Play the Imitation Game: Model Extraction Attack against Autonomous Driving Localization

Qifan Zhang, Junjie Shen
{qifan.zhang,junjies1}@uci.edu
University of California, Irvine

Zhou Li, Qi Alfred Chen
{zhou.li,alfchen}@uci.edu
University of California, Irvine

Mingtian Tan, Zhe Zhou
{18210240176,zhouzhe}@fudan.edu.cn
Fudan University

Haipeng Zhang
zhanghp@shanghaitech.edu.cn
ShanghaiTech University

1 INTRODUCTION

In the recent decades, the advancement of technologies in machine learning, sensing, and control has elevated autonomous vehicles (AV) from ideation to reality. A growing number of AV companies have emerged and some have pushed their products to public roads. For instance, Google and Baidu have been operating self-driving taxis [24, 72] for years. Among all the components inside AV, the Autonomous Driving (AD) system is the most important piece, acting as the AV’s “brain”. The AD system commands the actuators according to the prediction of perception models.

One key component in the pipeline of AD is localization, which computes the real-time vehicle position. Ensuring the accuracy of localization is fundamental to the safety of AV, for which most of the AV companies use a complex Multi-Sensor Fusion (MSF) model [82, 94] to fuse the readings of multiple sensors. Essentially, MSF takes input from sensors like GPS, IMU and LiDAR, and runs a state estimation model, e.g., Kalman Filter (KF), to predict AV’s state, including position, heading direction, velocity, etc. As a result, the prediction made by MSF is highly robust, even in bad weather conditions or when one sensor is under attack, like GPS spoofing [32]. Yet, Shen et al. [79] showed that the integrity of a MSF model can be violated, by demonstrating a successful attack on the production-grade AD system, i.e., Baidu Apollo [7]. In the meantime, the confidentiality of a MSF has not been discussed, not to mention the demonstration of attacks and defense. Considering the importance of MSF, we study the MSF confidentiality issues.

Confidentiality of MSF models. By examining the production-grade MSF implementation, e.g., the one from Baidu Apollo, we found confidentiality is indeed a great concern. Although Apollo is an open-sourced project, the source code of MSF module is not released and cannot be decompiled into a readable format. In fact, based on our discussion with industrial partners, the parameters of MSF are considered the project’s top intellectual property, since they devoted years of hard work to tune the parameters and localization became the deciding factor for their product to outperform their competitors.

On the other hand, previous works in ML security have demonstrated model extraction attacks [19, 44, 89], which queries a black-box ML models on a remote server (e.g., public cloud), can infer the secret parameters of ML models. Given that the input to and the output from an MSF model can be observed, a natural idea is to borrow such model extraction technique to attack MSF. Yet, a few challenges prevent the direct application of the existing model extraction attacks, including the physical-world constraints to attacker’s
observations, the complexity of MSF models, and its interaction with other controller components (detailed in Section 3.3).

Our attack. To tackle these challenges in extracting MSF models, we leverage the two main insights of the AD system: 1) Though MSF models differ from ML models, their parameters can be approximated through gradient-based optimization, as their equations are derivable. 2) When the input or output is inaccessible, in particular, when the MSF’s output is only sent to AD controllers and the channel between them might not be interceptable, we can emulate derivable AD controllers and use their output for optimization. In fact, the emulated AD controllers do not need to be the exact same implementations, and the only requirement is that their performance is comparable to the ones in the target AD system.

Based on the above insights, we propose TaskMaster, a new model extraction attack against MSF models, with a set of techniques, like training unrolling, search-space reduction, and multi-stage optimization. Though our approach can be classified as system identification (SI) [57], we found none of the existing approaches directly work in our setting due to the complexity of MSF in AV and the constraints on attacker’s data access. We examine TaskMaster under three attack settings (intrusive in-AV attacker, non-intrusive in-AV attacker, and AV follower), and evaluate it on the real-world vehicle sensor traces (the KAIST Complex Urban sensor traces [46]). The evaluation shows that model extraction attack is a practical threat. As a highlight of our findings, by collecting data points within 25-second window of a targeted AV, we can train an ESKF model (a variation of KF model used by AV companies) reaching centimeter-level accuracy to the ground-truth model. Starting from the extracted model, the cost of the adversary (e.g., unethical competitors) can be greatly reduced.

