identified in Section 4.2.1 and various variables of the control program to `getelementptr` instructions. The generated mapping table supports the creation of appropriate inputs for additional analysis and enables us to interpret the results.

4.2.3 Source-sink analysis. This step extracts a subgraph of the DFG generated in Section 4.2.2. The goal of this source-sink analysis is to identify a candidate group of SCVs. SCV candidates are defined as follows.

- inp: Set of sources
- $outp$: Set of sinks
- $Con = \{ x \mid \forall a \in inp,$ there is a path from a to $x\}$
- $S = \{ x \mid \forall a \in outp,$ there is a path from x to $a\}$
- $SCV_c = Con \cap S$

Set *inp* is a set of sources in the DFG, i.e., a set of variables in the RV control program that an attacker can change. Set *outp* is a set of sinks, i.e., a set of variables related to control commands. The set of sources and sinks correspond to inputs and outputs of the RV control program identified in Section 4.2.1, respectively, and the lists for each are described in Appendix B. *Con* is the set of all nodes reachable from elements of *inp*, and *S* is the set of all nodes reachable to the elements of *outp*. This implies that *Con* is the set of variables in the RV control program that can be affected when an attacker changes the control program inputs. *S* is a set of variables that may be used to generate control commands. Thus, SCV_c, the intersection of *Con* and *S*, is an SCV candidate set that can be affected by attacks that generate inappropriate control commands.

Source-sink analysis performs a forward breadth-first search (BFS) and backward BFS to generate a subgraph of the DFG. This

subgraph consists of all nodes on the paths from *inp* to *outp* as well as the edges between the nodes on the paths. In some cases, even if there is an edge between two nodes, the corresponding value flow is connected only when a specific function is called. Therefore, function call contexts are stored and checked to obtain more accurate results by reducing false positives during the BFS process[46]. The algorithm for the source-sink analysis is outlined in detail in Appendix A.

4.3 Identifying mSCVs

Monitoring all variables in SCV_c is inappropriate due to real-time constraints. For efficient detection and recovery, we extract monitored SCVs (mSCVs) from SCV_c such that the set of mSCVs can still reflect all the changes in the inputs. By monitoring only mSCVs, we can reduce the overhead of detection and recovery mechanisms. To extract mSCVs, we perform a backward search on the subgraph generated in Section 4.2.3. Nodes encountered during this backward search are evaluated and classified according to two criteria: safety-criticality and recovery-suitability.

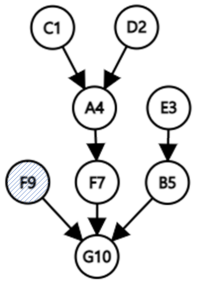
4.3.1 Nodes evaluation. We describe how to evaluate nodes based on two criteria: safety-criticality and recovery-suitability.

Safety-criticality We classify a node as safety-critical if its value change can generate inappropriate control commands. Since it is difficult to change the value of a variable in the LLVM IR code and simulate it, we identify the C++ variable and its location corresponding to that node. We implement a module that randomly changes the C++ variable value at that location and simulate it using RV simulator[2, 22] to identify whether that variable is safety-critical. At this time, the flight of the RV is evaluated by identifying negative impact occurrences, and we do this using the control instability detector methodology[32]. We observe the RV's roll, pitch, yaw, and altitude to monitor the negative effects. We classify a node as safety-critical if there is a difference in the values of roll, pitch, yaw (>10 °), or altitude (>1 meter) in comparison to normal flight.

In some cases, it may not be possible to determine the safety-criticality of a variable by simply changing a single variable at a specific location. We introduce such an example in Figure 4. We assume the situation of evaluating node F7 corresponding to the variable (F) in line 7. If the value of the variable z is greater than zero, the value of G10 does not change, no matter how much the value of F7 changes. To handle this case, the control flow must be adjusted so that the value moves from F7 to G10. The conditional branch of the LLVM IR that determines this control flow exists in the same code block with the LLVM IR instruction corresponding to the node, so it can be easily identified. After that, we extract the condition ($z > 0$) that determines the control flow and identifies the variable (z) used in this condition. If this variable (z) is an element of *Con*, the value may be modified by changing inputs. Therefore, in this case, the conditional statement of the C++ code corresponding to the LLVM IR conditional branch is partially modified so that the value change of F7 affects G10. Then we use the above module to change the value of the variable at that location to determine whether or not the variable is safety-critical. However, if the variable used in the condition cannot be changed by the inputs, the evaluation will perform without changing the conditional statement.

```
1  C = x;
2  D = y;
3  E = x + y;
4  A = min(C, D);
5  B = range(0, E, 1);
6  if(z > 0)
7      F = A;
8  else
9      F = 1;
10 G = F * B;
```

(a) Example code



(b) Corresponding DFG

Figure 4: Code and its corresponding data flow graph

C1, on the other hand, is used as an input for function *min*, which produces value A4. Figure 4b is a simplified DFG, so detailed nodes for function *min* are not expressed in it, but the actual *min* function consists of several conditional branches in the LLVM IR. In this situation, D is the variable used for the conditional statement. Since variable D is an element of *Con*, we change the C++ code so that the change in the value of C1 can flow to A4 without any limitations. Our algorithm, outlined in Section 4.3.2, searches the nodes in reverse, so after F7 is evaluated and classified, C1 is evaluated. Therefore, the conditions for data to flow from F7 to G10 are identified by the previous node evaluation. In conclusion, in the process of evaluating C1, it is only necessary to identify the conditions to flow to its child node (i.e., A4).

E3, on the other hand, branches based on 0 and 1. These static values cannot be changed by inputs, so E3 is evaluated without changing the conditional statement. As a result of the evaluation, if E3 is not an SCV, it means that changing the value of E3 does not change the child node (i.e., B5) to an improper value. In other words, even if an attacker changes various input values to abnormally change the value of E3, which is a non-SCV, these value changes on E3 have no abnormal or detrimental effects on B5. In conclusion, the edge from the non-SCV node to the SCV node can be considered as broken in our DFG.

Recovery-suitability The recovery-suitability evaluation process is similar to the safety-criticality evaluation process. First, a predicted value is generated through the average of the previous values of the variable at a specific location corresponding to the node. We replace variable values with predicted values using the same module utilized in the safety-criticality evaluation. At this time, the conditions for the corresponding variable value to flow to the final output, which is identified in the safety-criticality evaluation process, are equally utilized. Then we perform simulations and evaluate the RV's flight. If the RV continues to operate consistently after replacing the predicted value, we classify the node as "high recovery-suitability." If the RV can operate safely, but moves to a different position, or there are minor changes in the flight, then we classify the node as "moderate recovery-suitability." Finally, if the RV is unable to operate safely and there is a significant negative impact on the flight, we classify the node as "low recovery-suitability."

Variables generated by combining a number of variables, however, such as the actuator output, require more accurate value settings. Therefore, in this case, a reliable flight may not be possible

when actual values are replaced with predicted values. However, monitoring these variables can collectively identify changes in multiple inputs, so attacks can be detected by monitoring only a small number of mSCVs. Conversely, variables whose values are generated independently such as configuration parameters might be better suited for recovery. However, in this case, a large number of mSCVs must be monitored, resulting in significant overhead. Therefore, it is necessary to determine the correct balance between the size of the mSCV set and recovery-suitability.

4.3.2 S-t Cut. In this section, we describe how to extract the set of mSCVs using node evaluation results. We extract the set of mSCVs that satisfies the following four conditions: 1) All paths from sources to sinks must pass through at least one mSCV. 2) The mSCV must be safety-critical. 3) The recovery-suitability of the mSCV must not be low. 4) The smaller the size of the mSCV set, the better.

$$Monitor = \{M_1, M_2, \cdots M_n\} \tag{1}$$

$$\forall a \ \in inp, \forall\, b \in outp, \quad \forall i, \quad \exists\, c \in M_i \ : \ a \rightarrow c \rightarrow b \tag{2}$$

The first condition can be expressed as the formula above. That is, no matter what inputs change, any such changes must pass through an mSCV to change the control commands. Therefore, the problem of identifying a set of mSCVs is the same as dividing one graph into two groups, meaning that all source nodes would be located in one group and all sink nodes would be located in another group—thus becoming synonymous with the s-t cut problem. However, we do not cut the graph through the minimum number of edges, so we cannot use existing algorithms[19, 21]. Our algorithm extracts a set of mSCVs that utilizes the node evaluation results in order to satisfy the four conditions. The pseudocode for this algorithm is presented in Algorithm 1. Our algorithm also searches in reverse, starting from the sink nodes, to identify mSCVs. When a particular node pops out from the queue, that node is evaluated as described in Section 4.3.1. Any further search is determined by these evaluation results, and a set of mSCVs is finally extracted.

Algorithm 1: S-t cut

```
Function Main(outp):
    Input outp: Set of sinks
    Output mSCV: Set of mSCVs

    mSCV ← ∅;
    Q ← outp;
    visited ← outp;
    while Q ≠ ∅ do
        node ← Q.pop();
        if node.safety_critical = true then
            if node.recovery_suitability = low then
                edges_in ← incoming edges of node;
                foreach in ∈ edges_in do
                    src ← parent node of in;
                    if src ∉ visited then
                        Q.push(src);
                        visited.push(src);
                    end
                end
            end
            else if node.recovery_suitability = mid or high then
                mSCV.push(node);
            end
        end
    end
    return mSCV;
```

If a node is evaluated as non-safety-critical, no further backward search is performed from that node, and it is not selected as an mSCV. The non-safety-critical node does not need to be monitored because the RV can operate normally even if an attacker were to combine various inputs and change the value of the node. In this case, a conflict between condition 1) and 2) may occur. However, according to the discussion in Section 4.3.1, the edges between a non-safety-critical node and a safety-critical node could be considered to be broken. If a node is evaluated as safety-critical and its recovery-suitability is evaluated as low, the node is not selected as an mSCV, and an additional backward search is performed starting from that node. Conversely, if a node is categorized as safety-critical and its recovery-suitability is evaluated as moderate or high, the node is selected as an mSCV and no further search is performed.

