AVGuardian: Detecting and Mitigating Publish-Subscribe Overprivilege for Autonomous Vehicle Systems

David Ke Hong, John Kloosterman, Yuqi Jin, Yulong Cao Qi Alfred Chen Scott Mahlke, Z. Morley Mao
University of Michigan *UC Irvine* *University of Michigan*
{*kehong, jklooste, yuqijin, yulongc*}*@umich.edu* *alfchen@uci.edu* {*mahlke, zmao*}*@umich.edu*

***Abstract*—Autonomous vehicle (AV) software systems are emerging to enable rapidly developed self-driving functionalities. Since such systems are responsible for safety-critical decisions, it is necessary to secure them in face of cyber attacks. Through an empirical study of representative AV software systems *Baidu Apollo* and *Autoware*, we discover a common *overprivilege* problem with the publish-subscribe communication model widely adopted by AV systems: due to the coarse-grained message design for the publish-subscribe communication, some message fields are over-granted with publish/subscribe permissions. To comply with the least-privilege principle and reduce the attack surface resulting from such problem, we argue that the publish/subscribe permissions should be defined and enforced at the granularity of *message fields* instead of messages.**

To systematically address such publish-subscribe overprivilege problems, we present AVGuardian, a system that includes (1) a static analysis tool that detects overprivilege instances in AV software and generates the corresponding access control policies at the message field granularity, and (2) a low-overhead, module-transparent, runtime publish/subscribe permission policy enforcement mechanism to perform online policy violation detection and prevention. Using our detection tool, we are able to automatically detect 581 overprivilege instances in total in Baidu Apollo. To demonstrate the severity, we further constructed several concrete exploits that can lead to vehicle collision and identity theft for AV owners, which have been reported to Baidu Apollo and confirmed as valid. For defense, we prototype and evaluate the policy enforcement mechanism, and find that it has very low overhead, does not affect original AV decision logic, and also is resilient to message replay attacks.

1. Introduction

Autonomous driving holds the promise to improve road safety and significantly improve transportation mobility efficiency in our daily lives. Advanced autonomous driving algorithms and software are gaining importance. Autonomous vehicle (AV) systems are being developed and deployed in real vehicles [15], [34], [43], [52] and have demonstrated great promise towards full autonomous driving in the near future. Despite this rapid development, AV systems are facing a number of cybersecurity threats, for example, attacks on automobile Electronic Control Unit (ECU) [32], [59], [70], [75] and key sensing devices for autonomous driving [27], [47], [79]. However, a highly critical attack surface is still underexplored so far: the AV software systems for making autonomous driving decisions. Since these decisions have direct impact on road safety, it is necessary to understand potential security vulnerabilities in the design and implementations of AV software systems, and proactively address them in the AV system development stage.

We observe that AV software systems are usually composed by a number of key self-driving modules, which interact through *a publish-subscribe communication model* to exchange computation states using different types of messages defined by AV developers. Modules are granted publish or subscribe permission to be a publisher or subscriber for certain types of messages. To improve the functionality and reliability of autonomous driving, these modules are becoming feature rich and as a consequence the messages also become increasingly complex. Based on our empirical study on two popular AV software platforms *Baidu Apollo* [14] and *Autoware* [17], we observed 60-80 types of publish-subscribe messages, each consisting of up to dozens of fields. Despite the new functionality enabled by the rich set of fields, this complex message structure also introduces a new security problem to the publish-subscribe messaging system in an AV system. Specifically, in both *Baidu Apollo* and *Autoware*, we found a number of code examples indicating a common *overprivilege problem* with this messaging model, stemming from a lack of sufficient granularity when granting publish or subscribe permissions of key messages to a module.

Through further in-depth study on both AV software systems, we discover two common types of overprivilege in its publish-subscribe messaging, when either: 1) fields in a published message are not used in a particular subscriber; 2) the values of certain fields in a published message are directly copied from other messages subscribed to by the publisher. We characterize these behaviors as subscriber- or publisher-side overprivilege in AV systems, respectively, since the granted publish or subscribe permission of a message to a module does not follow the least-privilege principle at the message field granularity. As shown later in §2.2, such overprivilege patterns are found to generally appear across both the old and the latest versions of both AV systems. We have constructed several concrete exploits of such non-compliance, which demonstrates that this problem indeed exposes a new attack surface to AV systems and may lead to vehicle collision and identity theft for AV owners under a realistic threat model (detailed in §3) inspired by existing automotive attack surface analysis. Therefore,

we argue that this publish-subscribe overprivilege problem should be fully addressed for attack surface reduction to ensure secure AV software system design. In particular, to overcome this problem, the publish/subscribe permission should be defined at a message field granularity. Subscribe permission for a field should be granted to a subscriber only when the subscriber uses that field for computation, and publish permission should be granted to a publisher only when the publisher modifies the state of that field before publishing.

However, enabling such fine-grained permission control can be challenging in AV software systems, because AV software development is a multidisciplinary task and typically conducted by a large team of developers with different domains of expertise. Moreover, some AV software systems are built upon an open platform to encourage open-source contribution of self-driving algorithms and code. As a result, AV system designers may not have complete knowledge about the exact usage of message fields in an AV software module and tend to include as many fields in each message as possible to simplify software development. Even if an explicit message field-level permission model and static access control policy enforcement are enabled in AV software, we cannot fully trust a module to comply with the enforcement at run time, since a module can be compromised and the pre-defined access control logic can be bypassed. Because the runtime policy enforcement will be performed on every published or subscribed message, the runtime enforcement must incur little overhead. We propose a systematic overprivilege detection and mitigation approach to address these challenges for AV software.

To effectively detect and mitigate publish-subscribe overprivilege in AV systems, we propose AVGuardian, consisting of a static analysis tool that systematically detects overprivilege instances in AV software and generates the corresponding access control policies at the message field granularity, and a runtime policy enforcement mechanism to perform online policy detection and prevention. Our static analysis approach handles complex real-world C++ source code, including virtual functions and asynchronous programming models, to both achieve high precision in overprivilege detection and prevent under-granting publish/subscribe permissions. Our runtime policy enforcement can defend against publish-subscribe overprivilege with a single module compromised, and does not require any additional efforts from AV software developers or changes to the AV software development process. In addition, our design further includes a defense mechanism against message relay attacks. As we observe that several popular AV software systems [14], [17], [80] are developed on top of the ROS middleware [40] (a layer between the upper AV software modules and underlying commodity OS), we instrument ROS to prototype the policy enforcement component of AVGuardian so that it is transparent to the upper AV software modules (i.e., without requiring modification of existing AV software modules for them to be safeguarded by AVGuardian). Trace-based performance evaluation in realistic setup shows that the runtime policy enforcement incurs only 10-millisecond increase of the end-to-end delay for AV's control decision making and does not affect original decision logic. Also, it effectively detects message replay attacks with zero false positives and zero false negatives in realistic exploitation scenarios.

We have performed responsible disclosure to the Baidu Apollo development team, and have received confirmation that our attack findings are all valid under our threat model, and the publish-subscribe overprivileged attack is indeed a general security challenge in AV software development. The team also commented that it is highly beneficial to have a systematic and automated approach to detect and prevent overprivilege problems, which is exactly the research goal in this paper.

The contributions of this paper are as follows:

- We discover the overprivilege problem in publish-subscribe messaging model for AV software systems, and perform the first characterization and systematic study. To demonstrate the severity of such problem, we construct three concrete attacks by exploiting vulnerabilities resulting from overprivilege problems in GNSS and LiDAR driver modules. Video demos of the attacks are available at **https://sites.google.com/view/avguardian**.
- We design and implement a data-flow analysis tool to help AV developers perform static detection of publish-subscribe overprivilege problems in AV software and generate fine-grained permission control policies at the message field level to mitigate the security consequence from overprivilege. Based on the groud-truths of overprivilege identified from runtime profiling and manual inspection, we observe zero false positives in overprivilege detection and less than 1.7% false negative rate. Using this tool, we are able to automatically detect 520 subscriber-side overprivilege instances and 61 publisher-side overprivilege instances in Baidu Apollo.
- We design an efficient and module-transparent policy enforcement solution to perform online detection and prevention of violation of permission control policies for publish-subscribe communication in ROS-based AV systems. We prototype this solution in ROS and find that it incurs very low overhead, i.e., only 8-millisecond increase in end-to-end delay in Baidu Apollo, does not affect original AV decision logic, and also is resilient to message replay attacks.

2. Background & Motivation

This section introduces the background of AV software systems and the publish-subscribe messaging model, and presents our discovered overprivilege problems in this messaging model on representative AV software systems.

2.1. AV Software System

The software architecture of existing AV systems falls into two categories: model-based [14] and end-to-end [57]. Our study focuses on model-based systems since such designs have already been adopted in many real-world AV systems [15], [17]. The model-based design is most common amongst state-of-the-art AV systems [97]. Figure 1 shows a typical processing pipeline by a set of modules in model-based AV systems to perform key self-driving

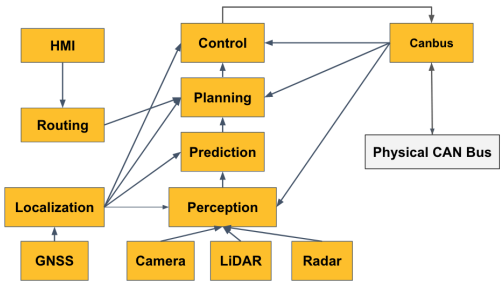

Figure 1. Typical architecture of AV software systems (based on Baidu Apollo): rectangles representing a ROS node/nodelet, arrows representing the ROS message flow through the publish-subscribe communication.

functionalities, including localization, routing, obstacle perception and prediction, path planning, and control decision execution. AV software also contains driver modules for peripheral sensor devices, such as GNSS, LiDAR, radar, and cameras. Similar to the architecture of ECUs and CAN bus in commodity automobiles, these modules are instantiated as nodes (each in separate processes) that run on a middleware such as ROS [40] and communicate through a publish-subscribe message channel, acting like a virtual CAN bus for AV systems. In such message channels, the producer of a message is called a *publisher* and the consumer is called a *subscriber*.

In the context of AV systems, the computation pipeline of AV software is dictated by the publish-subscribe message flow. Sensing input from peripheral devices is processed, module-by-module, until a final control decision is reached and executed on the physical actuators. Thus, the messages sent between these modules are responsible for mission-critical communication and directly influence end-to-end self-driving decisions in AV systems. Therefore, safeguarding the messaging channel in AV systems is critical to ensure secure and safe autonomous driving.