Contributions. We summarize the contributions of this work as follows:

- We present the first study about the confidentiality of the AD localization models.
- To address the new challenges posed by the unique structure of MSF, we develop a new model extraction approach, TaskMaster, comprising optimization techniques tailored to the control-theory models.
- We examine TaskMaster under three attack settings with the real-world sensor traces, and our result indicates model extraction attack is feasible and could benefit unethical competitors.
- The implementation of TaskMaster is published at [5].

Ethics and disclosure. We have disclosed our findings to the developers of the Baidu Apollo team.

2 BACKGROUND

In this section, we first overview the architecture of Autonomous Driving (AD) system of an Autonomous Vehicle (AV), focusing on the localization module and its design based on Multi-Sensor Fusion (MSF). Then, we describe the most popular MSF algorithm that is based on Kalman Filter. Another AD component that is investigated in this work, AD controller, is introduced in Appendix A.

2.1 AD Localization

Generally, an AD system is composed of 3 main modules: sensor/information collector, on-board computer and actuator/command executor. Specifically, the sensor/information collector takes input from sensors like GNSS (Global Navigation Satellite System) receiver, LiDAR, IMU (Inertial Measurement Unit), camera, and communication devices like LTE/5G Antenna. The data is sent to the onboard computer to infer a model of the world. Based on the destination and route planning, the controller inside the computer generates the control vector to direct the vehicle, including three parameters: steering control, throttle control, and brake control. These 3 controlling parameters will be fed to the actuator/command executor to change the physical position of the vehicle and also affect the running state of the AD system.

In this work, we investigate the localization (or state estimation) component, which computes the real-time ego-vehicle position on the map. Localization is critical in ensuring driving safety and correctness, requiring centimeter-level accuracy [74], robustness under severe weather and road conditions, and high-fidelity under cyberattacks. A trivial solution for localization is to directly use the input from GNSS. However, the GNSS signal significantly degrades due to atmosphere delays and multi-path effect [38]. Moreover, civilian GNSS lacks signal authentication and is vulnerable under spoofing attack [87], in which the attacker can override the authentic signal with stronger power. Using LiDAR, which measures the reflection of laser light, individually is also fragile for localization, especially under poor weather conditions like rain [32]. Hence, localization based on Multi-Sensor Fusion (MSF), which fuses the input from multiple sensors like GPS, IMU, and LiDAR, has become the optimal solution so far, as it delivers much more accurate and robust result, by addressing the weakness of individual sensors [82, 94].

MSF algorithms. Among the existing MSF algorithms, Kalman Filter (KF)-based MSF [60] has gained much broader adoption, compared to the others (e.g., Particle Filter [39]). According to the survey by Shen et al. [79], out of the 18 top-tier robotics papers for 2018-2019, 14 papers adopted KF-based MSF. Baidu Apollo [7], an open-source AD system that has gained prominent buy-in from the AD industry [18] (e.g. being deployed in the self-driving taxi services in China [72]), also chooses KF-based MSF [94].

We focus on the confidentiality of the KF-based MSF model, and the ESKF (Error-State Kalman Filter) model used by AV (e.g., Baidu Apollo ESKF) is our primary target, mainly because it reaches the highest localization accuracy [79] and its implementation has been considered as a secret (detailed in Section 3.1). It is worth mentioning that our approach can be generalized to other KF-based MSF models, e.g., Extended Kalman filter (EKF).