As a result, we identify an set of mSCVs that has 27 safety-critical variables with moderate or high recovery-suitability. Detailed information on the 27 mSCVs is provided in Appendix D. However, the identified set of mSCVs is not the only set; it is merely the one that can monitor all changes in inputs that generate inappropriate control commands. To show the completeness of the identified mSCVs, we prove that the set of mSCVs extracted by our algorithm satisfies the first condition. The proof for this proposition is described in Appendix F.

4.4 Monitoring mSCVs

4.4.1 Attack detection rule. As previously discussed, any attack that generates inappropriate control commands inevitably changes the values of mSCVs. Also, in this case, the values of mSCVs change notably. Since the RV control program runs the main loop in short cycles, even if the value of a variable needs to change under normal circumstances, it will change slighty between each cycle. However, if an attacker changes the RV's inputs in step *n*, this effect will appear immediately in step *n+1*. Note that an attacker cannot directly change the values of mSCVs. That is, the attacker indirectly changes the values of mSCVs by changing the input values, and this makes it difficult for malicious changes to bypass our detection mechanism. SCVMON detects the attacks by identifying notable changes in 27 mSCVs.

Based on this observation, we determine two thresholds for each mSCV. One is the threshold for the difference in mSCV values of two successive cycles, and the other is the threshold for the accumulated difference in mSCV values over several successive cycles. We construct several normal flight scenarios to identify the lower bounds of these thresholds. We assign several different missions and simulate the RV's flight under different wind conditions. Also, we implement a log module in the RV control program to record the values of 27 mSCVs in every cycle. We identify the maximum difference and the top 99.7% (3σ rule) difference under this normal flight conditions.

We also simulate the RV's flight by directly changing each mSCV value to identify the minimum difference that makes the RV unstable. This minimum difference is used as the upper bound of thresholds. If this minimum difference is greater than the maximum difference under normal conditions, we set a threshold within the range of the two difference values. However, in normal flight conditions, the value may change significantly with a very small

probability, and at this time, the maximum difference in normal conditions may be greater than the minimum difference in attack conditions. In this case, the top 99.7% difference is used instead of the maximum difference to determine threshold. In other words, in normal flight conditions, each mSCV changes beyond the threshold only with a very low probability (<0.3%). Setting the threshold as described above can minimize the occurrence of false positives. Also, an attack that ultimately destabilizes RVs requires changes in values higher than the minimum difference, and setting thresholds lower than the minimum difference can minimize false negative occurrences. The threshold for the accumulated difference in mSCV values over several successive cycles is also determined in the same way. Some mSCVs are static variables whose values rarely change under normal conditions. In this case, we only use the threshold for the difference in mSCV values of two successive cycles and set it to zero to identify whether the value has changed. The thresholds of each variable are described in Appendix D.

Algorithm 2: Attack detection and recovery

```
Function Main(mSCV):
    Input mSCV: Set of mSCVs

    while True do
        act.diff_queue.push(|act.cur_val − act.prev_val|) ;  ▷ fixed-size
        if all scv.attack = False then
            act.attack ← False ;     ▷ All recovery modes are terminated
        end
        if |act.cur_val − act.prev_val| > threshold1 or
          mean(act.diff_queue) > threshold2 then
            act.attack ← True ;      ▷ Anomaly detection on the actuator
        end
        act.prev_val ← act.cur_val;

        foreach scv ∈ mSCV do
            if scv.type = "dynamic" then
                scv.val_queue.push(scv.cur_val);
                scv.recover_val ← mean(scv.val_queue);
            end
            else if scv.type = "static" and scv.attack = False then
                scv.recover_val ← scv.cur_val;
            end
            if scv.attack = False then
                scv.diff_queue.push(|scv.cur_val − scv.prev_val|);
                if |scv.cur_val − scv.prev_val| > threshold1 or
                  mean(scv.diff_queue) > threshold2 then
                    scv.attack ← True ;    ▷ Anomaly detection on SCVs
                    scv.count ← 400 ;      ▷ Switch to caution state
                end
            end
            else
                if act.attack = False and scv.type = "dynamic" then
                    scv.count ← scv.count − 1;
                    if scv.count = 0 then
                        scv.attack ← False ; ▷ Return to normal state
                    end
                end
                else if act.attack = True then
                    if scv.recover_val ≠ None and
                      |scv.recover_val − scv.cur_val| < threshold then
                        scv.recover_val ← None;
                        scv.attack ← False ;▷ Terminate recovery mode
                    end
                    else
                        scv.cur_val ← scv.recover_val ;    ▷ Recovery
                          mode
                    end
                end
            end
            scv.prev_val ← scv.cur_val;
        end
    end
```

For attack detection methods using the modeling technique, the accuracy of the model may decrease over time. However, as our methodology simply detects attacks based on differences in mSCVs values, we do not need to account for this conceptual drift. Also, natural phenomena such as wind have continuity, so in general, wind speed does not change discontinuously. That is, in most cases, environmental factors do not change the mSCV values dramatically over a very short period (2.5 ms) and therefore do not significantly affect our detection mechanism. In addition, even if false positives occur due to external environmental factors, the influence of these false positives can be minimized through the flexible mode-switching described in Section 4.4.2.

4.4.2 Attack detection and recovery mechanism. We present a simple pseudocode for our attack detection and recovery mechanism in Algorithm 2. SCVMON monitors 27 mSCVs and variables related to actuator outputs, and pushes the values of the current mSCVs in queues that store the values of the previous 400 cycles. SCVMON switches the "caution" state if the specific mSCV changes abnormally. In the "caution" state, if the *_actuator* variable, which determines actuator outputs, is also abnormally changed, SCVMON determines there to be an attack. SCVMON switches the "caution" state back to the "normal" state if the value of the *_actuator* is normal for 400 cycles after an abnormal change is detected in an mSCV, which has a dynamic value. In contrast, if an abnormal change is detected in an mSCV, which has a static value, SCVMON continues to maintain the "caution" state. In conclusion, SCVMON detects attacks through abnormal changes in *_actuator* as well as mSCVs. This approach is based on previous insight that, to destabilize an RV, variables affecting physical elements must be changed.

When an attack is detected, SCVMON recovers only the abnormal mSCVs. For dynamic mSCVs, recovery mode replaces the value of mSCVs with a prediction generated using the average of values from 80 to 100 cycles prior to attack detection. For static mSCVs, recovery mode reverts to the initial value before the change. To identify mSCVs, we select only variables with high or moderate recovery-suitability. In this case, as we mentioned in Section 4.3.1, the RVs can operate normally even if the value is replaced by a naive prediction that contains an error in the actual value. This indicates that our method of recovering by simply generating predictions can provide adequate recovery. SCVMON exits recovery mode when the prediction approaches the actual value. In other words, by providing flexible mode-switching, SCVMON can provide mission continuity from false positives or transient attacks. In addition, SCVMON can adequately detect both simultaneous attacks and attacks that consist of multiple steps[11](discussed in Appendix C). Furthermore, SCVMON continuously monitors mSCVs during recovery mode, so even if attacks change the value of other mSCVs during recovery mode, SCVMON can also detect and recover from such attacks. In Section 5.2, we evaluate the effectiveness of SCVMON.

5 EVALUATION

In this section, we describe the implementation and evaluation of SCVMON. We evaluate SCVMON from two main perspectives: (1) how effectively SCVMON can detect and recover from various attacks and (2) how much overhead SCVMON incurs.

5.1 Implementation and Experimental Setup

We use the Copter 4.0.8 version of ArduPilot[4], a popular RV control program, as our target RV control program. We select the most common and widely used quadcopter as the target vehicle and evaluate 3DR IRIS+ quadcopter, a quadcopter model supported by ArduPilot. However, our approach for identifying and monitoring SCVs in RV control programs is not limited to specific RV types or models. All experiments are conducted on a desktop PC with an 8-core 3.6 GHz AMD Ryzen 7 CPU and 32 GB of RAM.

Source-sink analysis[2] We implemented the DFG generation and source-sink analysis with SVF[52] version 2.4. To compile ArduPilot and SVF, we use LLVM version 13.0. We apply the additional rules as described in Section 4.2.2 to construct a DFG. After constructing the DFG using the default libraries provided by SVF, we implement our source-sink analysis algorithm in C++. For additional tasks, we use Python 3.8. Finally, we perform a source-sink analysis based on Section 4.2.3 and extract a subgraph.

SCVMON We write approximately 500 new lines of code and modify 40 existing lines of code for ArduPilot in C++ to implement the attack detection and recovery mechanism. Additionally, we create a Python code to identify thresholds for attack detection.

Previous mechanisms In order to compare SCVMON with previous mechanisms, we implement CI, PID-piper, and SSR. VirtualDrone and BlueBox require additional hardware/software, MiniMe's code and model are not open to the public, LA is not applied to the actual RV control program, and SAVIOR is implemented in another control program, PX4[43]. For these reasons, these mechanisms are excluded from our comparison. However, since CI does not consider recovery, we implement CI by adding a recovery module for proper comparison. This module performs recovery by replacing the predicted value for the control invariant, which is an attack detection factor. Since the SSR code is not open to the public, we implement it as closely as possible to the original research paper.

Simulation configuration We simulate an RV and its environment using the APM simulator[2] and Gazebo[22]. We use MAVLink[39] as the communication protocol between the RV and the GCS and QGroundControl[44] as the GCS program.