2.2. Publish-Subscribe Overprivilege Problem

Due to the criticalness of these messages, the security of this publish-subscribe message channel has already attracted attention of the research community [44], [58], [64], [92]. A state-of-the-art solution, SROS (Security Enhancements for ROS) [44], [92], defines a message-level publish-subscribe permission model and authentication mechanism to enhance the security of publish-subscribe messaging in ROS. However, we find that such message-level permission granting is actually not fine-grained enough to satisfy the least-privilege principle [85]. Based on our investigation on the Baidu Apollo and Autoware AV software system, we discover overprivilege problems when messages are both published and subscribed to.

1) **Publisher-side overprivilege:** In message publishers, the values of some fields in published messages may be directly copied from messages that module subscribes to. That is, the publisher only copies these fields without changing, but the granted publish permission allows both value copying and changing, which thus grants more than what is needed.
2) **Subscriber-side overprivilege:** In message subscribers, certain fields in a message may be received but not used in the subscribing module. In other words, the subscriber is over granted with the subscribe permission for these fields.

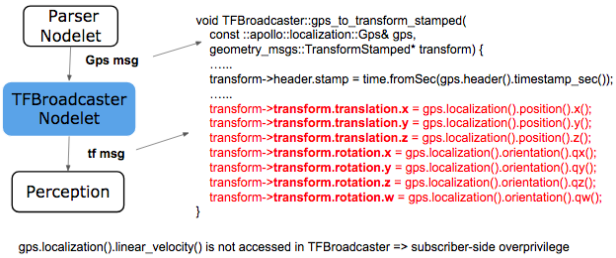

Figure 2. Examples of publisher- and subscriber-side overprivileges.

```
message Gps {
  optional apollo.common.Header header;
  optional apollo.localization.Pose localization;
}

message Pose {
  optional apollo.common.PointENU position;
  optional apollo.common.Quaternion orientation;
  optional apollo.common.Point3D linear_velocity;
  optional apollo.common.Point3D linear_acceleration;
  optional apollo.common.Point3D angular_velocity;
  optional double heading;
  optional apollo.common.Point3D linear_acceleration_vrf;
  optional apollo.common.Point3D angular_velocity_vrf;
  optional apollo.common.Point3D euler_angles;
}
```

Figure 3. Field definition of the `Gps` message in Baidu Apollo

Figure 2 illustrates a real example of subscriber-side overprivilege on the Gps message (defined in Figure 3) and publisher-side overprivilege on tf message [48] at the TFBroadcaster node of the GNSS driver module of Apollo. First, the localization.linear_velocity field in subscribed Gps messages is never used in any code path of TFBroadcaster node. Thus, this is a subscriber-side overprivilege on Gps.localization.linear_velocity because read permission for this field is granted to TFBroadcaster but never used in it. Second, the state of the transform field in published tf messages is always copied from the localization field in subscribed Gps messages, which is a publisher-side overprivilege on tf.transform because TFBroadcaster does not need to change the value of tf.transform.

Even though these two example overprivilege problems are subtle, we find that they can have severe security and safety implications. As detailed in §8.1, an attacker can cause an AV running Apollo to lose sight of a front vehicle and crash into it by exploiting these over-granted privileges. This attack is demonstrated in our attack demo videos [16], and its validity has been confirmed by the Baidu Apollo developer team.

General existence of publish-subscribe overprivilege in ROS-based AV systems. Beside Apollo, we studied another popular open-source AV system Autoware [17], which is also built upon the ROS middleware. As shown in Table 1, we are also able to discover many publisher and subscriber-side overprivilege instances in key AV modules such as Perception, Planning and Actuation [17]. These results concretely show that the publish-subscribe overprivilege problem generally exists in ROS-based AV systems today.

Considering the general existence of the publish-subscribe overprivilege problem and its severity in AV systems, it is thus highly necessary to develop solutions to fully eliminate the problem early at the AV system

Overprivileged node	Type	Affected topic	Affected fields
waypoint_follower's twist_gate [21]	Pub	/vehicle_cmd	ctrl_cmd, steer_cmd, accel_cmd, brake_cmd, gear, lamp_cmd, emergency
AS [18]	Pub	/as/arbitrated_speed_commands	speed
lidar_tracker's obj_reproj [20]	Sub	/image_obj_tracked	total_num, real_data, lifespan
autoware_connector's can_odometry [19]	Sub	/vehicle_status	drivemode, steeringmode, gearshift, drivepedal, brakepedal, lamp, light

TABLE 1. SUMMARY OF OVERPRIVILEGED INSTANCES IN AUTOWARE. IN THE "TYPE" COLUMN, "PUB" MEANS PUBLISHER-SIDE OVERPRIVILEGE AND "SUB" MEANS SUBSCRIBER-SIDE OVERPRIVILEGE.

development stage. Thus, in this paper, we fulfill this very need by being the first to develop a systematic solution.

3. Threat Model

In this work, we assume that the attacker can fully compromise a single ROS module N in the victim AV system, which enables the attacker to: 1) sniff contents of arbitrary subscribed ROS-layer messages that N has been authorized to subscribe; 2) modify or inject ROS-layer messages that N has been authorized to publish; and 3) bypass or invalidate defense mechanisms on N. Though it is possible to compromise multiple modules, we believe that it is harder for attackers and thus less realistic.

Assuming a single compromised ROS module might be considered to be too strong as the threat model. However, we argue that it is reasonable for our work since (1) this is a common threat model considered by previous ROS security work [58], [64], [92], (2) our work aims at developing defense solutions, so our contribution can be even more valuable if we can proactively and systematically solve a problem with strong adversaries, and (3) specifically for AV systems, such threat model can be particularly realistic when N is a driver module for peripheral devices. Previous work has concretely demonstrated that all types of peripheral devices of an automobile, e.g., bluetooth and cellular, can be fully compromised remotely through common software vulnerabilities such as buffer overflow [27], [32], [47], [59], [70], [76]. By inspecting the commit logs of Baidu Apollo's open-source software repository [14], various patches can be found related to common implementation mistakes, such as out-of-bound array indices, uninitialized variables, and wrong definition of if-else conditions, in the driver modules of LiDAR, GNSS, radar and CAN bus devices [3]–[12]. Thus, it is likely that similar software security problems in traditional automobiles' peripheral devices also exist in the driver modules of AV's peripheral devices, making these driver modules vulnerable to remote compromises. In particular, some AV software systems are encouraging open contribution [54] or peripheral hardware integration from third-party vendors [13], which is a common source for vulnerable code in automobile systems [59]. As a concrete attack scenario, one may exploit software vulnerability in GPS receivers' OS [79] or GPS daemon [1] to compromise a GPS receiver as the attack entry point, and then gain remote control of Apollo's GNSS driver module by exploiting its memory bugs [3]–[7].

We assume that the target AV software is from trusted developers, and the underlying middleware (e.g., ROS) has been safeguarded with state-of-the-art authentication and access control mechanisms provided by Secure ROS (SROS) [41], [45], [92]. We assume that the access control policies, i.e., granted publish and subscribe permissions, are correctly enforced in SROS. Since this work focuses on the overprivilege problem described in §2.2, the security of the policy enforcement in SROS is out of the scope of this work.

Thus, after compromising module N, the attacker can perform any action allowed by the granted publish and subscribe permissions in SROS (introduced in §2). For example, the attacker can abuse the over-granted write permission of publish-overprivileged fields by publishing malicious information in the overprivileged fields, or abuse the over-granted read permission of subscribe-overprivileged fields by passively sniffing sensitive information from the overprivileged fields.

4. Problem Novelty and System Design

4.1. Overprivilege Problem Novelty

Previous work studies overprivilege at the coarser topic granularity in the publish-subscribe communication of distributed systems [55], [56], [90], [92] and identifies the loosely-coupled communication paradigm as the main cause of this overprivilege. We find that this topic-level overprivilege rarely exists in Baidu Apollo and Autoware. Instead, our work performs overprivilege detection and prevention at the message field granularity for the publish-subscribe communication model. Also, compared to previously-observed overprivilege problems in smartphone and smart home systems [67]–[69], [74], the publisher-subscribe overprivilege is novel in two aspects, creating both new design challenges and new opportunities for a practical solution. First, previous overprivilege problems occur in systems with regular user interactions like smartphone and smart home systems, where it is reasonable to rely on user judgment based on context to block unnecessary permission granting [74], [83], [93]. However, in the AV context, this is no longer acceptable since the whole design purpose is to enable autonomous driving without human input. This different usage context poses a new and more stringent design challenge for detecting and mitigating overprivilege problems without user in the loop. In §5 and §6, we detail how we address this new challenge using static analysis techniques.

Second, different from previous work on overprivilege in API accesses, the overprivilege here occurs when accessing message fields during the publish-subscribe communication at the middleware (e.g., ROS [40]) of AV systems, a layer between the upper AV software modules and underlying commodity OS. This makes it possible to mitigate overprivilege entirely in the messaging layer through instrumentation of the ROS middleware without requiring modification of existing AV software modules (*module transparency*) to achieve policy-based defense. In §6, we detail how we design and prototype such a novel solution to enforce publish-subscribe permission control at the message field level in the publish-subscribe message channel of ROS-based software systems. Our solution defends against our module-compromise threat

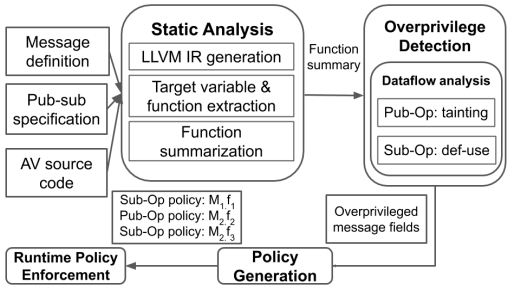

Figure 4. AVGuardian overview showing our publish-subscribe overprivilege detection and mitigation workflow. "Pub-Op" stands for publisher-side overprivilege w.r.t. a message field, "Sub-Op" stands for subscriber-side overprivilege w.r.t. a message field.

model, while the code-level policy placement approaches proposed by previous work [71], [84], [88], [95] cannot.