2.2 Kalman Filter based Multi-Sensor Fusion

Kalman Filter (KF) [56], also known as linear quadratic estimation (LQE), uses prior state measurements to produce estimates of the posterior states. Equation 1 and 2 show the how a state in time $k$ is estimated from a state in time $k - 1$.

$$x_k = Fx_{k-1} + Bu_{k-1}$$
$$P_k = FP_{k-1}F^T + Q$$

(TaskMaster is a fictional character in Marvel Comics who can mimic any fighting style of a superhero.)
In particular, KF iteratively executes two phases: Prediction and Update. For Prediction (Equation 1), \( x_k \) (the predicted state at \( k \)) and \( P_k \) (the predicted covariance matrix measuring the confidence of \( x_k \)) are computed based on \( x_{k-1} \), \( P_{k-1} \) and \( u_{k-1} \) (the measurement of kinetics). For Update (Equation 2), the observations of the real-world environment (e.g., through sensors), denoted as \( z_k \), are used to refine \( x_k \) and \( P_k \), in order to reduce prediction errors. \( H_k \) is used to map the true state space (where \( x_k \) resides) into the observed space (where \( z_k \) resides). \( Q \) and \( R \) are the covariance matrix of the process noise and the covariance matrix of the observation noise. \( F \) and \( B \) represent the state-transition model and the control-input model.

Error-State Kalman Filter in AD system. We use the MSF implemented by Baidu Apollo to demonstrate how KF is applied for AD. Specifically, Baidu Apollo fuses the readings from IMU, LiDAR, and GNSS with Error-State Kalman Filter (ESKF), a variant of KF [94]. IMU measures the acceleration (\( axel_{k-1} \)) and angular velocity (\( omega_{k-1} \)) of the AV, which are used to construct the control vector \( u_{k-1} = (axel_{k-1}, omega_{k-1}) \), for Prediction phase. The predicted state \( x_k \) is a vector consisting of 16 values. It is represented as \( (pos_k, vel_k, quat_k, ba_k, bg_k) \), where \( pos_k \) represents the AV’s current location (3 elements), \( quat_k \) represents the heading direction in form of quaternion (4 elements), \( vel_k \) represents velocity (3 elements), \( ba_k \) represents accelerometer bias (3 elements), and \( bg_k \) represents gyrometer bias (3 elements). \( P_k \) is a \( 15 \times 15 \) matrix. For Update phase, the position measurements from GNSS and the position measurements and car heading measurements from LiDAR are considered as the observations \( z_k \) after data processing. When Update phase is finished, an Error-state Reset phase (\( \hat{P}_k = G \hat{P}_k G^T \)) is introduced by ESKF to reset \( \hat{P}_k \), to address the issue of observation drifting [94]. \( G \) is defined in Equation 11 of Appendix B.

In addition, Baidu Apollo uses different \( R \) and \( H \) for GNSS and LiDAR. For GNSS, we assume \( R_G \) (noise distribution injected into GNSS data) and \( H_G \) are used to replace \( R \) and \( H \) in Equation 2, changing the Update phase to Equation 12 of Appendix B. For LiDAR, the Update phase is separated into two sub-phases that use the position sensing data and pose (or yaw of the AD car) sensing data separately. The output of the first sub-phase is fed to the second sub-phase. The Update phase is changed to Equation 13 of Appendix B. To notice, Prediction phase happens whenever IMU sends new input, and Update phase happens whenever LiDAR or GNSS sends new input. Hence, Prediction and Update do not necessarily happen in turns. Figure 1 illustrates the workflow.

3 ATTACK OVERVIEW

In this section, we first describe the motivation of our adversary. Then, we describe the three scenarios that the attack could happen, differentiated by attackers’ capabilities. Finally, we overview the workflow of our attack, termed TaskMaster.