Attack module To carry out data-oriented attacks, malicious actors need to utilize a variety of special-purpose hardware and resources. However, it is difficult to use all these resources to generate actual attacks to evaluate SCVMON, so we simulate attacks in a similar way as previous studies [13, 14, 16, 45]. We evaluate SCVMON and previous mechanisms by implementing an attack module on the control program that generates the same effect as the actual attack.

5.2 Effectiveness

We launch several attacks to verify the effectiveness of SCVMON against data-oriented attacks. These attacks include stealthy attacks that make subtle changes to inputs as well as attacks that make large changes to inputs. We measure detection time, crash time, and safe flight time for CI, PID-piper, SSR, and SCVMON. As shown in Table 2, CI cannot properly recover from data-oriented attacks because it does not specify a proper recovery location. PID-piper detects attacks by inferring actuator values and detects attacks in most cases, but the actuator prediction in PID-piper is generated without considering various inputs. As such, it fails to generate accurate prediction and establish an accurate recovery point, resulting in a crash. SSR can properly detect and recover from multi-sensor spoofing attacks and allow safe flight for a specific period, but still, it fails to detect and recover data-oriented attacks that simultaneously change sensor measurements and configuration parameters. In contrast to other mechanisms, SCVMON can quickly detect and recover from various data-oriented attacks. In all cases, the RV can operate safely for more than three minutes, and in some cases, it can continue to perform its assigned mission even after an attack is launched.

Next, we evaluate SCVMON with a wider variety of attacks. There are numerous attacks that change the various RV inputs, but it is impossible to test all of these attacks. Therefore, we construct 20 attacks that adequately represent a range of attack types. Data-oriented attacks change the values of sensor measurements and configuration parameters. Through static analysis, we identify configuration parameters that can destabilize RVs and match these parameters with related sensors to generate attack types. Attacks from No. 1 to No. 10 change individual sensors and correlated configuration parameters. Attacks from No. 11 and No. 12 change configuration parameters related to motors, and attacks from No. 13 to No. 20 change various sensors and configuration parameters simultaneously. Data-oriented attacks inevitably change the values of mSCVs, and these 20 attack cases include various situations in

Table 2: Comparison to previous mechanisms

Attack Target	Mechanism	Detection time	Safe flight time	Completion of mission*
Gyro	CI	<0.01s	>3m**	✓
	PID-Piper	<0.01s	>3m	✓
	SSR	<0.01s	>3m	✓
	SCVMON	<0.01s	>3m	✓
Gyro(S), Baro(S)	CI		7s	
	PID-Piper	0.5s	<0.1s	
	SSR	0.4s	1m 20s	
	SCVMON	<0.01s	>3m	✓
Gyro(S), Param(N)	CI		<0.1s	
	PID-Piper	0.25s	<0.1s	
	SSR		<0.1s	
	SCVMON	<0.01s	>3m	✓
Baro(S), Param	CI		7s	
	PID-Piper	0.7s	9s	
	SSR		7s	
	SCVMON	<0.01s	>3m	✓
Gryo, Baro, Accel, GPS	CI	<0.01s	<0.1s	
	PID-Piper	<0.01s	<0.1s	
	SSR	<0.01s	40s	
	SCVMON	<0.01s	>3m	△
Gyro(S), GPS(S) Baro(S), Accel(S), Param(N)	CI		<0.1s	
	PID-Piper	<0.01s	<0.1s	
	SSR		<0.1s	
	SCVMON	<0.01s	>3m	△

S: Stealthy attack, N: Change in parameters to a value within the normal range
✓: Able to perform assigned missions, △: Able to perform part of assigned missions
* Waypoint mission ** Since the simulation cannot proceed indefinitely, if safe flight is possible for more than 3 minutes after the attack occurs, we mark it as >3m.
Details of the attack are described in Appendix E.

[2]Source-sink analysis code is available at https://github.com/SCVMON/SCVMON-SVF

Table 3: Attack cases used to evaluate SCVMON

No	Attack	Detection time	Safe flight time	Completion of waypoint mission		Completion of various mission	
				Progress	Additional	Progress	Additional
1	Gyroscope spoofing attack	<0.01s	>3min	✓	✓	✓	✓
2	Gyroscope spoofing attack + parameter attack(1)*	<0.01s	>3min	✓	✓	✓	✓
3	Gyroscope spoofing attack + parameter attack(2)	<0.01s	>3min	✓	✓	✓	✓
4	Gyroscope spoofing attack + parameter attack(9)	<0.01s	>3min	✓	✓	✓	✓
5	Barometer spoofing attack	<0.01s	>3min	✓	△	△	△
6	Barometer spoofing attack + parameter attack(1)	<0.01s	>3min	✓	△	△	△
7	Accelerometer spoofing attack	0.01s	>3min	✓	✓	△	△
8	Accelerometer spoofing attack + parameter attack(1)	0.01s	>3min	✓	✓	△	△
9	GPS spoofing attack	0.02s	>3min	△		△	
10	GPS spoofing attack + parameter attack(7)	0.02s	>3min	△		△	
11	Motor related parameter attack(2)	<0.01s	>3min	✓	✓	✓	✓
12	Motor related parameter attack(4)	<0.01s	>3min	✓	✓	✓	✓
13	Multi-sensor spoofing attack(2)	<0.01s	>3min	✓	✓	△	△
14	Multi-sensor spoofing attack(2)	<0.01s	>3min	✓	△	△	△
15	Multi-sensor spoofing attack(3)	<0.01s	>3min	✓	△	△	△
16	Multi-sensor spoofing attack(4)	<0.01s	>3min	△		△	
17	Multi-sensor spoofing attack(2) + parameter attack(12)	<0.01s	>3min	✓	✓	△	△
18	Multi-sensor spoofing attack(3) + parameter attack(13)	<0.01s	>3min	✓	△	△	△
19	Multi-sensor spoofing attack(4) + parameter attack(19)	<0.01s	>3min	△		△	
20	Multi-sensor spoofing attack(4) + parameter attack(29)	<0.01s	>3min	△		△	

*Number of parameters or sensor measurements, ✓: Able to perform missions, △: Able to perform part of missions

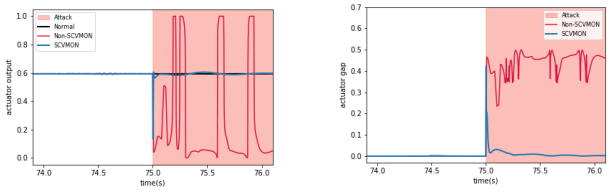
which each mSCV or a combination of mSCVs is changed. Note that attack No. 20 is a worst-case scenario, in which most mSCVs that affect control commands are simultaneously changed. We use the missions provided by ArduPilot's AutoTest suite (e.g., loiter, delay, spline) and our own simple missions (e.g., waypoint) to verify the effectiveness of SCVMON. The results of these attacks are listed in Table 3, and the details of the attacks are described in Appendix E.

Regardless of the missions, SCVMON can quickly detect all attacks in less than 0.02 seconds, and the RV operates safely for more than three minutes after an attack is detected. For simple waypoint missions, the RV can complete the mission in progress during most attacks, and in some cases, it can even perform additional missions after an attack is detected. However, SCVMON cannot support all the various missions of an RV. Nonetheless, even if a mission fails, our recovery mechanisms still work to realize SCVMON's main goal: continuous and safe operation of RVs. In conclusion, by monitoring only 27 mSCVs, SCVMON can quickly detect and recover from attacks without considering which precise inputs and corresponding values have changed.

Failure case analysis As shown in Table 3, SCVMON fails to facilitate the proper performance of assigned missions and/or additional missions in some cases. These failures occur because the recovery mode attempts to operate on variables with moderate recovery-suitability. Recall that in the process of identifying mSCVs, variables with moderate recovery-suitability may produce some undesirable effects on the RV flight (i.e., make the RV move to a different position or prompt other minor changes) when replaced with predictions. Therefore, if an attack changes one or more mSCVs and some of them are categorized as having moderate recovery-suitability, mission operation may become partially hindered. However, even in this situation, SCVMON can support safe flight for a certain period.

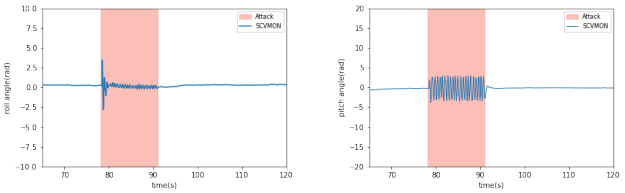
Actuator output In Figure 5a, we compare actuator outputs in normal mode with those in recovery mode during an attack. An attack that changes four sensor measurements and multiple configuration parameters (No. 20 in Table 3) starts at 75 seconds, which is denoted in pink. The black horizontal line indicates normal actuator values generated when there is no attack. During the attack period, the values generated by the RV without SCVMON and those generated by the RV in recovery mode are shown in red and blue, respectively. In the RV without SCVMON, the red line fluctuates greatly, and the RV ultimately crashes. However, in the RV with SCVMON, there is a different result. Just after the attack starts, the blue line fluctuates for a very short period and quickly converges with the black line. This indicates that the RV with SCVMON quickly detects the attack and enters recovery mode, during which prediction values are combined with other variables to immediately generate adequate actuator outputs. Figure 5b shows the output gaps of the actuator between the two modes. While the red line stays in high gap values, the gap values of the blue line are reduced to zero shortly after the attack and remain stable thereafter. As a result, we conclude that SCVMON can promptly generate appropriate values to control the physical elements of the RV through appropriate predictions.