4.2. AVGuardian System Design

To address the overprivilege problem, we design a novel system, *AVGuardian*, to provide fine-grained control of publish-subscribe permissions in AV software systems for reducing the publish-subscribe overprivilege attack surface. Figure 4 summarizes the system design. AVGuardian takes the source code of AV modules, message definitions, and message publish-subscribe specifications (i.e., the messages each module registers to publish and subscribe) as input and achieves fine-grained publish-subscribe permission control in two major steps:

(1) Offline overprivilege detection: During AV software development, AVGuardian performs static program analysis to automatically examine each AV module's source code and detects publisher- and subscriber-side overprivilege instances in the fields of the messages defined by the module's publish-subscribe messaging specification. Our static analysis tool is designed to handle complex real-world code with common object-oriented and asynchronous programming constructs in order to achieve zero false positive (FP) and low false negative (FN) in overprivilege detection.

(2) Online fine-grained access control: At runtime once the software is deployed, for each detected publisher- or subscriber-side overprivilege instance, a fine-grained access control policy is generated and applied to the detected overprivileged message fields by online monitoring and policy enforcement. The access control is performed at the ROS middleware-level publish-subscribe messaging layer so it is *module-transparent*, meaning that no changes to the existing AV modules are required for them to be safeguarded by this access control.

In the detection phase, static data flow analyses capture use/modification behaviors for each subscribed/published message field along all possible control flows. In this process, an FP happens if a message field that is truly used/modified at runtime in the AV system is reported as unused/unmodified by our static analysis. Our system aims at achieving zero FP, because FPs will lead to under-granting permissions in the follow-up online policy enforcement and affect intended functionalities of AV systems. Note that FPs of overprivilege detection mean FNs in the dataflow analyses, i.e., some program flows in which the message fields are used/modified through some program flows are not captured by our analyses. Previous static analyzer [91], through conservative analysis of all possible control flows, provides soundness guarantee for the absence of specified security problems in Android app, i.e., zero FNs. Our static analysis design follows such conservative analysis principle in order to provide zero FNs in dataflow analysis, which translates to zero FPs in the overprivilege detection.

On the other hand, an FN in overprivilege happens if a message field that is detected as used or modified is in fact not used or modified at runtime. FN in overprivilege means FP in the dataflow analyses, and since our conservative static dataflow analyses may indeed over-approximate the use or modification behavior of certain message fields (a common problem for static analysis [77], [78], our system may have FNs in overprivilege detection. Note that even though having FNs limits the effectiveness of attack surface detection, it won't affect the correct function of AV systems.

5. Overprivilege Detection Tool

Identifying both publisher- and subscriber-side overprivilege instances at the message field granularity is a prerequisite for policy generation and runtime enforcement to mitigate overprivilege in AV software. We propose systematic detection of overprivileged message fields on the publisher and subscriber side using static analysis. Specifically, we design a static analysis tool for tracking data flow in a flow-sensitive, field-sensitive and inter-procedural manner (§5.2). In principle, our tool is capable of detecting publish-subscribe overprivilege in C++ code of general ROS-based AV systems.

Besides the **field/object-sensitivity** requirement for analyzing composite message structures, *zero* false positive is needed in the overprivilege detection to prevent removing true read/write permissions required by legitimate functionalities of an AV module. However, the extensive use of **virtual function** and **asynchronous event callback** in the complex C++ code base for AV systems [14], [17] can cause under-approximation of program behaviors and lead to false positives of our overprivilege detection. In §5.3, we propose practical solutions to address these challenges to meet above two design requirements.

5.1. Static Analysis Overview

Pre-processing: We perform static analysis on function-level control flow graphs (CFGs) generated from LLVM intermediate representation (IR) of an AV module's source code. Analysis sources and sinks are determined based on the lifecycle and event callbacks of a module. We combine CFGs of functions that can be invoked along some control flow path from an analysis source to build an inter-procedural CFG (ICFG) for inter-procedural analysis.

Subscriber-side overprivilege: The subscriber-side overprivilege problem is formulated as follows: within a module N, a field f in N's subscribed message M_s is over-granted with read permission if $M_s.f$ is never used for computation in any possible control flow path within N. We use inter-procedural, flow-, and field-sensitive define-use analysis to detect such instances (§5.2).

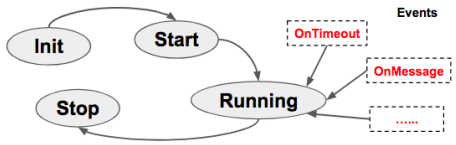

Figure 5. Lifecycle and event callbacks in an AV module

Publisher-side overprivilege: The publisher-side overprivilege problem is formulated as follows: within a module N, a field f in N's published message M_p is overgranted with write/modify permission if $M_p.f$ at publishing is never modified in any possible control flow path within N, but is directly copied from certain fields in subscribed messages by N. We use inter-procedural, flow-, and field-sensitive taint analysis to detect such instances (§5.2).

5.2. Dataflow Analysis Framework

AV module lifecycle: As illustrated in Figure 5, implementation of an AV module typically follows a predefined life cycle and usually includes multiple event callbacks for performing message subscription or periodic task processing (e.g., message publishing). In Apollo, when a module is launched, it first enters *Init* phase and then transits to *Start* phase, where key program states (e.g., publish-subscribe messaging interface) are initialized, message subscription event or periodic timer callbacks are registered and some "main" entry function is called to start actual processing. When a module is stopped, it enters the *Stop* phase to clear its program states and terminate. Based on this observation, to ensure completeness, our dataflow analysis captures all possible entry points of the execution to form analysis sources: 1) *Init* and *Start* lifecycle functions, and 2) event callback functions registered in the *Init* or *Start* function. Analysis sinks are defined using invocations of ROS's message publishing API for publisher-side overprivilege analysis or the end points of an ICFG for subscriber-side overprivilege analysis.

Inter-procedural context sensitivity: Starting from one of above entry points, an inter-procedural CFG (ICFG) is generated by expanding from the CFG of that entry point function in a recursive manner: for each callee function invoked along some control flow path and containing arguments associated to a subscribed/published message, its CFG is generated and attached to the invocation point of the caller's CFG. Our analysis performs data flow tracking on this ICFG and also jumps into the CFG of a callee function to update the dataflow information of function parameters and class variables when a function invocation is met. The inter-procedural analysis can be computationally expensive without function summarization [94]. To ensure analysis efficiency, we perform summarization of functions that invoke message variables based on the caller-callee order. The current summary of a function f consists of 3 parts: 1) a target message variable set consisting of message variables that are defined in f or passed into f, 2) variables defined in f and with a taint source from the target message variable set, 3) def-use statements for variables in 1) and 2). Once a function is summarized, the subsequent analysis will directly read the summary to update the summary of current caller function when encountering it again.

Message taint tracking: To detect use/modification of message fields along all possible control flows, we need to start from each variable storing a subscribed message instance and track the use/modification on the propagated taints for each message field across all control flows towards the program end points or where the message publishing API is invoked. A *modify* or *copy* label is assigned between the taint source and destination variable as a binary indicator whether the taint source propagates to the destination variable with modification or not. Currently, we determine which label to assigned based on the different types of LLVM instructions. Subscribed message instances may propagate across functions and class objects in an AV module. For example, a subscribed message instance may be copied into a member variable of a class object that is accessed in later execution. Also, a message instance can be passed by reference into a callee function where its fields can be propagated or modified. Thus, to support inter-procedural dataflow analysis, we summarize tainting results of a callee function that can be invoked along some control flow path and reuse this summary for the taint analysis of the caller function.

Overprivilege detection: To detect a publisher-side overprivilege, we trace the taint tracking results of each target message variable: for all control flow paths from any entry point to a sink (i.e., message publishing), if a field f of the message variable M_p to be published contains only taint source from some subscribed message field without any modify label, $M_p.f$ is a publish-overprivileged field. To detect a subscriber-side overprivilege, def-use analysis is performed on each target message variable as well as its taint sources with copy label: along all control flow paths from any entry point to any sink of an ICFG, take the union of use statements on these variables to identify a set of accessed fields (i.e., fields with true read permission) and the other fields defined in a subscribed message are subscriber-side overprivileged at the current module.

5.3. Key Analysis Challenges

Field and object sensitivity: Detecting message field overprivilege requires tracking the data flow at field granularity. As shown in Figure 3, messages defined in the publish-subscribe message model in AV systems usually consist of many fields, some of which may be of composite or recursive types. Also, a message variable can be defined as a member in a class object. We support field sensitivity with the standard technique of detecting field access of a message variable based on `getelementptr` LLVM instruction and expanding it with an offset element inferred from the operands of `getelementptr`. Depending on the levels of composition in a composite typed field, the number of field-sensitive variables to be tainted and analyzed can become very large before primitive fields are hit. A configurable depth (3 by default) is defined to limit the level of field-sensitive analysis on a message variable. Also, we observe that some message fields at certain levels are not semantically meaningful to be differentiated further (e.g., the latitude and longitude value for a GPS field in a localization message). For recursive typed message fields (e.g., list, vector, map), a configurable length is also defined to limit the iteration on elements for analysis, which is a common practice in field-

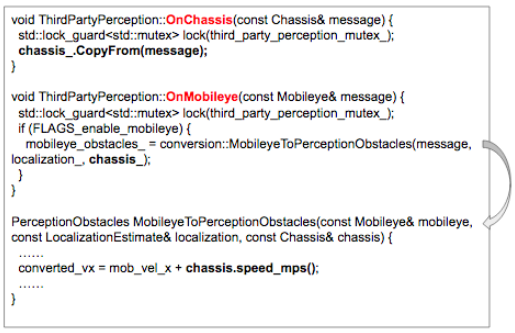

Figure 6. Order of event callbacks affects data flow tracking.

sensitive data flow analysis. This field-sensitive analysis also applies to class objects containing message variables.

Virtual function handling: Inheritance with base and derived classes in C++ are widely used in AV software to provide extensible interfaces for supporting multiple options of self-driving algorithms and vehicle models. However, this poses class binding uncertainty to our static analysis: at the LLVM IR level, calling a virtual function occurs through an indirect call and the target callee address is loaded through a variable determined at run time. To guarantee zero false positive in over-privilege detection, our analysis framework tracks data flows in virtual function calls with over-approximation by enumerating all possible derived classes. To identify all possible implementation of a virtual function defined in subclasses, we leverage LLVM's devirtualization pass to extract virtual table (`vtable`) entries and their index for each virtual function. Specifically, we detect instructions for loading a `vtable` pointer and determine which virtual function is invoked by the subsequent indirect call based on its type and accessed `vtable` index. Then data flow analysis is performed in the implemented virtual function of each possible subclass.