3.1 Adversary Motivation

Though the procedure and equation of KF (including ESKF) are known and it is expected that every AD system follows them in implementation, the parameters of KF can be varied among AD systems, resulting in different localization performance. According to [94], the production-grade implementation of Baidu Apollo achieves 0.054 meters accuracy, which outperforms the academic implementations by a large margin (1.17 meters for JS-MSF [81] and 1.91 meters for ETH-MSF [26]). A lot of driving data from a human driver needs to be collected and different tuning approaches have to be experimented with by professional AV engineers [79]. In fact, we reached out to one author of Baidu Apollo ESKF [94] and learnt that it takes more than 6 months for a specialized team to tune ESKF. As such, the parameters of KF are considered as “intellectual property”, and kept as a secret by the AV companies (e.g., Baidu’s leadership decides to keep the current and future versions of ESKF close-source, as we learnt from the author of [94]). We also tried to reverse-engineer the KF parameters from Apollo’s binary files but failed. The details are elaborated in Appendix C.

This work focuses on the confidentiality of four covariance matrices \( Q \), \( R_G \), \( R_L \) and \( R_H \) (process noises, GNSS noises, position observation noises, and yaw observation noises) of an ESKF model, which are explained in Section 2.2. The generalizability of the extracted parameters is explained in Appendix D.

Hacking AV to extract KF parameters. Hacking into the AD system of the targeted AV and then stealing the KF model is another unethical approach for the same attacker’s goal. A few recent works demonstrated it is feasible to exploit the vulnerabilities in Wi-Fi modules of Tesla, send messages through CAN (Controller Area Network) bus, and take control of the AD system remotely, e.g., by opening a Linux shell [68, 86]. However, due to the high investment into AV security by AV companies (e.g., Tesla puts US$1 million for bug bounty [22]), such vulnerabilities are very rare and can be quickly patched. Moreover, even if the shell is obtained by an adversary who is interested in KF parameters, the files containing KF models are very likely to be protected.
3.2 Adversary Model

We assume the adversary wants to steal the KF model of a competitor’s AV and integrate it into her own AV products to save the work for KF tuning. Instead of assuming “whitebox” access and directly extracting the model parameters (e.g., by reading the files/memory/registers containing the KF parameters), our adversary has “blackbox” access to a KF model, meaning she can observe the input and output, and use the information to infer the KF parameters of victim’s AV\(^3\). We assume 3 attack scenarios based on adversary’s capabilities, which are also summarized in Figure 2.

**AS1: Intrusive In-AV Attacker.** We assume the attacker has exclusive physical access to the targeted AV, e.g., by purchasing, renting or borrowing the AV, and the attacker is able to sniff the data transmitted within the AD system, by inserting the sniffers directly onto the paths between ECUs (Electronic Control Units). As such, the attacker is able to observe the input to ESKF \((u_{k-1}, z_k)\), and the output from ESKF \((\hat{x}_k)\). With such information, the attacker attempts to extract a victim’s AV’s KF model.

**AS2: Non-intrusive In-AV Attacker.** We assume the attackers cannot sniff the data within the AD system, but they can plug in a transceiver onto the AV’s Universal Serial Bus (USB), and use the transceiver to read the messages, including the readings of sensors (IMU, LiDAR, and GNSS) [40] which are unencrypted. Alternatively, the attacker can bring her own sensors. For example, LiBackpack [33] integrates LiDAR and GNSS at the backpack size, and mobile devices usually have IMU sensors [84]. Observation noises could be encountered, including the measurement accuracy of LiBackpack and synchronization issues of IMU on the mobile devices, which would reduce the attack accuracy. The attacker cannot observe the data between ESKF and controller, therefore she has no direct visibility into \(\hat{x}_k\). The output of the controller, termed \(y_k\), including steering, braking, and throttling, can be observed, by sniffing the command issued to those actuators. In this regard, we assume the attacker knows nothing about the controller and we do not consider controller parameters as a secret.

**AS3: AV Follower.** This scenario has the most stringent attack condition that the attacker has to be outside of the AV, e.g., by driving another car to follow the AV in close vicinity. With high-resolution sensors on her AV, including GNSS, LiDAR and camera, the attacker collects the motion traces of the victim AV, and infers the sensor readings of the victim AV \((u_{k-1}, z_k)\) and the controller output \((y_k)\), but the readings are inaccurate. We model the input to victim’s KF as the combination of the sensor readings of attacker’s AV and attacker’s measurement noises. Shen et al. [79] adopts a similar approach to model the inaccurate attacker’s readings when launching GPS spoofing against another AV on the move.