Mode-switching If false positives occur, the RV enters recovery mode, which may partially hinder successful completion of the mission. However, SCVMON exits recovery mode if it determines that a false positive has occurred or an attack has ceased. Figure 6 shows the changes in roll angle and pitch angle when a gyroscope attack occurs. The attack period is denoted in pink, and it starts



(a) Actuator output **(b) Actuator output gap**

Figure 5: Comparison of actuator outputs



(a) Roll angle **(b) Pitch angle**

Figure 6: Mode-switching

at 78 seconds and ends at 91 seconds. SCVMON detects the end of the attack after 91 seconds and returns to normal mode. As a result, the RV can operate normally even though recovery mode has been initiated. SCVMON supports this flexible switching, allowing more stringent attack detection rules. Strict attack detection rules enable fast attack detection, which can minimize the impact of an attack. In conclusion, SCVMON can provide more reliable recovery while minimizing the impact of attacks and recovering quickly.

External environment Under various circumstances, SCVMON's recovery mode supports the safe operation of RVs, but it cannot be executed permanently. It means recovery mode cannot perfectly cope with all situations. In other words, recovery mode is a temporary solution to avoid attacks or assist missions, and cannot replace the existing control command generation process. In particular, changes in the external environment—such as wind and temperature—can hinder proper recovery. Therefore, we evaluate the effect of external environment on recovery. We select seven attacks in Table 3 that can represent other attacks and evaluate these by repeating simulation ten times each. For example, attacks from No. 1 to No. 4 show the same results in Table 3, and attack No. 4 is a more severe attack than others. As such, we select attack No. 4 as a representative case. We assign simple waypoint missions that are performed most frequently in RVs.

To evaluate wind effect on SCVMON, we evaluate the flight of RVs during recovery mode under two conditions: weak wind (1 to 3m/s) and strong wind (5 to 7m/s), where the direction and speed of wind change dynamically. We define it as a strong wind condition if an RV is unable to perform its mission (e.g., move to the desired position, hold position) in the absence of an attack. We evaluate safe flight in two aspects: attitude control and altitude control. If attitude control fails, RVs crash immediately, but if the altitude changes slightly, RVs may continue to operate safely. However, if the altitude changes significantly, a crash may occur, so if the actual altitude and the target altitude differ by more than 10 meters, we determine that the altitude control has failed and measure the time to reach that state.

Table 4: Wind effect

Attack (Table 3)	Wind (1~3m/s)		Wind (5~7m/s)	
	Attitude	Altitude	Attitude	Altitude
No. 4	✓	✓	✓	✓
No. 6	✓	✓	✓	37.2s (25s)*
No. 8	✓	✓	✓	43.8s (27s)
No. 10	✓	✓	✓	✓
No. 12	✓	✓	✓	✓
No. 18	✓	✓	✓	36s (24s)
No. 20	✓	✓	✓	7.2s (6s)

*Average failure time (Minimum failure time)

As can be seen in Table 4, in the event of weak wind, proper attitude control and altitude control are possible even though the recovery mode is executed. However, in the event of strong wind, attitude control is possible, but altitude control fails in certain attack cases. In this case, countermeasures such as user intervention or a parachute should be used. Unlike RVs crashing immediately in the absence of SCVMON, our recovery mechanism can provide sufficient time (6 seconds under harsh conditions) to carry out these countermeasures. In the case of mission performance, an RV cannot perform its mission in strong winds, even if there are no attacks; in weak winds, the results are the same as in Table 3.

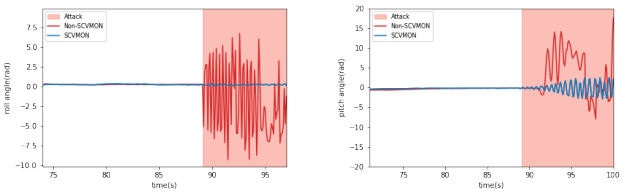
To evaluate temperature effect on SCVMON, we evaluate the flight of the RV by changing the temperature from 0 °C to 50 °C, but these changes do not affect the flight of the RV. In conclusion, SCVMON supports the continuation of safe operation in various attack situations except for certain harsh conditions and can provide sufficient time to perform secondary countermeasures even in situations in which safe operation is restricted.

5.3 Case Studies

Case study 1: Gyroscope spoofing attack with a multiple parameter attack (No. 3 in Table 3). Typically, gyroscope measurements are close to zero in normal flight conditions. This attack subtly changes the x-axis and y-axis gyroscope measurement to 0.1. At the same time, the attacker changes the values of *ATC_RAT_RLL_P* and *ATC_RAT_PIT_P* to 0.35, which are the maximum values in the normal range. This attack is similar to a stealthy attack. Figure 7 shows the roll and pitch angles over time when these attacks occur. The attack period is denoted in pink. Without SCVMON, the RV may fail attitude control and crash, as shown by the red line in Figure 7. The RV control program overcompensates to respond to the maliciously modulated gyroscope sensor value. This error is amplified by changed values of *ATC_RAT_RLL_P* and *ATC_RAT_PIT_P*, and this causes the roll and pitch angles of the RV to change significantly after an attack, leading to a crash.

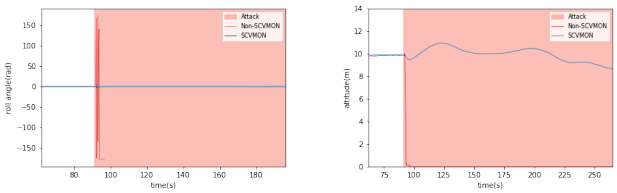
With SCVMON, however, the RV can quickly detect and recover from such attacks. Four mSCV values change when an attack is launched, and as the RV becomes unstable, the actuator value changes shortly thereafter. SCVMON detects an attack and begins recovery by replacing the four mSCVs with predicted values. The blue lines in Figure 7 show the roll and pitch angles when an attack occurs and when recovery mode is active. The roll angle does not change much, and the RV is only slightly shaken due to the pitch angle change. Generally speaking, a normal flight is possible. In conclusion, the RV can operate safely even when it is under attack.

Case study 2: Multi-sensor spoofing attack (No. 13 in Table 3). We consider an RV that performs a continuous mission of moving to three specific points and then returning home. Suppose that an attack occurs while the RV is moving to the first point. In this scenario, the attacker simultaneously changes the measurements of the accelerometer and gyroscope sensors to very large values. The z-axis measurements of the accelerometer represent the vertical



(a) Roll angle (b) Pitch angle

Figure 7: Attack detection and recovery during gyroscope spoofing with multiple parameter changes

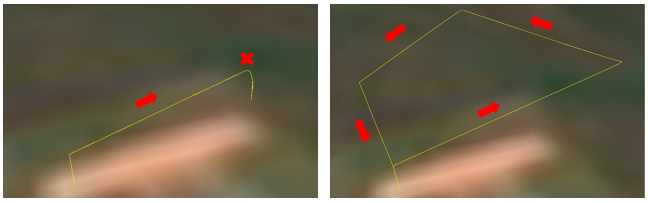


(a) Roll angle (b) Altitude

Figure 9: Attack detection and recovery under multi-sensor spoofing with multiple parameter changes

acceleration of the RV, which affects the thrust, and the gyroscope measurements, which are related to attitude control.

Figure 8 shows the trajectory of the RV when such an attack occurs. When SCVMON is not applied, the RV cannot generate adequate thrust and fails to control the attitude, resulting in a crash, as shown in Figure 8a. On the other hand, if SCVMON is applied, the mechanism detects abnormal changes in five mSCVs and recovers. Despite the malicious changes in these two sensor values, SCVMON minimizes the impact of the attacks by generating appropriate mSCV values. As a result, the RV can perform existing and additional missions, as shown in Figure 8b.



(a) An RV without SCVMON (b) An RV with SCVMON

Figure 8: Trajectory of an RV when a multi-sensor spoofing attack is launched

Case study 3: Multi-sensor spoofing attack with a multiple parameter attack (No. 20 in Table 3). An attacker simultaneously changes the accelerometer, gyroscope, barometer, and GPS sensor measurements, as well as 29 configuration parameters related to motor output and PID control. Multiple configuration parameters and sensor measurements interact with each other to generate malicious control commands.

Figure 9 shows the roll angle and altitude changes in the RV over time. If SCVMON is not applied, the roll angle of the RV changes significantly, and the altitude of the RV decreases rapidly and eventually crashes, as shown by the red line in Figure 9. If SCVMON is applied, malicious changes are detected in 23 mSCVs, and the corresponding variables are recovered. This recovery allows the RV to continue operating safely with proper attitude control, as shown by the blue line in Figure 9a.

The RV can also partially perform a previously assigned task of maneuvering to a particular point, but as shown in Figure 9b, the execution of recovery mode creates an altitude difference. Additionally, no other missions can be added after an attack is launched. This is because seven of the 23 variables in which recovery is performed have moderate recovery-suitability. However, unlike the result of a crash, even if an attack occurs with SCVMON, the RV can recover immediately and operate safely for more than two minutes, as shown in Figure 9b. Therefore, our recovery mode provides sufficient time for secondary countermeasures—such as user intervention or parachuting—even if an attack changes a large number of inputs simultaneously.

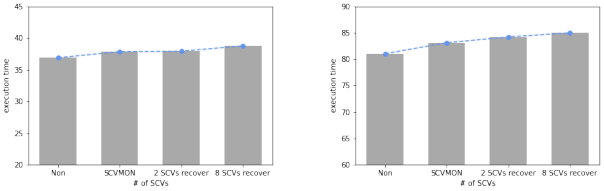
5.4 Performance Evaluation

We evaluate the performance of SCVMON in terms of storage and runtime overheads. To determine the storage overhead, we implement SCVMON and compare it to the existing firmware size. For the runtime overhead, we measure the execution times of tasks based on logs.

Storage overhead The significant increase in firmware size may be inappropriate due to resource limitations in RVs[23]. Approximately 500 lines of SCVMON code are added to the RV control program. While the size of the original firmware is 2,963 KB, and the size with SCVMON increases slightly to 2,972 KB. This means that SCVMON only expands the firmware size by 0.3%, so it is reasonable in terms of storage overhead.