Asynchronous event callbacks handling: The processing of asynchronous event callbacks and their ordering depend on runtime events, while at static analysis they are independent entry points. Assuming no or a specific order, however, may cause under-approximation in data flow tracking. As illustrated in Figure 6, assuming no or different orders of callbacks may lead to different data flow tracking results: if OnChassis callback is analyzed before OnMobileye, use on **speed_mps** field in the subscribed Chassis message is captured and otherwise missed. To avoid false positive in overprivilege detection, we enumerate all possible ordering among event callbacks in a module. To implement that, we define a synthetic entry point function containing an infinite loop within which all callback functions are added sequentially and the invocation of each one is predicated on some random condition. Figure 7 shows the resulted CFG for the synthetic entry point function that handles 6 asynchronous event callbacks, where arbitrary orders among callbacks are already encoded. Therefore, our data flow tracking need not take special handling on these event callbacks.

6. Overprivilege Mitigation

We leverage the uniqueness of the overprivilege problem identified in this paper to design a publish-subscribe permission control system as an overprivilege mitigation solution. It uses the overprivilege detection results for permission control policy generation and requires no change to the existing AV modules (*module transparency*). To flexibly balance the trade-off between defense effectiveness and messaging overhead to AV's runtime performance, it supports two modes: 1) attack detection only, which runs independently of AV's decision making process and incurs zero overhead; 2) attack detection and policy enforcement for attack recovery.

6.1. Access Control Policy Design

As shown in Figure 4, given the publisher- and subscriber-side overprivilege instances detected by our static analysis, AVGuardian generates corresponding access control policies for each overprivilege instance during AV software deployment and applies low-overhead, module-transparent policy enforcement at runtime. Specifically, policies for subscriber-side overprivilege are defined based on the unused fields of a target subscriber, while policies for publisher-side overprivilege are defined based on the over-granted publisher, the fields that are over-granted write permission, and the source that was copied from. Violation of generated policies are detected online as anomaly indicators and pre-configured recovery strategies are performed. To highlight, our policy enforcement based overprivilege mitigation approach has the following key features:

- Online overprivilege policy violation detection and prevention and resilient to message replay attacks
- Low performance overhead with acceptable delay to common ROS operations (e.g., module launch, publish-subscribe communication)
- Module transparency requiring no changes to existing AV modules or additional efforts in the AV software development process for AV modules to be safeguarded by our policy enforcement
- Flexible runtime policy reconfiguration by changing a module's policy file without recompiling it

6.2. Policy Enforcement

Our threat model assumes an attacker can compromise a single module to inject and run arbitrary code (i.e., code execution in other modules are legitimate and not compromised). To ensure effective overprivilege mitigation, AVGuardian needs to enforce access control policies on published messages before they reach to a compromised subscriber, since an attacker once compromising a module can arbitrarily bypass any access control logic implemented on it. To achieve that, the policy enforcement component is designed as a shim layer connecting the publish-subscribe communication endpoints. Figure 8 is a running example on how this shim layer mitigates publisher- and subscriber-side overprivilege vulnerabiity on module B that is compromised by an attacker.

Subscriber-side overprivilege prevention: AVGuardian prevents subscriber-side overprivilege problems by proactively enforcing subscriber-side access control policies at publisher side (i.e., module A): given a message M to be published, for each subscriber S of M, the publisher clears or sets meaningless values to fields in M that are unused at

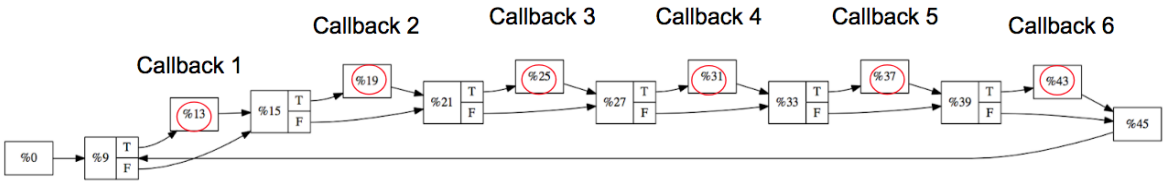

Figure 7. CFG for the synthetic entry point function, each block contains the definition of a callback. Dataflow analysis can be directly applied to these blocks.

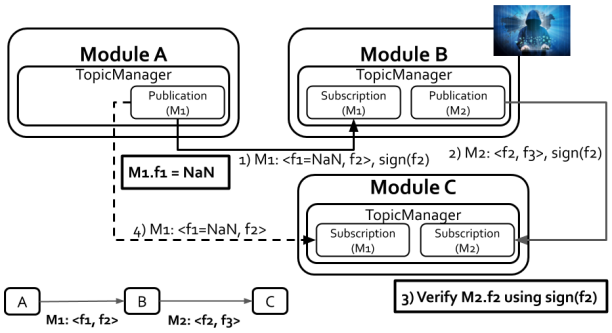

Figure 8. Detection and prevention of overprivilege policy violation. Assume module A publishes message M_1, module B (compromised by attacker) subscribes M_1 (with an overprivileged field f_1) and publishes M_2 (with an overprivileged field f_2 copied from $M_1.f_2$), module C subscribes M_2, and B is controlled by an attacker. Step 1-3 represent the message flow with policy enforcement on publisher- and subscriber-side overprivilege, where $sign(f_2)$ is the signature generated by Module A using $M_1.f_2$. Step 4 represents our proposed recovery strategy to directly contact publish originator when policy violation is detected.

S. In this case, a module will receive no more information than it actually uses when subscribing to messages from other modules. Under this design, a message to be published needs to follow access control policies defined on a per-subscriber basis. Therefore, the performance overhead becomes proportional to the number of subscribers.

Publisher-side overprivilege policy monitoring: Digital signatures are used to detect modification of values in publish-overprivileged message fields. First, from the static overprivilege detection, we can identify the original publisher (i.e., A) whose published message (M_1) is copied from by the publish-overprivileged module (B) to craft a published message (M_2), denoted as a *publish originator*. Then, by leveraging the key management feature of SROS [92], which has been integrated into ROS, a publish originator signs the source (i.e., $M_1.f_2$) of each overprivileged field before publishing a message. The publish-overprivileged module (B), if not compromised, will simply include an overprivileged field ($M_2.f_2$) without modification along with its signature into its published message. Finally, policy violations are checked at the subscriber (C) through verifying the signature of each overprivileged field to confirm that either the current value of an overprivileged field is consistent with that published by its publish originator or that a publish-overprivileged module has modified it. Using this method, AVGuardian can perform online detection if a publish-overprivileged module abuses the over-granted write permission at the message field granularity. If a policy violation is detected, AVGuardian reports this anomaly and activates recovery strategies pre-configured by AV developers.

Recovery strategies: When a publisher-side overprivilege policy violation is detected, we propose a solution, shown in Figure 8, that can recover the correct value of the overprivileged message fields with best effort and thus continue correct system operations. As shown, once the violation is detected, the subscriber (i.e., C) starts to subscribe M_1 from the publish originator (A) to obtain legitimate state for $M_2.f_2$ based on the latest $M_1.f_2$. Note that due to the asynchronous communication nature in publish-subscribe messaging, there is no guarantee that a copy of $M_2.f_2$ retrieved using $M_1.f_2$ is consistent to that from B. We make this design choice due to our observation that a module in AV software usually fetches the latest message in a subscription queue for its processing (e.g., Adapter::GetLatestObserved in Apollo).

Defense against replay attack: For publisher-side overprivilege, even with the overprivileged message fields signed by the publish originator, the attacker may still exploit the vulnerability using message replay attack, i.e., saving a signed message with values of its interest from the publish originator and replaying them later at a desired attack time. To defend against such attack, in our design we require a publish originator to sign the publish-overprivileged field with the publishing timestamp. The subscriber of a publish-overprivileged message maintains a message expiration window based on the one-way delay of an overprivileged field from its publish originator to its subscriber. When receiving an overprivileged message, it compares such window with the time difference between the current time and the publishing timestamp signed with the overprivileged field in the received message. Since it is typical that different modules in a AV system run in a single industrial PC [36], the modules share the same clock source and thus are already synchronized. If the time difference exceeds the expiration time window, potential replay attacks may be ongoing and the suspicious messages are discarded to prevent potentially malicious consequences. Furthermore, the recovery strategies above can be applied accordingly.

AV developers can configure the message expiration window by profiling one-way delays from a publish originator to subscribers for each publish-overprivileged field. In our experiments, we choose 95-percentile of our profiled delays as a threshold. While false negatives or positives in replay attack detection may occur given the variation of message transmission, queuing, and processing delay, our empirical study in §7.3 using real-world AV system traces shows that for realistic exploitation scenarios, this mechanism can effectively detect replayed messages with both *zero* false positive rate and *zero* false negative rate.

Module transparency: We implement our policy enforcement mechanism in the ROS middleware, which sits between the upper AV software modules and underly-

ing commodity OS, by instrumenting the TopicManager API [42] in the ros_comm module. TopicManager is a publish-subscribe communication protocol API in ROS and invoked by each ROS node at runtime to manage the connection among ROS nodes and forwarding of published and subscribed messages (through the Publication and Subscription component illustrated in Figure 8). We extend it to support the common operations including message field clearing, signing and verifying for our access control policy enforcement. Overprivilege access control policy files are read by the TopicManager during launch of a ROS node. Given the input policies, the instrumented TopicManager intercepts outgoing messages to be published from its hosting ROS node and incoming messages subscribed from other ROS node and apply a corresponding input policy if the intercepted message contains certain overprivileged fields. Therefore, the policy enforcement requires no change to AV software or additional efforts from AV developers.

7. Evaluation

AVGuardian's overprivilege detection tool is implemented based on LLVM. Its runtime policy enforcement is prototyped through instrumentation of the ROS middleware (ROS Indigo used by Baidu Apollo [29]). We choose Apollo for our evaluation study because it is a popular production-level AV software platform with rapid growth of users and partners [15], [24], [28], [34], [35], [46], [50], [51]. We perform runtime profiling of AV modules in Apollo using real-world and fuzzed message traces to evaluate false positives/negatives in our overprivilege detection, and also micro-benchmarking and end-to-end evaluation on performance overhead and resilience to message replay of runtime policy enforcement.