3.3 KF Model Extraction

At the high level, extracting KF model resembles extracting machine-learning (ML) models, of which the related works are surveyed in Section 7. In essence, model extraction against ML models also assumes blackbox access, though the attacker uses the prediction APIs provided by the deployed model \(O : \mathcal{X} \rightarrow \mathcal{Y}\) to issue queries (e.g., requesting classification of images) \(X \in \mathcal{X}\), and obtains the responses, including the labels \(Y \in \mathcal{Y}\), and optionally confidence scores \(S_Y\) or logits \(L_Y\). With \(X, Y\) (together with \(S_Y\) or \(L_Y\) if available), the attacker runs an extraction algorithm \(A\) and obtains an extracted model \(\hat{O}\). The extraction is considered successful, if \(\hat{O}\) matches one of the criteria [45]: 1) Functional equivalent: \(\forall x \in \mathcal{X}, \hat{O}(x) = O(x)\); 2) High fidelity: for a target distribution \(\mathcal{D}_F\) over \(\mathcal{X}\), \(Pr_{x \sim \mathcal{D}_F}[S(\hat{O}(x), O(x))]\) is maximized, where \(S\) is a similarity function; 3) High accuracy: given a true task distribution \(\mathcal{D}_A\) over \(\mathcal{X} \times \mathcal{Y}\), \(Pr_{(x,y) \sim \mathcal{D}_{A}}[\argmax(\hat{O}(x)) = y]\) is maximized. \(\hat{O}\) with high fidelity tries to replicate the decisions of \(O\), including mis-classifications, while \(\hat{O}\) with high accuracy aims to match or even exceed the accuracy of \(O\).

Following the above terminology, we aim to recover a KF model, in particular \(Q, R, R^p\) and \(R^s\), at high accuracy or high fidelity, and we focus on ESKF in this work. A variety of learning-based approaches have been developed towards this goal [19, 44, 89]. Though none of the related works investigated control-theory models, we found the learning-based approaches hold promise in addressing our problem. When considering ESKF in isolation, our task is similar to model extraction against RNN models, both in high accuracy or high fidelity, and we use ESKF as a proof of concept.

A similar research direction is system identification [57], which aims to construct the mathematical models of dynamic systems from measured input-output data. Section 7 reviews the existing methods, but we found none of them are directly applicable to the complex MSF models, in particular ESKF used by AVs.

**Challenges.** Extracting the parameters from KF models encounters prominent challenges that cannot be addressed by the existing approaches. 1) ESKF is complex, which takes the input generated by the heterogeneous sensors (GNSS, LiDAR, and IMU) at a vastly different resolution sensors on her AV, including GNSS, LiDAR and camera, the attacker collects the motion traces of the victim AV, and infers the sensor readings of the victim AV \((u_{k-1}, z_k)\) and the controller output \((y_k)\), but the readings are inaccurate. We model the input to victim’s KF as the combination of the sensor readings of attacker’s AV and attacker’s measurement noises. Shen et al. [79] adopts a similar approach to model the inaccurate attacker’s readings when launching GPS spoofing against another AV on the move.
different pace. 2) AV is not always controlled by the attacker (e.g., under AS3), and the number of traces about the targeted AV might be small. Given that the search space of the secret is not small (e.g., Q and R are 15x15, 3x3 matrices), the attacker’s search strategy has to be highly efficient. 3) When the output of ESKF (i.e., \( \hat{x} \) and \( \hat{P} \)) is not directly observable, e.g., under scenario AS2 and AS3, the data available to the model extraction is incomplete.

To address these challenges, we proposed a novel method for learning-based KF model extraction, termed TaskMaster, involving techniques like multi-stage optimization, search-space reduction and controller simulation. The details are described next.