Runtime overhead The RV performs various tasks, including the main loop. Each task is executed in a different cycle, and ArduPilot's main loop runs 400 times per second. We measure the average execution time of each task in the following three cases.

1. SCVMON is not applied.
2. SCVMON is applied, but recovery mode is not executed since no attack is detected.
3. SCVMON is applied, and recovery mode is executed.

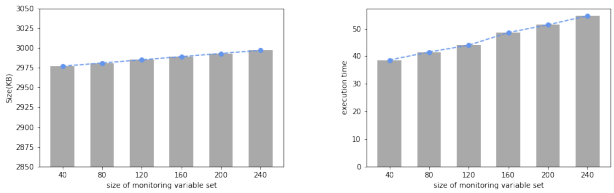


(a) Average runtime (b) Top 10% runtime

Figure 10: Runtime overhead

We also calculate the average of the top 10% longest execution times for each case. Note that there is no significant difference in execution time for tasks other than the main loop. The reason is that various variables used in other tasks generate the mSCVs, but these are located only in the main loop because SCVMON performs monitoring at the point where control commands are generated. Hence, we evaluate the runtime overhead based on the execution time of the main loop. Figure 10a presents the average execution time of the main loop in each scenario. The runtime overhead of Case 2 is only 2.7%. The runtime overhead of Case 3 increases depending on the number of mSCVs performing recovery. The overhead increases by 5.1% when it recovers eight mSCVs. Regarding the average of the top 10% longest execution times, there are overheads of 2.6%, 4.0%, and 4.9% for the three cases, respectively, as shown in Figure 10b. These runtime overhead results demonstrate that our approach is practical and efficient.

Scalability test In our methodology, we can extract a larger mSCV set by assigning a fine-grained recovery-suitability attribute value outlined in Section 4.3.1. Note that the mSCV set is not unique. Hence, we also measure overhead for various mSCV set sizes. To do this, we randomly select variables of the RV control program to make different mSCV sets. As shown in Figure 11, the storage overhead increases slightly, and the runtime overhead increases linearly.



(a) Storage overhead **(b) Runtime overhead**

Figure 11: Overhead with an increasing *mSCV set* size

In summary, our evaluations demonstrate that monitoring 27 mSCVs can help the RV adequately detect and recover from 20 types of data-oriented attacks. However, it may be necessary to monitor a larger number of mSCVs to perform a more accurate recovery. In this case, it is possible to extract and apply an appropriately sized mSCV set by utilizing the proposed methodology and considering the hardware performance of the target RV.

6 DISCUSSION

Bypass SCVMON SCVMON detects attacks based on deviations in mSCVs values. Thus, theoretically, our detection mechanism can be bypassed by constantly changing the value of the mSCVs very finely every cycle. However, bypassing our mechanism is quite difficult for two reasons. First, mSCVs are not inputs of the RV control program. That is, an attacker cannot directly change the values of mSCVs, which requires identifying and changing the inputs that affect those mSCVs. A single mSCV value is generated through complex calculations of various inputs and multiple variables on RV control programs. Therefore, identifying the combination of input changes that causes an mSCV value to change slightly each cycle (2.5 ms) can be quite challenging. Some mSCVs store the values of configuration parameters that are inputs of an RV control program. However, these are all mSCVs with static values, so the threshold is set to zero. Second, it may be difficult to make small changes in sensor measurement values in very short periods. To bypass our mechanism, an attacker would have to continuously perform microscopic input changes during each cycle. If the input value changes notably even once during the process, SCVMON can detect the attack. However, it can be quite difficult to implement these attacks with existing methods, such as using acoustic noise. In addition, in order to circumvent this, cyber attacks using compromised cyber components can be performed, but such attacks can be mitigated through various existing security mechanisms[28, 34, 41]. In conclusion, attackers can theoretically bypass SCVMON, but there are significant challenges in performing these attacks.

Manual work The process of identifying the inputs and outputs of the RV control program requires some manual work. In addition, a number of manual tasks are required in the process of extracting the mSCV set from the subgraph generated through source-sink analysis. Finally, the process of identifying the thresholds for mSCVs also requires some manual work, but some can be automated. For example, a program that automatically extracts the threshold of mSCVs through the automation of RV simulation can be implemented in the future. Also, please note that this work is only required during the first attempt to apply SCVMON to RVs. SCVMON does not require additional work after the initial application because it does not have to account for concept drift.

Complex control programs We discuss the applicability of SCVMON's methodology to more complex control programs. A main limitation of applying our method to more complex systems is the increased difficulty of static analysis for SCV identification. If the control program uses more inputs as well as code size increases, node explosion may occur during DFG generation and source-sink analysis. To solve this problem, it may be possible to perform static analysis in smaller code snippet units and combine these results; we leave the details of designing such a method for future work. In addition, the normal state can be defined differently depending on the nature of the system, which may require changes in detection and recovery mechanisms. To this end, various methods including machine learning can be utilized. Finally, although complex systems require more manual work, this can be mitigated by automation, as mentioned above.

7 CONCLUSION

In this paper, we propose SCVMON, a novel method to detect attacks and help RVs recover from various attacks. We achieve this by monitoring SCVs based on systematic proposals to identify SCVs suitable for recovery in RVs. The evaluation of SCVMON in actual RV control programs demonstrates that it can quickly detect and recover from various data-oriented attacks.

ACKNOWLEDGMENTS

This work was supported by the National Research Foundation of Korea (NRF) Grant funded by the Korean Government Ministry of Science and ICT (MSIT) under Grant NRF-2021R1A2C2014428.

REFERENCES

[1] Ardupilot. 2021. ArduPilot Complete parameter list. https://ardupilot.org/copter/docs/parameters.html

[2] ArduPilot. 2021. ArduPilot SITL simulator. https://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html

[3] ArduPilot. 2021. Autopilot Inputs and Outputs. https://ardupilot.org/copter/docs/common-flight-controller-io.html

[4] ArduPilot. 2022. ArduPilot. https://ardupilot.org/

[5] ARM. 2022. Trustzone for Cortex-M. https://www.arm.com/technologies/trustzone-for-cortex-m

[6] BBC. 2016. Drone hits british Airways plane approaching Heathrow Airport. https://https://www.bbc.com/news/uk-36067591

[7] BBC. 2016. US opens investivation into Tesla afeter fatal crash. https://www.bbc.com/news/technology-36680043

[8] BBC. 2017. Drone collides with commercial aeroplane in Canada. https://www.bbc.com/news/technology-41635518

[9] BBC. 2018. Tesla in fatal California crash was on Autopilot. https://www.bbc.com/news/world-us-canada-43604440

[10] Katharina Ley Best, Jon Schmid, Shane Tierney, Jalal Awan, Nahom M. Beyene, Maynard A. Holliday, Raza Khan, and Karen Lee. 2020. *How to Analyze the Cyber Threat from Drones: Background, Analysis Frameworks, and Analysis Tools.* RAND Corporation, Santa Monica, CA.

[11] Hongjun Choi, Zhiyuan Cheng, and Xiangyu Zhang. 2022. RVPLAYER: Robotic Vehicle Forensics by Replay with What-if Reasoning. *Proceedings of the Network and Distributed System Security Symposium.*

[12] Hongjun Choi, Sayali Kate, Yousra Aafer, Xiangyu Zhang, and Dongyan Xu. 2020. Cyber-Physical Inconsistency Vulnerability Identification for Safety Checks in Robotic Vehicles. In *Proceedings of the ACM Conference on Computer and Communications Security.* 263–278.

[13] Hongjun Choi, Sayali Kate, Yousra Aafer, Xiangyu Zhang, and Dongyan Xu. 2020. Software-based realtime recovery from sensor attacks on robotic vehicles. In *RAID 2020 Proceedings - 23rd International Symposium on Research in Attacks, Intrusions and Defenses.* 349–364.

[14] Hongjun Choi, Wen Chuan Lee, Yousra Aafer, Fan Fei, Zhan Tu, Xiangyu Zhang, Dongyan Xu, and Xinyan Deng. 2018. Detecting attacks against robotic vehicles: A control invariant approach. In *Proceedings of the ACM Conference on Computer and Communications Security.* 801–816.

[15] Pritam Dash, Mehdi Karimibiuki, and Karthik Pattabiraman. 2019. Out of control: stealthy attacks against robotic vehicles protected by control-based techniques. In *Proceedings of the 35th Annual Computer Security Applications Conference.* 660–672.

[16] Pritam Dash, Guanpeng Li, Zitao Chen, Mehdi Karimibiuki, and Karthik Pattabiraman. 2021. PID-Piper: Recovering Robotic Vehicles from Physical Attacks. In *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN).* 26–38.

[17] Drew Davidson, Hao Wu, Rob Jellinek, Vikas Singh, and Thomas Ristenpart. 2016. Controlling UAVs with Sensor Input Spoofing Attacks. In *10th USENIX Workshop on Offensive Technologies (WOOT 16).*

[18] Aolin Ding, Praveen Murthy, Luis Garcia, Pengfei Sun, Matthew Chan, and Saman Zonouz. 2021. Mini-Me, you complete me! data-driven drone security via DNN-based approximate computing. *RAID 2021 Proceedings - 24th International Symposium on Research in Attacks, Intrusions and Defenses.*

[19] Yefim Dinitz. 1970. Algorithm for Solution of a Problem of Maximum Flow in Networks with Power Estimation. *Soviet Math. Dokl.* 11 (1970), 1277–1280.