7.1. Setup

We perform overprivilege detection in Baidu Apollo 5.0 code base (∼400K LOC). It consists of 20 modules and 71 publish-subscribe message topics. As a part of the offline analysis, our tool summarizes functions of a module in Baidu Apollo in 30 minutes to 6 hours depending on the code size. The function summary can be reused later for overprivilege detection. After the summarization, the overprivilege detection within an AV module completes in up to 3 minutes. Note that this whole overprivilege analysis is an offline task and thus does not affect the runtime system performance. Also, the efficiency can be further improved by analyzing functions in parallel [94].

Ground truth identification. In the evaluation of the overprivilege detection results, we need to obtain the ground-truth unused/unmodified message fields. To more efficiently identify such ground-truth, we first use dynamic testing on target modules to automatically identify message fields that are used/modified, which can then be excluded from the ground-truth unused/unmodified message fields. Specifically, we inject messages from 4 input traces captured from Baidu's test driving of level-4 autonomous vehicles running Apollo in the real-world traffic [22] and profile the runtime execution of a target module to capture usage events on subscribed messages. To intercept such events, we instrument the protobuf library [39] (as the main API for defining publish-subscribe messages in Apollo) to record the use of message fields at runtime, which is thus the ground truth. To increase the code coverage, we also generate more diverse input messages by randomly fuzzing values of fields in different types of input messages, using the 4 real-world message traces as our seeds. In total, more than 10K different values for each field are generated.

Since dynamic testing cannot provide 100% code coverage (a fundamental coverage limitation of dynamic analysis [81], [86], [96], the message fields that are not found to be used in the dynamic testing may still be used/modified in the uncovered code portions. Thus, we then manually inspect these remaining fields to determine the ground truth. In our evaluation, the dynamic testing step helps automatically rule out 37.8% (355/939) used/modified fields, which substantially reduce the manual analysis efforts.

7.2. Accuracy of Overprivilege Detection

As shown in Table 2 and 3, comparing with the profiling results, our overprivilege detection uncovers in total 525 unused fields for subscriber-side overprivilege, and 62 unmodified fields for publisher-side overprivilege. After checking with the ground-truth ones we identified (detailed in §7.1), we did not find any FPs, which is consistent with our design goal and the corresponding conservative data flow analysis design choices (detailed in §4.2). We also break down the reduction of FPs due to analysis enhancements to address virtual function and asynchronous callback issues. By tracking data flow in more control flow paths, the virtual function enhancement reduces FNs in message field use detection (or equivalently FPs in subscriber-side overprivilege detection) by 3.87% and FNs in message field modify detection (or equivalently FPs in publisher-side overprivilege) by 8.82%. Combining with the heuristic for event callback order enumeration, the FPs of subscriber- and publisher-side overprivilege types are reduced by 4.24% and 10.3%.

Our data-flow analysis may over-approximate used/modified message fields due to conservative resolution of virtual functions and asynchronous event callbacks, leading to FPs in detecting message field use/modification, or equivalently FNs of overprivilege detection. For subscriber- and publisher-side overprivilege, an FN is observed when static analysis detects a field is used or modified, but no use or modification are observed at run time. We observe 2 FNs of subscriber-side overprivilege and 1 FN for publisher-side overprivilege. The 2 observed FNs of subscriber-side overprivilege are due to the use of a subscribed message field that can only be triggered along certain code path, which cannot be triggered due to the lack of runtime environment. From code inspection, the FN of publisher-side overprivilege is due to the definition of a published message field through an arithmetic operation taking a subscribed message field and a variable (indeed constant value) loaded from a configuration file. Our conservative analysis cannot determine the constant and mistakes this field definition as a modification. Note that these FNs cause no functional errors in policy enforcement (i.e., not blocking true read/write permission). Such low FN rate

(less than 1.7%) also demonstrates the effectiveness of our overprivilege detection in attack surface reduction.

Feature	Detected unused field #	FP rate
none	543	4.24%
+ virtual function	522	0.38% (-3.87%)
+ callback ordering	541	3.88% (-0.37%)
+ both	520	0% (-4.24%)

TABLE 2. FP IN SUBSCRIBER-SIDE OVERPRIVILEGE DETECTION.

Feature	Detected unmodified field #	FP rate
none	68	10.3%
+ virtual function	62	1.61% (-8.82%)
+ callback ordering	67	8.96% (-1.7%)
+ both	61	0% (-10.3%)

TABLE 3. FP IN PUBLISHER-SIDE OVERPRIVILEGE DETECTION.

7.3. Effectiveness of Policy Enforcement

We evaluate the performance overhead due to runtime policy enforcement in a recommended container environment on a server (Intel Xeon CPU E5-4620v2 2.60GHz, 128GB RAM) using real-world traces provided by Apollo [23]. Digital signature operations are implemented using libgcrypt [33] and 1024-bit RSA keys. Given that our container runs the same Apollo codebase installable on real vehicles and that an industrial PC with more powerful CPU configuration than that of our server is recommended for running Apollo on real vehicles [36], the following performance results (e.g., message latency) should be better on real vehicles.

Overhead analysis: Our policy enforcement requires field-wise operations on a message to be published towards a target subscriber. Table 4 shows the overhead of extra operations for enforcing a policy for publisher or subscriber-side overprivilege. The launch time for different modules is similar, around 1.58 seconds without policy enforcement and increases by 0.1% with enforcement for loading a list of policies and its key to memory. In publish-subscribe communication, a publisher may enforce subscriber-side overprivilege policies by clearing unused fields of a message to each target subscriber, and publisher-side policies by signing the source of an overprivileged field. A target subscriber verifies a signature accordingly for a publisher-side policy. Note that the batch digital signature generation can be accelerated using specialized hardware [73], [87], [89].

Source of overhead	Overhead (microsecond)
Policy parsing (100 policies)	487 ± 343
Key loading	105 ± 42
Per-field clearing	42 ± 25 (21%)
Per-field signing	1753 ± 578 (72%)
Per-field verifying	101 ± 44 (7%)

TABLE 4. BREAKDOWN OF PERFORMANCE OVERHEAD IN AVGUARDIAN'S POLICY ENFORCEMENT.

End-to-end overhead: We evaluate the end-to-end performance overhead of AV's decision cycle shown in Figure 9. The policies are generated based our overprivilege detection results on these modules. Policy enforcement happens at message publishing in each of these modules: 1) each needs to clear a few subscribe-overprivileged fields before publishing to a target subscriber and 2) Localization and Planning module as publish originators need to generate signatures for two publish-overprivilege fields in ADCTrajectory and one in ControlCommand message.

We measure the end-to-end delay from when a LocalizationEstimate message is to be published until a ControlCommand message is published by Control module

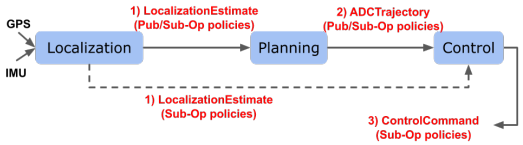

Figure 9. End-to-end policy enforcement evaluation setup

to close current control cycle. This delay without policy enforcement is on average 110 ± 3 milliseconds. With enforcement of publisher and subscriber-side overprivilege polices, this delay increases to 118 ± 9 milliseconds and 138 milliseconds in the worst case. The numbers in parentheses in Table 4 indicate the percentage each source contributes to the overall overhead. We validate that this 7.3% overhead does not affect AV decision logic, since the message sequence and order were confirmed unchanged with and without our policy enforcement. Also, FPGA accelerators available in production AVs (e.g., equipped in Apollo Extension Unit [2]) and with increasing programmable support [37], [38] can be invoked by AV software to significantly reduce the latency of digital signature operations (e.g., 2x speedup [99]).

Defense effectiveness against message replay: To evaluate our replay attack defense mechanism, we use real-world traces captured from Baidu Apollo's testing on local roads of Sunnyvale, California [23], and simulate message replay attacks by replaying the most recent publish-overprivileged field other than the up-to-date one, assuming this always leads to an exploit. This scenario puts the most stringent requirement to attack detection. We aim to identify an expiration window value that fully separates the replayed copy and the up-to-date copy based on the time gap from when the overprivileged field is sent by a publish originator to when it reaches a subscriber. We profile this time gap with and without message replay for different message topics containing publish-privileged fields that are detected from our static analysis tool.

Profiling results show that our defense mechanism can have false positives and false negative in this worst case, but the AUC [66] for all topics (listed in Table 5) are greater than 0.95, which is generally considered above good performance [98] and implies that 95% of the time our mechanism will give correct positive or negative detection of message replay. Table 5 also lists the expiration window to achieve 99% true positive (TP) or true negative (TN) rate of replay detection and their corresponding FP or FN rate in detecting replay. Results indicate that 1) the expiration window is different for different message topics to achieve 99% TP/TN rate in replay attack detection; 2) the FP/FN rate of replay attack detection when our mechanism works 99% of the time is less than 6%.

Meanwhile, we further find that for specific exploitation scenarios, since the attacker usually needs messages with specific values in the overprivileged fields to cause meaningful damages at the vehicle control level, our relay attack defense mechanism can actually achieve *zero false positive and false negative rates* on two concrete exploitation scenarios demonstrated in §8.1 and §8.2. We simulate message replay attacks using the real-world traces provided by Apollo [23] by letting the malicious node capture and replay messages with attack-desired values. For the TF attack scenario, we examine old GPS messages with

Overprivileged node	Message topic	99% TP rate		99% TN rate		AUC
		Expiration window	FP rate	Expiration window	FN rate	
Control	ControlCommand	111.2ms	1.2%	114.2ms	1.4%	0.999
GNSS TFBroadcaster	tf	10.26ms	1.2%	10.85ms	1.9%	0.995
Velodyne compensator	PointCloud	10.21ms	1.5%	10.5%	2.1%	0.99
Routing	RoutingResponse	100ms	0%	100ms	0%	1
Monitor	StaticInfo	100ms	0%	100ms	0%	1
Prediction	PredictionObstacles	10.1ms	1.5%	10.62ms	1.8%	0.988
Planning	ADCTrajectory	10.4ms	3.2%	10.95ms	3.5%	0.976
Localization	LocalizationEstimate	9.2ms	5.5%	9.8ms	5.8%	0.956
RelativeMap	MapMsg	10.3ms	3%	10.8ms	2.9%	0.977

TABLE 5. EXPIRATION WINDOWS AND ACCURACY FOR DETECTING REPLAY ATTACKS ON DIFFERENT PUBLISH-OVERPRIVILEGED TOPICS.

exploited fields set using values that can cause obstacles to be relocated out of the AV's current traffic lane and find that only messages at least 4 seconds before can cause the relocation. Given that the profiled one-way message delay from the GNSS parser nodelet to the perception module is less than 110 milliseconds, our message replay defense mechanism can detect such attack without incurring any false positives or negatives. For the PCL attack, since PointCloud messages with zero height or width values may only happen when the LiDAR sensor is broken, we cannot observe any such messages in our real-world traces and thus there are no replay attack opportunities.