4 ATTACK IMPLEMENTATION

The goal of the attacker is to learn an ESKF model \( \hat{O} \) that mimics the target model \( O \). In this section, we first describe how we optimize the training procedure of ESKF to learning \( \hat{O} \) in an efficient way when the ESKF output is available. Then, we describe how to train \( \hat{O} \) without the ESKF output, by emulating controllers. We summarize the symbols in Appendix E.

4.1 Extracting ESKF Alone

Under AS1, TaskMaster uses \( u_{k-1} \), \( z_{k}^{\alpha} \), \( z_{k}^{\beta} \) (ESKF input, position measurement and yaw measurement) and \( x_{k} \) (ESKF output) to train \( \hat{O} \). The attacker can directly sniff IMU output to get \( u_{k-1} \). By sniffing GNSS output, \( z_{k}^{\alpha} \) is obtained. By sniffing LiDAR locator output, \( z_{k}^{\beta} \) and \( x_{k} \) are obtained. In the end, the attacker obtains a time sequence \( T = \{ t_{1}, ..., t_{i}, ... \} \) as input, where \( t_{i} = u_{k-1}, z_{k}^{\alpha} \) or \( z_{k}^{\beta} \). For output, \( \hat{x}_{k} \) can be intercepted from the wires between ECUs within AD system, which are produced after \( t_{i} \) is processed by the ESKF. We train \( \hat{O} \) in a recurrent way. Specifically, for each round \( i \), \( \hat{O} \) uses \( t_{i} \) and the last state \( P_{i-1} \) as input, and predicts a new state \( x_{i} \) and its covariance \( P_{i} \). The same input is sent to \( O \) to generate the predicted state \( \hat{x}_{i} \) and its covariance \( P_{i}^{\hat{O}} \). Notably, \( O \) is treated as a blackbox here. The difference between the output of \( O \) and \( \hat{O} \) is leveraged to update \( \hat{O} \).

We use an optimizer penalized by the logarithmic value of Mean Squared Error (MSE) (denoted as \( L \)) between \( x_{i} \) and \( \hat{x}_{i} \), as shown in Equation 3. The difference between \( P_{i} \) and \( P_{i}^{\hat{O}} \) is not integrated because \( P_{i}^{\hat{O}} \) is an internal variable that cannot be obtained when \( O \) is considered blackbox. We compute the logarithmic MSE to make the convergence process faster.

\[
L(x, x') = \log \left( \frac{1}{N} \sum_{i=1}^{N} \left\| x_i - x'_{i} \right\|^2 \right) \tag{3}
\]

Training \( \hat{O} \) is similar as training an LSTM model at the high level, where unrolling [14] is performed on the ESKF model. Specifically, the feed-forward process calculates the series \( x_{i}^{\alpha} \) one by one (i.e., unrolled) and then calculates the loss according to Equation 3. While in the back-propagation process, the parameters of the ESKF model are only updated once according to the gradient from the loss to each variable (as if not unrolled).

With the above strategy, we train a shadow ESKF model \( \hat{O} \). Alternatively, we can train a shadow LSTM model to extract \( \hat{O} \). However, we found this approach did not work well, because some operations like pose transformation are not modeled well under LSTM, and it is hard to make the training converge with unbalanced data (e.g., IMU, LiDAR and GNSS are 50:6.5:1 in data volume).