[20] Fan Fei, Zhan Tu, Ruikun Yu, Taegyu Kim, Xiangyu Zhang, Dongyan Xu, and Xinyan Deng. 2018. Cross-Layer Retrofitting of UAVs Against Cyber-Physical Attacks. In *2018 IEEE International Conference on Robotics and Automation (ICRA).*

[21] L. R. Ford and D. R. Fulkerson. 1956. Maximal Flow Through a Network. *Canadian Journal of Mathematics* 8 (1956), 399–404. https://doi.org/10.4153/CJM-1956-045-5

[22] Gazebo. 2022. Gazebo simulator. https://gazebosim.org/home

[23] Thomas Hamilton and David A. Ochmanek". 2020. *Operating Low-Cost, Reusable Unmanned Aerial Vehicles in Contested Environments: Preliminary Evaluation of Operational Concepts.* RAND Corporation, Santa Monica, CA.

[24] Daojing He, Yinrong Qiao, Shiqing Chen, Xiao Du, Wenjie Chen, Sencun Zhu, and Mohsen Guizani. 2019. A Friendly and Low-Cost Technique for Capturing Non-Cooperative Civilian Unmanned Aerial Vehicles. *IEEE Network* 33, 2 (2019), 146–151.

[25] Jinseob Jeong, Dongkwan Kim, Joonha Jang, Juhwan Noh, Changhun Song, and Yongdae Kim. 2023. Un-Rocking Drones: Foundations of Acoustic Injection Attacks and Recovery Thereof. *Proceedings of the Network and Distributed System Security Symposium.*

[26] Chijung Jung, Ali Ahad, Jinho Jung, Sebastian Elbaum, and Yonghwi Kwon. 2021. Swarmbug: Debugging Configuration Bugs in Swarm Robotics *(ESEC/FSE 2021).*

[27] Arslan Khan, Hyubgsub Kim, Byoungyoung Lee, Dongyan Xu, Antonio Bianchi, and Dave Jing Tian. 2021. M2MON: Building an MMIO-based Security Reference Monitor for Unmanned Vehicles. In *30th USENIX Security Symposium (USENIX Security 21).*

[28] Chung Hwan Kim, Taegyu Kim, Hongjun Choi, Zhongshu Gu, Byoungyoung Lee, X. Zhang, and Dongyan Xu. 2018. Securing Real-Time Microcontroller Systems through Customized Memory View Switching. In *Proceedings of the Network and Distributed System Security Symposium.*

[29] Hyungsub Kim, Muslum Ozgur Ozmen, Antonio Bianchi, Z Berkay Celik, and Dongyan Xu. 2021. PGFUZZ: Policy-Guided Fuzzing for Robotic Vehicles. In *Proceedings of the Network and Distributed System Security Symposium.*

[30] Hyungsub Kim, Ozgur Ozmen, Z Berkay Celik, Antonio Bianchi, and Dongyan Xu. 2022. PGPATCH: Policy-Guided Logic Bug Patching for Robotic Vehicles. *2022 IEEE Symposium on Security and Privacy.*

[31] Hyungsub Kim, Ozgur Ozmen, Z Berkay Celik, Antonio Bianchi, and Dongyan Xu. 2023. PatchVerif: Discovering Faulty Patches in Robotic Vehicles. *32nd USENIX Security Symposium (USENIX Security 23).*

[32] Taegyu Kim, Chung Hwan Kim, Junghwan Rhee, Fan Fei, Zhan Tu, Gregory Walkup, Xiangyu Zhang, Xinyan Deng, and Dongyan Xu. 2019. RVFuzzer: Finding input validation bugs in robotic vehicles through control-guided testing. In *Proceedings of the 28th USENIX Security Symposium.* 425–442.

[33] Taegyu Kim, Altay Ozen, Fan Fei, Zhan Tu, Xiangyu Zhang, Xinyan Deng, Dave Tian, Dongyan Xu, and Chung Hwan Kim. 2020. From Control Model to Program: Investigating Robotic Aerial Vehicle Accidents with MAYDAY. *29th USENIX Security Symposium (USENIX Security 20).*

[34] Patrick Koeberl, Steffen Schulz, Ahmad-Reza Sadeghi, and Vijay Varadharajan. 2014. TrustLite: A Security Architecture for Tiny Embedded Devices. In *Proceedings of the Ninth European Conference on Computer Systems.*

[35] Renju Liu and Mani Srivastava. 2017. PROTC: PROTeCting Drone's Peripherals through ARM TrustZone. In *Proceedings of the 3rd Workshop on Micro Aerial Vehicle Networks, Systems, and Applications.* 1–6.

[36] LLVM. 2022. LLVM insturction. https://llvm.org/docs/LangRef.html.

[37] LLVM. 2022. LLVM Project. https://llvm.org/

[38] Aaron Luo. 2016. Drones hijacking - multi-dimensional attack vectors and countermeasures. DefCon 24.

[39] MAVLink. 2022. Mavlink. https://mavlink.io/en/

[40] B Nassi, R Bitton, R Masuoka, A Shabtai, and Y Elovici. 2021. SoK: Security and Privacy in the Age of Commercial Drones. In *2021 IEEE Symposium on Security and Privacy.* 73–90.

[41] Job Noorman, Pieter Agten, Wilfried Daniels, Raoul Strackx, Anthony Van Herrewege, Christophe Huygens, Bart Preneel, Ingrid Verbauwhede, and Frank Piessens. 2013. Sancus: Low-Cost Trustworthy Extensible Networked Devices with a Zero-Software Trusted Computing Base. In *Proceedings of the 22nd USENIX Conference on Security Symposium.*

[42] OP-TEE. 2022. Open portable trust execution environment. https://www.op-tee.org/

[43] PX4. 2022. PX4 Autopilot. https://px4.io

[44] QGroundControl. 2019. QGroundControl. http://qgroundcontrol.com/

[45] Raul Quinonez, Jairo Giraldo, Luis Salazar, Erick Bauman, Alvaro Cardenas, and Zhiqiang Lin. 2020. SAVIOR: Securing Autonomous Vehicles with Robust Physical Invariants. In *29th USENIX Security Symposium (USENIX Security 20).* 895–912.

[46] Thomas Reps, Susan Horwitz, and Mooly Sagiv. 1995. Precise interprocedural dataflow analysis via graph reachability. In *Proceedings of the 22nd ACM SIGPLAN-SIGACT symposium on Principles of programming languages.* 49–61.

[47] Harshad Sathaye, Gerald LaMountain, Pau Closas, and Aanjhan Ranganathan. 2022. SemperFi: Anti-spoofing GPS Receiver for UAVs. *Proceedings of the Network and Distributed System Security Symposium.*

[48] Harshad Sathaye, Martin Strohmeier, Vincent Lenders, and Aanjhan Ranganathan. 2022. An Experimental Study of GPS Spoofing and Takeover Attacks on UAVs. *31st USENIX Security Symposium (USENIX Security 22).*

[49] Yunmok Son, Hocheol Shin, Dongkwan Kim, Youngseok Park, Juhwan Noh, Kibum Choi, Jungwoo Choi, and Yongdae Kim. 2015. Rocking Drones with Intentional Sound Noise on Gyroscopic Sensors. In *24th USENIX Security Symposium (USENIX Security 15).* 881–896.

[50] Yunmok Son, Hocheol Shin, Dongkwan Kim, Youngseok Park, Juhwan Noh, Kibum Choi, Jungwoo Choi, and Yongdae Kim. 2015. Rocking Drones with Intentional Sound Noise on Gyroscopic Sensors. In *Proceedings of the 24th USENIX Conference on Security Symposium (SEC'15).* USENIX Association, USA, 881–896.

[51] Jie Su, Jianping He, Peng Cheng, and Jiming Chen. 2016. A Stealthy GPS Spoofing Strategy for Manipulating the Trajectory of an Unmanned Aerial Vehicle. *IFAC-PapersOnLine* 49, 22 (2016), 291–296.

[52] Yulei Sui and Jingling Xue. 2016. SVF: interprocedural static value-flow analysis in LLVM. In *Proceedings of the 25th international conference on compiler construction.* 265–266.

[53] Timothy Trippel, Ofir Weisse, Wenyuan Xu, Peter Honeyman, and Kevin Fu. 2017. WALNUT: Waging Doubt on the Integrity of MEMS Accelerometers with Acoustic Injection Attacks. In *2017 IEEE European Symposium on Security and Privacy (EuroS&P).* 3–18.

[54] David I. Urbina, Jairo A. Giraldo, Alvaro A. Cardenas, Nils Ole Tippenhauer, Junia Valente, Mustafa Faisal, Justin Ruths, Richard Candell, and Henrik Sandberg. 2016. Limiting the Impact of Stealthy Attacks on Industrial Control Systems. In

Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. 1092–1105.

[55] Man Ki Yoon, Bo Liu, Naira Hovakimyan, and Lui Sha. 2017. VirtualDrone: Virtual sensing, actuation, and communication for attack-resilient unmanned aerial systems. *ACM/IEEE 8th International Conference on Cyber-Physical Systems, ICCPS 2017*.

[56] Kexiong (Curtis) Zeng, Shinan Liu, Yuanchao Shu, Dong Wang, Haoyu Li, Yanzhi Dou, Gang Wang, and Yaling Yang. 2018. All Your GPS Are Belong To Us: Towards Stealthy Manipulation of Road Navigation Systems. In *27th USENIX Security Symposium (USENIX Security 18)*.

[57] Lin Zhang, Xin Chen, Fanxin Kong, and Alvaro A. Cardenas. 2020. Real-Time Attack-Recovery for Cyber-Physical Systems Using Linear Approximations. In *Proceedings - Real-Time Systems Symposium*. 205–217.