8. Findings

Our tool detects 520 subscriber-side and 61 publisher-side overprivilege instances in Baidu Apollo 5.0 code base. We further identify 9 instances with important security or privacy implications (summarized in Table 6). Based on our threat model (§3) that an attacker exploits common code flaws to compromise the driver modules of the peripheral devices in an AV system and then leverage those with overprivilege vulnerability in its published/subscribed messages to launch attacks, we narrow down to the overprivileged nodes in Table 6 that are in the driver modules of peripheral devices and susceptible to be compromised based on the patches to the implementation flaws (e.g., memory bugs) [3]–[9], [11], [12] in the commit logs of Apollo's repository [14].

We are able to construct two attacks that exploit publish-overprivileged fields in the compromised driver module for the GNSS or Velodyne's LiDAR device and cause obstacle relocation (**TF attack** in §8.1) or obstacle removal (**PCL attack** in §8.2) in Apollo's object perception. Figure 10 shows these outcomes occur in the simulated traffic scenes collected on local roads of Sunnyvale, California [23] using Apollo's simulator SimControl, which results in AV's collision to the manipulated obstacles. Video demos of simulated attacks are at **https://sites.google.com/view/avguardian**. We also construct a **VIN stealing attack** that exploits a subscribe-overprivileged field in the GNSS driver module (assuming it is compromised) to steal the Vehicle Identification Number (VIN) of an AV and highlight potential identity and privacy theft to AV owners (§8.3). We performed responsible disclosure of our attacks to Apollo developer team and received confirmation that our attacks are valid.

8.1. Obstacle Relocation Attack (TF Attack)

TF attack is launched by a compromised TFBroadcaster (TransformBroadcaster) nodelet [49] in the GNSS

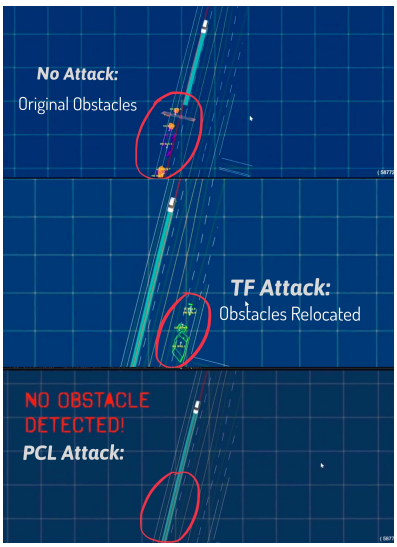

Figure 10. Comparison of attack outcomes in an identical traffic scenario: obstacle relocation (TF attack) vs. obstacle remove (PCL attack)

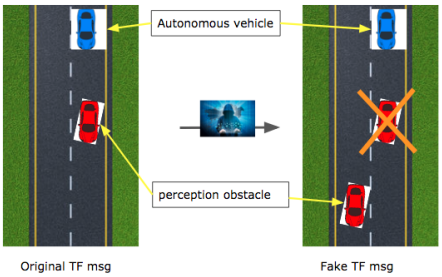

Figure 11. Relocating obstacles in AV perception by exploiting publish-overprvilege of /tf message on TFBroadcaster

driver module, which exploits a publish-overprivileged field in a published **/tf** message. AVGuardian detects over-granted write permission of tf.transform to TFBroadcaster that copies values in localization field of subscribed Gps messages to tf.transform for publishing.

Attack construction: We discover two sub-fields of tf.transform, **translation** and **rotation**, are used for the affine transformation $y = Ax + b$ to estimate the position and size of a perceived obstacle in the physical world, where A is constructed using rotation and b using translation. Specifically, translation consists of values for the x, y, and z dimension: adding an offset to certain dimension will relocate the estimated position by the same offset on that dimension. Therefore, by manipulating the translation field in a /tf message to be published, an attacker can cause the relocation of an obstacle moving ahead on the same lane to be on another lane and at a farther distance (illustrated in Figure 11). To worsen the attack outcome, an attacker can further exploit the subscribe-overprivileged field localization.linear_velocity of the

Overprivileged node	Type	Affected topic	Affected fields	AV security implication
TFBroadcaster	Pub	tf	transform.translation	Relocate perceived obstacles to cause collision (TF attack §8.1)
TFBroadcaster	Pub	tf	transform.rotation	Reduce the perceived size of obstacles
Velodyne's compensator	Pub	PointCloud	height, width	Remove perceived obstacles to cause collision (PCL attack §8.2)
Pandora's, RS-LiDAR's, LS-LiDAR's motion compensator	Pub	PointCloud	height, width	Remove perceived obstacles to cause collision
Control	Pub	ControlCommand	signal	Manipulate the on/off state of signal light
TFBroadcaster	Sub	GPS	pose.linear_velocity	Vehicle speed reconnaissance for launch TF attack (§8.1)
GNSS	Sub	Chassis	license.vin	AV owner's identity theft (VIN stealing attack §8.3)
Perception, RelativeMap, 3rd-party perception, Control	Sub	Chassis	chassis_gps	AV's location privacy leakage
Prediction, Control	Sub	Planning	debug.routing	AV's route privacy leakage

TABLE 6. SUMMARY OF OVERPRIVILEGED INSTANCES WITH SECURITY IMPLICATION. IN THE "TYPE" COLUMN, "PUB" MEANS PUBLISHER-SIDE OVERPRIVILEGE AND "SUB" MEANS SUBSCRIBER-SIDE OVERPRIVILEGE.

Gps message on TFBroadcaster (illustrated in Figure 2) to perform reconnaissance on the speed of an attacked AV and launch TF attacks when localization.linear_velocity is high. We demonstrate in SimControl with a real-world sensor trace: an attacker-controlled TFBroadcaster abuses the write-permission of tf.transform field by adding no more than 15 to values on its x and y dimension in a series (e.g., 5-second duration) of published /tf messages, which results in the obstacle relocation outcome in Figure 10 and causes a vehicle collision.

AVGuardian's defense: To mitigate TF attacks, when GPS topic is published from the parser nodelet in the GNSS module, the policy enforcement component uses the parser's private key to sign localization.position and localization.orientation field in a Gps message and appends the signatures to the published message. Before forwarding a /tf message to a target subscriber, the policy enforcement component verifies the signatures in the message using the parser's public key to detect any abuse of the write permission on tf.transform field by TFBroadcaster. If any signature verification fails, an anomaly is flagged and recovery starts by requesting for a latest Gps message from the GNSS parser.

8.2. Obstacle Removal Attack (PCL Attack)

PCL attack is launched by a compromised compensator nodelet [30] in the Velodyne's LiDAR driver module, exploiting publish-overprivileged fields in a published **PointCloud** (/apollo/sensor/velodyne64/compensator/PointCloud2) message. AVGuardian detects over-granted write permission for metadata fields width, height of PointCloud topic at the compensator. They are copied from the same metadata fields of subscribed /apollo/sensor/velodyne64/PointCloud2 messages.

Attack construction: PointCloud messages serve as point cloud input to the LiDAR processing in the perception module for detecting surrounding obstacles. LiDAR-based perception performs element-wise copy of the point cloud contents in a PointCloud message to some buffer based on its **width** and **height** dimension. By setting either of the fields as 0, LiDAR's point cloud contents are zeroed out and no obstacles will be detected (illustrated in Figure 12). We demonstrate in SimControl with a real-world sensor trace: an attacker-controlled compensator abuses the write-permission of height field by setting it as 0 in a series (e.g., 5-second duration) of published PointCloud messages, which results in the obstacle removal outcome in Figure 10 and causes a vehicle collision.

AVGuardian's defense: To mitigate PCL attacks, when /apollo/sensor/velodyne64/PointCloud2 messages are published from the converter nodelet in Velodyne driver module, the policy enforcement component uses the converter's private key to sign height and width field in each message. Before forwarding a PointCloud message to a target subscriber, the signatures in the message are verified using converter's public key and a latest /apollo/sensor/velodyne64/PoitCloud2 message is retrieved from the converter if any verification fails.

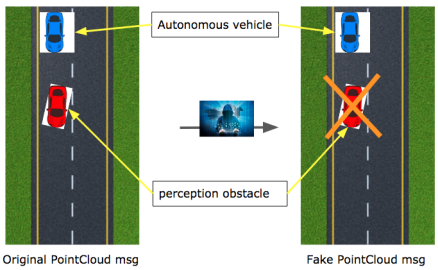

Figure 12. Removing obstacles from AV's perception view by exploiting publish-overprvilege of PointCloud message on Velodyne compensator

8.3. VIN Stealing Attack

Chassis topic has a field **license** in Apollo 3.0/3.5 and alternatively **vehicle_id** in Apollo 5.0 that contains an AV's Vehicle Identification Number (VIN) [25], [26]. AVGuardian detects that this field is set in each Chassis message published by the Canbus module, but is unused by any subscriber, including the GNSS driver module. An attacker controlling the GNSS driver module can passively sniff this subscribe-overprivileged field to steal the VIN of an attacked AV and use it to uncover personal information of the AV owner, including name, address, and even phone number and email address [53]. Furthermore, attackers can use a single stolen VIN to register dozens of stolen vehicles for masking vehicle theft or filing insurance claims on totaled vehicles, and even to make duplicate keys for an attacked AV [31]. AVGuardian's mitigation approach in §6 prevents such attacks by enforcing a subscribe-overprivilege policy at runtime to clear license or vehicle_id field of each published Chassis message.

8.4. Apollo Developer Feedback

We performed responsible disclosure to the Apollo developer team. They confirmed that our attack findings are valid under our threat model. The overprivilege problem is prevalent in Apollo: 69.4% (520/749) of subscribed message fields are indeed unused by a subscriber and 30.1% (61/203) of message fields published by the different publishers are publisher-side overprivileged. The Apollo team commented that this overprivilege is likely

due to the fact that they aim to encourage open-source contribution to Apollo and thus provide a unified and liberal message interface. They also commented that it can be highly beneficial to have a systematic approach to automatically uncover and prevent overprivilege problems, which is exactly the research goal in this paper.