Search-space reduction of \( Q \). Though \( Q \) is a 15x15 matrix, we found not every value has to be tuned. According to [81], \( Q \) can be described with Equation 4.

\[
\begin{align*}
V_{i} &= \sigma_{a_{i}}^{2} (\Delta t)^{2} I \\
\Theta_{i} &= \sigma_{a_{i}}^{2} (\Delta t)^{2} I \\
A_{i} &= \sigma_{a_{i}}^{2} \Delta t I \\
\Omega_{i} &= \sigma_{a_{i}}^{2} \Delta t I \\
Q &= \text{diag}(V_{i}, \Theta_{i}, A_{i}, \Omega_{i})
\end{align*}
\tag{4}
\]

where \( V_{i}, \Theta_{i}, A_{i} \) and \( \Omega_{i} \) represent velocity, quaternion/pose, accelerometer error state, and gyrometer error state. \( \Delta t \) is the difference between timestamps. \( \sigma_{a_{i}}, \sigma_{\omega_{i}}, \sigma_{a_{i}} \), and \( \sigma_{\omega_{i}} \) are the standard deviation of velocity, pose, accelerometer and gyrometer, and they are the variables to be optimized. Each of them is a scalar variable and they are located at the diagonal of the \( Q \) matrix. Hence, we limit the optimization process on the 4 variables while avoiding touching the others (they can be set to 0), reducing the variables to be optimized from 225 (15x15) to 4. In Section 5.2, we assess how this strategy impacts the attack performance.

Multi-stage optimization. Learning \( \hat{O} \) could be based on maximum likelihood estimation (MLE), which seeks a set of parameters that maximizes a likelihood function. However, MLE assumes that the output is solely dependent on the current input. When the output is also dependent on latent variables (i.e., unobserved or hidden variables), MLE does not work well [66]. Such a problem exists in ESKF: the input from sensors as well as the ESKF model states determine the output. To address the aforementioned issues, expectation maximization (EM) [66] can be performed which introduces an extra estimation step. EM has been particularly effective in learning Gaussian Mixture Model (GMM). The dataset used to train GMM consists of points generated from one or more Gaussian processes at different paces. The two steps of EM are:

- **E-Step.** Estimate the expected value for each latent variable.
- **M-Step.** Update the parameters of the model.
Anomaly filter is an optional component included by some AD like Baidu Apollo.

- **M-Step.** Optimize the parameters using maximum likelihood.

  Under EM, the initial estimation by E-step can assign random values to the latent variables. Along the iterations, the optimized model from M-step can estimate the latent parameters for existing and new data points. We adopt this idea and develop a multi-stage optimization technique for ESKF. Specifically, $Q,R_G,R_P^y,R_P^x$ are partitioned into two groups, i.e., $G_1 = \{Q\}$ and $G_2 = \{R_G,R_P^y,R_P^x\}$. Since the two groups have different frequencies and dimensions, we can choose different learning rates and decay rates. For $G_1$, parameters in $R_G,R_P^y,R_P^x$ will be treated as constant and only $Q$ will be optimized. For $G_2$, $Q$ will be constant and other parameters will be optimized. Notably, EM has been leveraged to tune KF, but it has to be adjusted under TaskMaster because we use the input and output of another blackbox KF model for optimization. In Appendix F, we summarize the whole training process.

### 4.2 Extracting ESKF from Controller Outputs

Under AS2 and AS3, the adversary has no visibility to the ground-truth output ($x_t^f$) of O, so the loss $L$ cannot be directly computed for training. On the other hand, $x_t^f$ is sent to the controller, who outputs $y_k$ (including steering, throttling, and braking) as the control signal, which is nonetheless observable. Hence, the attacker may regard the ESKF and the trailing controller as a whole, so she can train the series (ESKF + controller) with the observable ESKF input ($u_k, z_t^P, z_t^F$) and the observable controller output ($y_k$). Then she can rarely extract the ESKF O from the trained series. Notably, the attacker does not need to know what controllers are used by the victim AVs, and we do not consider the controller as a secret. In fact, the attacker can implement a trainable controller or even use an out-of-box, open-source implementation, which is quite different from the victim AVs.