A SOURCE-SINK ANALYSIS ALGORITHM

Algorithm 3: Source-sink analysis

```
Function ForwardBFS(sources):
    Input sources: set of start nodes (sources) for forward BFS
    Output visited: set of nodes visited through forward BFS

    Q ← sources;
    visited ← soures;
    while Q ≠ ∅ do
        node ← Q.pop();
        edges_out ← outgoing edges of node;
        foreach out ∈ edges_out do
            child ← child node of out;
            if child ∉ visited then
                Q.push(child);
                visited.push(child);
            end
        end
    end
    return visited;
Function BackwardBFS(sinks):
    Input sinks: set of start nodes (sinks) for backward BFS
    Output visited: set of nodes visited through backward BFS

    Q ← sinks;
    visited ← sinks;
    while Q ≠ ∅ do
        node ← Q.pop();
        edges_in ← incoming edges of node;
        foreach in ∈ edges_in do
            parent ← source node of in;
            if parnet ∉ visited then
                Q.push(parent);
                visited.push(parent);
            end
        end
    end
    return visited;
Function Main(inp, outp):
    Input inp: set of sources in DFG
    Input outp: set of sinks in DFG
    Output SCV_c: set of SCV candidates

    Con ← ForwardBFS(inp);
    S ← BackwardBFS(outp);
    SCV_c ← Con ∩ S;
    return SCV_c;
```

B AN RV CONTROL PROGRAM'S INPUT AND OUTPUT

Table 5 shows a list of inputs and outputs of the RV control program we used. We identified two types of inputs and one type of output. Variables related to configuration parameters store the values of one or more configuration parameters. For example, the variable _pid_rate_roll stores eight configuration parameter values. We identified the variables in which measurements from the physical sensor are stored. Finally, we set the variable *pwm_output* to the output. *pwm_output* is a value obtained by changing actuator outputs to the PWM range.

C MULTI-STAGE ATTACK

SCVMON can successfully recover from simultaneous data-oriented attacks that change sensor measurements and configuration parameters. However, an attacker can destabilize RVs by performing attacks at time intervals[11] rather than changing inputs simultaneously. Therefore, we discuss whether SCVMON can properly recover these data-oriented attacks consisting of multiple steps.

A key feature of these attacks is that the final step is what destabilizes an RV. Attacks that change the input before the final step do not destabilize the RV, but if the input does change in the final step, the safety of the RV is affected by generating inappropriate control commands along with changes to the inputs in the previous steps. In other words, the actuator value is abnormally changed in the final step, and SCVMON executes the recovery mode. However, as described in Section 4.3.2 and Section 4.4.1, attacks that change the inputs inevitably change the value of one or more mSCVs discontinuously. Therefore, if the data-oriented attacks in the previous steps are performed, SCVMON detects inappropriate changes in specific mSCVs and switches them to the "caution" state. However, at this time, the actuator value does not change abnormally, so the recovery mode is not executed. If the previous steps only change the static mSCVs, the mSCVs continuously maintain the caution state. Therefore, if the final step attack changes other mSCVs and this causes the actuator value to change abnormally, SCVMON recovers not only the mSCV changed in the final step but also the mSCVs changed in the previous steps.

We also discuss the case in which dynamic mSCVs change in the previous steps. If the attacks in the previous steps change the dynamic mSCVs, those mSCVs are converted to the caution state. However, if the actuator values are not affected, SCVMON switches back to the "normal" state after 400 cycles. After that, if the actuator value is also abnormally changed just like how a specific mSCV becomes changed through the final step of the attack, SCVMON executes the recovery mode. However, since the mSCVs changed in the previous steps were already converted to a normal state, SCVMON does not recover them. Note that the RVs are able to operate normally before the final step of the attack is performed. Therefore, as the mSCVs changed in the final step are recovered, the RVs can continue their safe operation. This is because it blocks the interaction between the changes in the previous steps and the changes in the final step. In conclusion, in this case, it is not possible to fully recover the changes caused by the previous steps attacks, but SCVMON can help RVs to operate safely.

D DETAILS OF SCVS

The mSCVs we use to detect and recover various attacks is represented in Table 6. Table 6 shows a brief description of these variables, the function where monitoring is performed, recovery-suitability, and thresholds for attack detection. All 27 variables are mSCVs and have moderate or high recovery-suitability, and each mSCV has one or two thresholds. Threshold 1 represents the difference between single cycles while Threshold 2 is the sum of differences

Table 5: RV control program inputs and outputs

Type	Group	Variables
Parameter	AHRS	gps_gain, _gps_use, _kp_yaw, _kp, _wind_max, _trim, _board_orientation, beta, _gps_minsats, _ekf_type, _custom_roll, _custom_pitch, _custom_yaw
	ARMING	accel_error_threshold
	ATC	_slew_yaw, _accel_yaw_max, _rate_bf_ff_enabled, _accel_roll_max, _accel_pitch_max, _input_tc, _angle_boost_enabled, _p_angle_roll, _p_angle_pitch, _p_angle_yaw, _angle_limit_tc, _ang_vel_roll_max, _ang_vel_roll_max, _ang_vel_pitch_max, _ang_vel_yaw_max, _pid_rate_roll, _pid_rate_pitch, _pid_rate_yaw, _thr_mix_min, _thr_mix_max, _thr_mix_man
	COMPASS	_state, _declination, _offset_max. _filter_range, _custom_roll, _custom_pitch, _custom_yaw
	PSC	_p_pos_z, _p_vel_z, _pid_accel_z, _p_pos_xy, _pid_vel_xy, _lean_angle_max
	WPNAV	_wp_speed_cms, _wp_radius_cm, _wp_speed_up_cms, _wp_speed_down_cms, _wp_accel_cmss, _wp_accel_z_cmss, _rangefinder_use
	INS	_gyro_offset, _accel_scale, _accel_offset, _gyro_filter_cutoff, _accel_filter_cutoff, _use, _still_threshold, _gyro_cal_timing, _trim_option, _acc_body_aligned, _accel_pos, _gyro_id, _accel_id
	LOIT	_angle_max, _speed_cms, _accel_cmss, _brake_accel_cmss, _brake_jerk_max_cmsss,_brake_delay
	CIRCLE	_radius, _rate
	BATT	_params
	GND	ground_pressure, _user_ground_temperature, _alt_offset, _primary_baro, _filter_range
	MOT	_yaw_headroom, _thrust_curve_expo, _spin_max, _batt_voltage_max, _batt_voltage_min, _batt_current_max, _pwm_type, _pwm_min, _pwm_max, _spin_min, _spin_arm, _safe_time, _batt_current_time_constant, _throttle_hover, _yaw_servo_angle_max_deg, _spool_up_time, _boost_scale, _batt_idx, _slew_up_time, _slew_dn_time,
	EK2	_enable, _fusionModeGPS, _gpsHorizVelNoise, _gpsVertVelNoise, _gpsVelInnovGate, _gpsHorizPosNoise, _gpsPosInnovGate, _gpsGlitchRadiusMax, _baroAltNoise, _hgtInnovGate, _hgtDelay_ms, _magNoise, _magCal, _magInnovGate, _easNoise, _tasInnovGate, _rngNoise, _rngInnovGate, _maxFlowRate, _flowNoise, _flowInnovGate, _flowDelay_ms, _gyrNoise, _accNoise, _gyroBiasProcessNoise, _gyroScaleProcessNoise, _accelBiasProcessNoise, _windVelProcessNoise, _wndVarHgtRateScale, _gpsCheckScaler, _yawNoise, _yawInnovGate, _tauVelPosOutput, _magEarthProcessNoise, _magBodyProcessNoise, _useRngSwHgt, _terrGradMax, _rngBcnNoise, _rngBcnInnovGate, _rngBcnDelay_ms, _useRngSwSpd, _extnavDelay_ms, _mag_ef_limit, _hrt_filt_freq
	RTL	rtl_altitude, rtl_cone_slope, rtl_speed_cms, rtl_alt_final, rtl_climb_min, rtl_loiter_time
	LAND	land_speed, land_speed_high
Sensor	Gyro	_gyro[i]
	Accel	_accel[i]
	Baro	sensors[i]
	GPS	state[i]
	Compass	_state[i]
Output	PWM	pwm_output

Table 6: Details of SCVs

No	SCV	Function	R	Threshold1	Threshold2 (window)	Description
1	get_pwm_output_max()	output_to_pwm()	H	0		maximum PWM value that can be output to motors
2	get_pwm_output_min()	output_to_pwm()	H	0		minimum PWM value that can be output to motors
3	_spin_max	thrust_to_actuator()	H	0		throttle out ratio which produces the maximum thrust
4	_spin_min	thrust_to_actuator()	H	0		throttle out ratio which produces the minimum thrust
5	compensation_gain	output_armed_stabilizing()	H	0		compensation for battery voltage and altitude
6	throttle_thrust	output_armed_stabilizing()	M	0.1	0.2(7)	throttle thrust input value
7	get_rate_roll_pid().get_p()	rate_controller_run()	H	0		roll axis rate controller P gain
8	get_rate_roll_pid().get_i()	rate_controller_run()	H	0		roll axis rate controller I gain
9	get_rate_roll_pid().get_d()	rate_controller_run()	H	0		roll axis rate controller D gain
10	get_rate_roll_pid().get_ff()	rate_controller_run()	H	0		roll axis rate controller feed forward
11	get_rate_pitch_pid().get_p()	rate_controller_run()	H	0		pitch axis rate controller P gain
12	get_rate_pitch_pid().get_i()	rate_controller_run()	H	0		pitch axis rate controller I gain
13	get_rate_pitch_pid().get_d()	rate_controller_run()	H	0		pitch axis rate controller D gain
14	get_rate_pitch_pid().get_ff()	rate_controller_run()	H	0		pitch axis rate controller feed forward
15	get_rate_yaw_pid().get_p()	rate_controller_run()	H	0		yaw axis rate controller P gain
16	get_rate_yaw_pid().get_i()	rate_controller_run()	H	0		yaw axis rate controller I gain
17	get_rate_yaw_pid().get_d()	rate_controller_run()	H	0		yaw axis rate controller D gain
18	get_rate_yaw_pid().get_ff()	rate_controller_run()	H	0		yaw axis rate controller feed forward
19	gyro_latest.x	rate_controller_run()	M	0.02	0.05(5)	corrected gyro x-axis using the latest INS data
20	gyro_latext.y	rate_controller_run()	M	0.02	0.05(5)	corrected gyro y-axis using the latest INS data
21	gyro_latext.z	rate_controller_run()	M	0.02	0.05(20)	corrected gyro z-axis using the latest INS data
22	euler_roll_angle	input_euler_angle_roll_pitch_yaw()	M	0.05	0.1(5)	desired roll angle
23	euler_pitch_angle	input_euler_angle_roll_pitch_yaw()	M	0.05	0.1(5)	desired pitch angle
24	euler_yaw_angle	input_euler_angle_roll_pitch_yaw()	M	0.15	0.25(5)	yaw heading
25	euler_roll_angle	input_euler_angle_roll_pitch_euler _rate_yaw()	M	0.05	0.1(5)	desired roll angle
26	euler_pitch_angle	input_euler_angle_roll_pitch_euler _rate_yaw()	M	0.05	0.1(5)	desired pitch angle
27	euler_yaw_rate	input_euler_angle_roll_pitch_euler _rate_yaw()	M	0.15	0.25(5)	desired yaw rate