9. Limitation & Scope

False negatives from overprivilege detection: Our current data flow analyses make conservative assumptions on possible control flows at run time (e.g., does not handle implicit flows) to minimize false negatives (FNs) in detecting the use/modification of message fields (or FPs in oveprivilege detection). Such over-approximation may cause FPs in detecting message field use/modification (or FNs in overprivilege detection). To mitigate defense ineffectiveness due to FNs in overprivilege detection, auto-generated access control policies can be inspected by AV developers to determine which subset to be enforced at runtime. We leave this as future work since (1) these cases are not prevalent in real-world AV code bases, e.g., only 3 FNs (less than 1.7% FN rate) observed in our evaluation (§7), and (2) though fewer FNs in overprivilege detection are desired for defense effectiveness, their existence does not block true read/write permissions in policy enforcement and thus will not affect the functionality of a system.
Potential improvements of policy enforcement: AVGuardian's policy violation detection on publisher-side overprivilege requires signing each overprivileged fields. To reduce the performance overhead proportional to the number of overprivileged fields in a message, GPUs and FPGA accelerators, already available in production AV systems [2], [36], can be used to parallelize batch operations [73], [87], [89] and speed up a single operation [99], respectively. Also, our defense against publisher-side overprivilege through contacting the publish originator, is motivated by the observation that the latest state of a message field is commonly consumed in the distributed processing of an AV system. It only provides best-effort recovery of the valid state of a publisher-side overprivileged field. To protect against stronger adversary, larger-size RSA keys (e.g., 2048-bit) can be directly configured and used in our current policy enforcement.
Threat model scope: AVGuardian is primarily a defense solution for attack surface reduction of AV systems. It defends against attacks exploiting the overprivilege at the publish-subscribe communication channel, since this overprivilege attack surface is unnecessarily exposed and can be minimized through runtime access control policy enforcement. As demonstrated by our vulnerability analysis and exploits on two popular representative AV software systems, this overprivilege attack surface is a general issue for AV systems and may cause severe safety and privacy consequences. AVGuardian does not handle other attacks, such as spoofing of non-overprivileged fields to the publish-subscribe communication channel at publisher side. A publisher, even if not compromised, must be granted write permission on those fields and AVGuardian cannot differentiate valid or spoofed states for them. Techniques to ensure computation integrity are needed to defend against this threat. Also, we do not consider scenarios that a compromised module maliciously withholds messages to be published.

10. Related Work

Vehicle security: Attack surface analysis has been conducted on in-vehicle network [59], [63], [75], vehicle applications [65], perception sensors [82], and connected vehicular communication [60]. Our work contributes to this area in discovering overprivilege problems with the publish-subscribe communication channel as a new attack surface in AV systems and proposing a systematic approach to mitigate it. One major defense solution to existing in-vehicle attacks is intrusion detection of an attacker-controlled ECU based on fingerprinting [61], [62]. We propose a different mitigation solution to the overprivilege problem through runtime policy enforcement to enable fine-grained permission control on AV software systems.
Permission models & overprivilege mitigation: Previous work uncovers and mitigates overprivilege problems in smartphone [67], [68] and smart home systems [69], [74] through static and dynamic analysis. Our overprivilege problem differs from them since (1) previous overprivilege problems happen in systems with regular user interactions while human interaction may not exist in AV systems, and (2) different from previous overprivilege in API accesses, our overprivilege is in the access to message fields for the publish-subscribe communication in AV systems. We develop a novel system-level solution to achieve fine-grained permission control on this communication channel.
Security policy enforcement: One major approach for security policy enforcement is access control APIs for developers [84]. Another is automated policy generation through static detection of security violations [88] and enforcement through OS-level authorization hook placement [71], app-level repackaging [95] or code-level instrumentation [72] to run policy enforcement code. Different from previous work on the code-level enforcement, AVGuardian enforces fine-grained access control policies from the static overprivilege detection at the publish-subscribe message channel under the module-compromise threat model, where code-level policy placement and enforcement can be bypassed.

11. Conclusion

We design and implement AVGuardian that systematically detects message field level overprivilege for the publish-subscribe communication in AV systems with zero false positive and performs policy enforcement on overprivileged fields at runtime with acceptable performance overhead to mitigate their security damage. AVGuardian discovers 581 overprivilege instances in Baidu Apollo, some leading to concrete exploits causing vehicle collision and AV owners' identity theft that have been confirmed valid by the Apollo developer team.

Acknowledgements

We thank the anonymous reviewers for their valuable comments. This work was supported in part by ONR under the grant N00014-18-1-2020 and by NSF under the grants CNS-1930041 and CNS-1544678.

References

[1] "GPSD vulnerability," https://nvd.nist.gov/vuln/detail/CVE-2018-17937, 2018.

[2] "Apollo Extension Unit (AXU)," http://apollo.auto/platform/hardware.html, 2019.

[3] "Apollo Github commit," https://github.com/ApolloAuto/apollo/commit/8bd16f986c38619fb5f4a9c6b330139dd705c96a, 2019.

[4] "Apollo Github commit," https://github.com/ApolloAuto/apollo/commit/3ca9d93a9a41e42b7d5867a96271b985b7caf603, 2019.

[5] "Apollo Github commit," https://github.com/ApolloAuto/apollo/commit/f9db5ebf699fd82261f1911b519ae16ab6874ee0, 2019.

[6] "Apollo Github commit," https://github.com/ApolloAuto/apollo/commit/39eacd9ced6e69ab6f87336e171962a98c3c5c85, 2019.

[7] "Apollo Github commit," https://github.com/ApolloAuto/apollo/commit/d8da92557d00550f2ca17bf9bd583f385dbbc67b, 2019.

[8] "Apollo Github commit," https://github.com/ApolloAuto/apollo/commit/5c13d1f5ce8c3492f912e1b76ec9538c2d64c938, 2019.

[9] "Apollo Github commit," https://github.com/ApolloAuto/apollo/commit/b80379957a72c39b1412e18b6bce1a6ccbb16eec, 2019.

[10] "Apollo Github commit," https://github.com/ApolloAuto/apollo/commit/5bca95d8d69b5a82269b4725f03a9010525b0b3f, 2019.

[11] "Apollo Github commit," https://github.com/ApolloAuto/apollo/commit/751e9ab63da7a2240dc54cb2c7c7cbde1b9a4ede, 2019.

[12] "Apollo Github commit," https://github.com/ApolloAuto/apollo/commit/c8abb2f80d7c41bfc5f542e3a55ebd2181c79e8a, 2019.

[13] "Apollo Hardware Development Platform," http://apollo.auto/platform/hardware.html, 2019.

[14] "ApolloAuto: An open autonomous driving platform," https://github.com/ApolloAuto/apollo, 2019.

[15] "Apolong: The 1st L4 Autonomous Driving Mini Bus Achieved Mass Production," http://apollo.auto/cooperation/detail_en_07.html, 2019.

[16] "Attack Video Demos for this paper," https://sites.google.com/view/avguardian, 2019.

[17] "Autoware: Open-Source To Self-Driving," https://github.com/CPFL/Autoware, 2019.

[18] "Autoware's AS package," https://github.com/CPFL/Autoware/blob/1.9.1/ros/src/actuation/vehicles/packages/as/nodes/pacmod_interface/pacmod_interface.cpp#L87, 2019.

[19] "Autoware's autoware_connector package," https://github.com/CPFL/Autoware/blob/1.9.1/ros/src/computing/perception/localization/packages/autoware_connector/nodes/can_odometry/can_odometry_core.cpp#L125, 2019.

[20] "Autoware's obj_reproj package," https://github.com/CPFL/Autoware/blob/1.9.1/ros/src/computing/perception/detection/lidar_tracker/packages/obj_reproj/nodes/obj_reproj/obj_reproj.cpp#L318, 2019.

[21] "Autoware's waypoint_follower package," https://github.com/CPFL/Autoware/blob/1.9.1/ros/src/computing/planning/motion/packages/waypoint_follower/nodes/twist_gate/twist_gate.cpp#L215, 2019.

[22] "Baidu Apollo Demo Bags," https://github.com/ApolloAuto/apollo/releases, 2019.

[23] "Baidu Apollo Sensor Data Sample," http://data.apollo.auto/help?name=data-sensor%20demostration-user%20guide&data_key=sensor&data_type=1&locale=en-us&lang=en, 2019.

[24] "Baidu's autonomous driving technology finds new application in urban cleaning," https://technode.com/2018/09/28/baidu-autonomous-driving, 2019.

[25] "Chassis message definition," https://github.com/ApolloAuto/apollo/blob/v3.5.0/modules/canbus/proto/chassis.proto, 2019.

[26] "Chassis message definition," https://github.com/ApolloAuto/apollo/blob/v5.0.0/modules/canbus/proto/chassis.proto, 2019.

[27] "Chinese researchers from Tencent discovered exploitable flaws in several BMW models," https://securityaffairs.co/wordpress/72793/hacking/bmw-models-hack.html, 2019.

[28] "CIDI equipped with Apollo 2.5 in running high-speed trucks," http://apollo.auto/cooperation/detail_en_06.html, 2019.

[29] "Collections of Apollo Platform Software," https://github.com/ApolloAuto/apollo-platform, 2019.

[30] "CompensatorNodelet," https://github.com/ApolloAuto/apollo/tree/v5.0.0/modules/drivers/velodyne/compensator, 2019.

[31] "Don't Let Identity Thieves Take You for a Ride," https://www.experian.com/blogs/ask-experian/dont-let-identity-thieves-take-you-for-a-ride, 2019.

[32] "Hackers Remotely Kill a Jeep on the Highway With Me in It," https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway, 2019.

[33] "Libgcrypt," https://gnupg.org/related_software/libgcrypt, 2019.

[34] "Mcity's Open CAVs," https://news.umich.edu/u-m-offers-open-access-automated-cars-to-advance-driverless-research, 2019.

[35] "Momenta: Apollo 1.5 Set Line Autopilot Solution," http://apollo.auto/cooperation/detail_en_03.html, 2019.

[36] "Nuvo-6108GC," https://github.com/ApolloAuto/apollo/blob/master/docs/specs/IPC/Nuvo-6108GC_Installation_Guide.md, 2019.

[37] "OPAE porting to Xilinx FPGA devices," https://github.com/RSPwFPGAs/opae-xilinx/wiki, 2019.

[38] "Open Programmable Acceleration Engine," https://opae.github.io, 2019.

[39] "Protocol Buffers," https://github.com/google/protobuf, 2019.