R: Recovery-Suitability, Threshold1: Threshold for the difference in mSCV values of two successive cycles
Threshold2: Threshold for the accumulated difference in mSCV values over several successive cycles

Table 7: Details of attacks

No	Attack target(sensor)	Attack target(parameters)
1(1)	gyro = 0.7	
2(3)	gyro.y = 0.1(S)	ATC_RAT_PIT_P=0.35(N)
3	gyro = 0.1(S)	ATC_RAT_RLL_P=0.35(N), ATC_RAT_PIT_P=0.35(N)
4	gyro = 0.1(S)	ATC_RAT_RLL_P=0.35(N), ATC_RAT_RLL_I=0.6(N), ATC_RAT_RLL_D=0.03(N), ATC_RAT_PIT_P=0.35(N),ATC_RAT_PIT _I=0.6(N), ATC_RAT_PIT_D=0.03(N), ATC_RAT_YAW_P=0.6(N), ATC_RAT_YAW_I=0.06(N), ATC_RAT_YAW_D=0.02(N)
5	baro(curr_alt) = 1010(curr_alt + 10)(S)	
6(4)	baro(curr_alt) = 1010(curr_alt + 10)(S)	ATC_RAT_PIT_P=3(N)
7	accel.z = -50	
8	accel.z = -50	PSC_ACCZ_P=1.5(N), PSC_ACCZ_I=3(N), PSC_ACCZ_D=0.4(N)
9	curr_pos = curr_pos+200(S)	
10	curr_pos = curr_pos+200(S)	PSC_POSXY_P=2(N), PSC_VELXY_P=6(N), PSC_VELXY_I=1(N), PSC_VELXY_D=1(N)
11		MOT_SPIN_MAX=0(N), MOT_SPIN_MIN=1(N)
12		MOT_PWM_MAX=2000(N), MOT_PWM_MIN=0(N) , MOT_SPIN_MAX=0(N), MOT_SPIN_MIN=1(N)
13	gyro = 10, accel.z = -100	
14(2)	gyro = 0.1(S), baro(curr_alt) = 1010(curr_alt + 10)(S)	
15	gyro = 10, accel.z = -100, baro(curr_alt) = 10000	
16(5)	gyro = 10, accel.z = -100, baro(curr_alt) = 10000, gps(curr_pos)=1000000	
17	gyro = 0.1(S), accel.z =-10(S)	ATC_RAT_RLL_P=0.35(N), ATC_RAT_RLL_I=0.6(N), ATC_RAT_RLL_D=0.03(N), ATC_RAT_PIT_P=0.35(N), ATC_RAT_PIT _I=0.6(N), ATC_RAT_PIT_D=0.03(N), ATC_RAT_YAW_P=0.6(N), ATC_RAT_YAW_I=0.06(N), ATC_RAT_YAW_D=0.02(N), PSC_ACCZ_P=1.5(N), PSC_ACCZ_I=3(N), PSC_ACCZ_D=0.4(N)
18	gyro = 0.1(S), accel.z =-10(S), baro(curr_alt) = 1010(curr_alt + 10)(S)	ATC_RAT_RLL_P=0.35(N), ATC_RAT_RLL_I=0.6(N), ATC_RAT_RLL_D=0.03(N), ATC_RAT_PIT_P=0.35(N), ATC_RAT_PIT _I=0.6(N), ATC_RAT_PIT_D=0.03(N), ATC_RAT_YAW_P=0.6(N), ATC_RAT_YAW_I=0.06(N), ATC_RAT_YAW_D=0.02(N), PSC_ACCZ_P=1.5(N), PSC_ACCZ_I=3(N), PSC_ACCZ_D=0.4(N), PSC_POSZ_P=3(N)
19(6)	gyro = 0.1(S), accel.z =-10(S), baro(curr_alt) = 1010(curr_alt + 10)(S), gps(curr_pos) =gps(curr_pos)+200(S)	ATC_RAT_RLL_P=0.35(N), ATC_RAT_RLL_I=0.6(N), ATC_RAT_RLL_D=0.03(N), ATC_RAT_PIT_P=0.35(N), ATC_RAT_PIT _I=0.6(N), ATC_RAT_PIT_D=0.03(N), ATC_RAT_YAW_P=0.6(N), ATC_RAT_YAW_I=0.06(N), ATC_RAT_YAW_D=0.02(N), PSC_ACCZ_P=1.5(N), PSC_ACCZ_I=3(N), PSC_ACCZ_D=0.4(N), PSC_POSZ_P=3(N), PSC_POSXY_P=2(N), PSC_VELXY _P=6(N), PSC_VELXY_I=1(N), PSC_VELXY_D=1(N), PSC_ACC_XY_FILT=5(N), PSC_ANGLE_MAX=45(N)
20	gyro = 10, accel.z = -100, baro(curr_alt) = 10000, gps(curr_pos)=1000000	ATC_RAT_RLL_P=0.35(N), ATC_RAT_RLL_I=0.6(N), ATC_RAT_RLL_D=0.03(N), ATC_RAT_PIT_P=0.35(N), ATC_RAT_PIT _I=0.6(N), ATC_RAT_PIT_D=0.03(N), ATC_RAT_YAW_P=0.6(N), ATC_RAT_YAW_I=0.06(N), ATC_RAT_YAW_D=0.02(N), PSC_ACCZ_P=1.5(N), PSC_ACCZ_I=3(N), PSC_ACCZ_D=0.4(N), PSC_POSZ_P=3(N), PSC_POSXY_P=2(N), PSC_VELXY _P=6(N), PSC_VELXY_I=1(N), PSC_VELXY_D=1(N), PSC_ACC_XY_FILT=5(N), PSC_ANGLE_MAX=45(N), MOT_PWM _MAX=2000(N), MOT_PWM_MIN=0(N) , MOT_SPIN_MAX=0(N), MOT_SPIN_MIN=1(N), MOT_THST_EXPO=0.8(N), ATC_ACCEL_R_MAX=180000(N), ATC_ACCEL_P_MAX=180000(N), ATC_ACCEL_Y_MAX=72000(N), ATC_SLEW _YAW=18000(N), ATC_INPUT_TC=1(N)

S: Stealthy attack, N: Change the parameters to a value within the normal range

between single cycles. At this time, if the window size is five, the sum of five consecutive values of the difference between single cycles is calculated. The sum of differences calculated in this way is compared to Threshold 2 to detect an attack.

E DETAILS OF ATTACKS

Section 5.2 introduces 20 attacks that change various inputs. Table 7 shows the details of these 20 attacks. A value of 6(4) in the "No" column signifies the sixth attack from Table 3 and the fourth attack from Table 2.

F PROOF OF COMPLETENESS OF MONITORING SET

We prove that the mSCV set extracted through our s-t cut algorithm satisfies the first condition. The first condition is "All paths from source to sink must pass through at least one mSCV."

Proof) We prove this through mathematical induction. In the first step of the s-t cut algorithm, the set of mSCVs (=$mSCV_1$) is a set of sinks. In this case, all paths from the source to the sinks essentially pass through the set of sinks, so at the initial step, this proposition is true. We assume that the corresponding condition is satisfied in step n and prove that it is satisfied in step $n+1$. We assume that the set of mSCVs identified in step n ($mSCV_n = \{v_1, ..., v_k\}$), the situation in which n backward searches are performed, satisfies the first condition. In the $n+1$ step, a one-step backward search is performed on $v_j (1 \le j \le k)$, which has a low recovery-suitability. The variables $p_1, ..., p_{a+b}$ are parent nodes identified through a one-step backward search from v_j. The variables $p_1, ..., p_a$ are classified as safety-critical, and thus, they are added to the $mSCV_{n+1}$ set. The variables $p_{a+1}, ..., p_{a+b}$ are classified as non-safety-critical. As a result, in step $n+1$, the following $mSCV_{n+1} = (mSCV_n \setminus \{v_j\}) \cup \{P_1, ..., P_a\}$ is generated. Any path that does not go through v_j must go through at least one element of $(mSCV_n \setminus \{v_j\})$ by assumption. On the other hand, all paths through v_j must pass through the identified parent nodes. Note that the edges between $p_{a+1}, ..., p_{a+b}$ and v_j are considered broken according to the previous discussion, so all paths through v_j pass through $p_1, ..., p_a$. That is, all paths from sources to sinks pass through the element of $mSCV_{n+1}$. Therefore, by mathematical induction, it is proven that the mSCV set extracted by our s-t cut algorithm satisfies the first condition.