[40] "ROS (Robot Operating System)," http://wiki.ros.org, 2019.

[41] "ROS2 Security," https://discourse.ros.org/t/ros2-security/2273, 2019.

[42] "ros::TopicManager Class Reference," http://docs.ros.org/diamondback/api/roscpp/html/classros_1_1TopicManager.html, 2019.

[43] "Self-Driving Ubers," https://www.uber.com/cities/pittsburgh/self-driving-ubers, 2019.

[44] "SROS," http://wiki.ros.org/SROS, 2019.

[45] "SROS2: ROS2 on top of DDS-Security," https://github.com/ros2/sros2, 2019.

[46] "Suning Logistics and Baidu Apollo Partnering on Self-Driving Technology," https://www.auvsi.org/industry-news/suning-logistics-and-baidu-apollo-partnering-self-driving-technology, 2019.

[47] "Tesla car hacking," https://www.peerlyst.com/posts/attack-surface-of-cars-connected-to-each-other-and-the-internet-ben-ferris, 2019.

[48] "/tf Message," http://docs.ros.org/melodic/api/tf/html/msg/tfMessage.html, 2019.

[49] "TransformBroadcaster," https://github.com/ApolloAuto/apollo/blob/v5.0.0/modules/transform/transform_broadcaster.h, 2019.

[50] "Volkswagen taps Baidu's Apollo platform to develop self-driving cars in China," https://www.reuters.com/article/us-volkswagen-autonomous/vw-taps-baidus-apollo-platform-to-develop-self-driving-cars-in-china-idUSKCN1N
2019.

[51] "Volvo and Baidu join forces to mass produce self-driving electric cars in China," https://www.cnbc.com/2018/11/01/volvo-baidu-to-mass-produce-self-driving-electric-cars-in-china.html, 2019.

[52] "Waymo," https://waymo.com, 2019.

[53] "Your vehicle's VIN could reveal more than you want," https://www.wavy.com/10-on-your-side/investigative/special-report-privacy-breakdown/1099915487, 2019.

[54] Autoware, "Autoware Contributing," https://gitlab.com/autowarefoundation/autoware.ai/autoware/blob/master/CONTRIBUTING.md, 2019.

[55] J. Bacon, D. M. Eyers, J. Singh, and P. R. Pietzuch, "Access control in publish/subscribe systems," in *Proceedings of the 2nd International Conference on Distributed Event-based Systems*, 2008.

[56] A. Belokosztolszki, D. M. Eyers, P. R. Pietzuch, J. Bacon, and K. Moody, "Role-based Access Control for Publish/Subscribe Middleware Architectures," in *Proceedings of the 2nd International Workshop on Distributed Event-based Systems*, 2003.

[57] M. Bojarski, P. Yeres, A. Choromanska, K. Choromanski, B. Firner, L. D. Jackel, and U. Muller, "Explaining How a Deep Neural Network Trained with End-to-End Learning Steers a Car," *CoRR*, 2017.

[58] B. Breiling, B. Dieber, and P. Schartner, "Secure communication for the robot operating system," in *2017 Annual IEEE International Systems Conference*, 2017.

[59] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, and T. Kohno, "Comprehensive Experimental Analyses of Automotive Attack Surfaces," in *USENIX Security*, 2011.

[60] Q. A. Chen, Y. Yin, Y. Feng, Z. M. Mao, and H. X. L. Liu, "Exposing Congestion Attack on Emerging Connected Vehicle based Traffic Signal Control," in *NDSS*, 2018.

[61] K.-T. Cho and K.-G. Shin, "Viden: Attacker Identification on In-Vehicle Networks," in *CCS*, 2017.

[62] K.-T. Cho and K. Shin, "Fingerprinting Electronic Control Units for Vehicle Intrusion Detection," in *USENIX Security*, 2016.

[63] K.-T. Cho and K. G. Shin, "Error Handling of In-vehicle Networks Makes Them Vulnerable," in *CCS*, 2016.

[64] B. Dieber, S. Kacianka, S. Rass, and P. Schartner, "Application-level security for ROS-based applications," in *IROS*, 2016.

[65] D. Dominic, S. Chhawri, R. M. Eustice, D. Ma, and A. Weimerskirch, "Risk Assessment for Cooperative Automated Driving," in *Proceedings of the 2nd ACM Workshop on Cyber-Physical Systems Security and Privacy*, 2016.

[66] T. Fawcett, "An Introduction to ROC Analysis," *Pattern Recognition Letter*, 2006.

[67] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android Permissions Demystified," in *CCS*, 2011.

[68] A. P. Felt, K. Greenwood, and D. Wagner, "The Effectiveness of Application Permissions," in *Proceedings of the 2nd USENIX Conference on Web Application Development*, 2011.

[69] E. Fernandes, J. Jung, and A. Prakash, "Security Analysis of Emerging Smart Home Applications," in *IEEE Security & Privacy*, 2016.

[70] I. Foster, A. Prudhomme, K. Koscher, and S. Savage, "Fast and Vulnerable: A Story of Telematic Failures," in *USENIX WOOT*, 2015.

[71] V. Ganapathy, T. Jaeger, and S. Jha, "Automatic Placement of Authorization Hooks in the Linux Security Modules Framework," in *CCS*, 2005.

[72] D. K. Hong, Q. A. Chen, and Z. M. Mao, "An Initial Investigation of Protocol Customization," in *Proceedings of the 2017 Workshop on Forming an Ecosystem Around Software Transformation*, 2017.

[73] K. Jang, S. Han, S. Han, S. Moon, and K. Park, "SSLShader: Cheap SSL Acceleration with Commodity Processors," in *NSDI*, 2011.

[74] Y. J. Jia, Q. A. Chen, S. Wang, A. Rahmati, E. Fernandes, Z. M. Mao, and A. Prakash, "ContexIoT: Towards Providing Contextual Integrity to Appified IoT Platform," in *NDSS*, 2017.

[75] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage, "Experimental Security Analysis of a Modern Automobile," in *IEEE Security & Privacy*, 2010.

[76] S. Mazloom, M. Rezaeirad, A. Hunter, and D. McCoy, "A Security Analysis of an In-Vehicle Infotainment and App Platform," in *USENIX WOOT*, 2016.

[77] A. Møller and M. I. Schwartzbach, *Static Program Analysis*, 2019.

[78] F. Nielson, H. R. Nielson, and C. Hankin, *Principles of Program Analysis*, 1999.

[79] T. Nighswander, B. Ledvina, J. Diamond, R. Brumley, and D. Brumley, "GPS Software Attacks," in *CCS*, 2012.

[80] NVIDIA, "NvROS," https://www.nvidia.com/en-us/nvdocs/drive/5_1/linux/DRIVE_Linux_AGX_PDK_Development_Guide/DRIVE_Platform_Update/robot_os, 2019.

[81] H. Peng, Y. Shoshitaishvili, and M. Payer, "T-Fuzz: Fuzzing by Program Transformation," in *IEEE Security & Privacy*, 2018.

[82] J. Petit, B. Stottelaar, M. Feiri, and F. Kargl, "Remote Attacks on Automated Vehicles Sensors: Experiments on Camera and LiDAR," in *Black Hat Europe*, 2015.

[83] A. Rahmati and H. V. Madhyastha, "Context-Specific Access Control: Conforming Permissions With User Expectations," in *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices*, 2015.

[84] F. Roesner, T. Kohno, A. Moshchuk, B. Parno, H. J. Wang, and C. Cowan, "User-Driven Access Control: Rethinking Permission Granting in Modern Operating Systems," in *IEEE Security & Privacy*, 2012.

[85] F. B. Schneider, "Least Privilege and More," in *IEEE Security & Privacy*, 2003.

[86] N. Stephens, J. Grosen, C. Salls, A. Dutcher, R. Wang, J. Corbetta, Y. Shoshitaishvili, C. Kruegel, and G. Vigna, "Driller: Augmenting Fuzzing Through Selective Symbolic Execution," in *NDSS*, 2016.

[87] R. Szerwinski and T. Güneysu, "Exploiting the Power of GPUs for Asymmetric Cryptography," in *Proceedings of the 10th International Workshop on Cryptographic Hardware and Embedded Systems*, 2008.

[88] L. Tan, X. Zhang, X. Ma, W. Xiong, and Y. Zhou, "Autoises: Automatically inferring security specifications and detecting violations," in *USENIX Security*, 2008.

[89] G. Vasiliadis, E. Athanasopoulos, M. Polychronakis, and S. Ioannidis, "PixelVault: Using GPUs for Securing Cryptographic Operations," in *CCS*, 2014.

[90] C. Wang, A. Carzaniga, D. Evans, and A. Wolf, "Security Issues and Requirements for Internet-Scale Publish-Subscribe Systems," in *Proceedings of the 35th Annual Hawaii International Conference on System Sciences*, 2002.

[91] F. Wei, S. Roy, X. Ou, and Robby, "Amandroid: A Precise and General Inter-component Data Flow Analysis Framework for Security Vetting of Android Apps," in *CCS*, 2014.

[92] R. White, H. I. Christensen, and M. Quigley, "SROS: Securing ROS over the wire, in the graph, and through the kernel," *CoRR*, 2016.

[93] P. Wijesekera, A. Baokar, A. Hosseini, S. Egelman, D. Wagner, and K. Beznosov, "Android Permissions Remystified: A Field Study on Contextual Integrity," in *USENIX Security*, 2015.

[94] Y. Xie and A. Aiken, "Saturn: A Scalable Framework for Error Detection using Boolean Satisfiability," in *TOPLAS*, 2007.

[95] R. Xu, H. Saïdi, and R. Anderson, "Aurasium: Practical Policy Enforcement for Android Applications," in *USENIX Security*, 2012.

[96] I. Yun, S. Lee, M. Xu, Y. Jang, and T. Kim, "QSYM : A Practical Concolic Execution Engine Tailored for Hybrid Fuzzing," in *USENIX Security*, 2018.

[97] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, "A Survey of Autonomous Driving: Common Practices and Emerging Technologies," *CoRR*, 2019.

[98] W. Zhu, N. Zeng, and N. Wang, "Sensitivity, Specificity, Accuracy, Associated Confidence Interval and ROC Analysis with Practical SAS," in *Northeast SAS User Group Proceedings*, 2010.

[99] J. Zutter, M. Thalmaier, M. Klein, and K. Laux, "Acceleration of RSA Cryptographic Operations Using FPGA Technology," in *20th International Workshop on Database and Expert Systems Application*, 2009.