### Mechanisms of the AD controllers

As described in Appendix A, Baidu Apollo uses PID controller for longitudinal control and LQR controller for lateral control (steering). When an AV receives a map and a destination point, it generates a planned trajectory consisting of a sequence of reference positions on the map ($tp^*$), and the AD controller generates corresponding control vectors to minimize the error between the current position and the reference positions. PID controller in AD generates the longitudinal control vector, based on the predicted position ($pos_k$) and velocity ($vel_k$) from the ESKF output $x_k$. The PID control vector contains the planned next position ($p_k$) and acceleration ($a_k$), which are used to derive control commands (throttle and brake). They can be computed as below:

$$p_k = K_{pp} \times \text{MinDist}(tp^*,pos_k) + K_{ip} \times \sum_{m=k-M}^{k-1} (pm)$$

$$a_k = K_{pa} \times \text{max_speed} - \text{vel}_k + K_{ia} \times \sum_{n=k-N}^{k-1} (an)$$

Where MinDist computes the minimum distance between the reference positions ($tp^*$) and the current position ($pos_k$), max_speed represents the maximum speed allowed on the AV during navigation, $M$ and $N$ are the number of positions and accelerations from the controller output in the past, $pm$ and $an$ are the corresponding positions and accelerations. $K_{pp}, K_{ip}, K_{pa}$ and $K_{ia}$ are the controller parameters. Notably, the above equation contains "Integral" (modeling the past) and "Proportional" (modeling the present), but does not contain "Derivative" (modeling the future), which is based on Apollo’s implementation.

Similar to PID controller, LQR controller generates the lateral control vector (yaw) to derive the control commands (steering), based on the planned trajectory, the past states, and the present state (from ESKF output). However, this process requires solving Discrete-time Arithmetic Riccati Equation (DARE), which cannot be implemented compatibly with gradient descending. Specifically, iterative methods have been used to obtain a numerical solution of the equation, whose process is not derivable. As such, if we simulate LQR controller after ESKF, we will not be able to derive the gradients to optimize ESKF parameters. To address this issue, we implement Stanley controller [37] and use it replace LQR controller. Stanley controller was used for lateral control during the 2005 DARPA Grand Challenge of Autonomous Robotic Ground Vehicles [23] by the Stanford team, who won the first place. It is a perfect match for our goal because its equations related to yaw computation are derivable, as shown in Equation 6. Since TaskMaster does not extract the controller parameters, using another controller of similar performance is acceptable.

$$\text{front_axle_vec} = (\cos(yaw_k + \frac{\pi}{2}), -\sin(yaw_k + \frac{\pi}{2}))^T$$

$$\text{error_front_axle} = (x_{min^*}, y_{min^*})^T \cdot \text{front_axle_vec}$$

$$\theta_e = \text{normalize_angle}(cyaw_{min^*} - yaw_k)$$

$$\theta_d = \text{arctan}(2(k \times \text{error_front_axle}, vel_k))$$

$$d_k = \theta_e + \theta_d$$

Where yaw_k is the yaw (or heading) derived from quat_k of ESKF’s output. front_axle_vec is the estimated front axle velocity, $x_{min^*}$ and $y_{min^*}$ are the x and y coordinates of the nearest position on the planned trajectory to the current position, error_front_axle represents the error to the reference states on the front axle, cyaw_{min^*} is the yaw associated with the nearest position, normalize_angle normalizes the difference between cyaw_{min^*} and yaw_k into $[-\pi, \pi]$, $\theta_d$ and $\theta_e$ are the cross track error and the heading error, and $d_k$ is the resulted yaw control vector. $k$ is the only tuning parameter and it can be optimized along with ESKF in our method.

### Extracting $\dot{O}$

In Figure 3, we illustrate how ESKF, PID controller and Stanley controller are connected for AS2 and AS3. Compared to AS1, $x_k$ is replaced by $p_k$, $d_k$ and $a_k$ (position, yaw, and acceleration). To accommodate this change, we modify the loss of Equation 3 to
Equation 7.

\[
L(p, d, a, p', d', a') = \lambda_p \log \left( \frac{1}{N} \sum_{i=1}^{N} ||p_i - p'_i||^2 \right) + \lambda_d \log \left( \frac{1}{N} \sum_{i=1}^{N} (d_i - d'_i)^2 \right) + \lambda_a \log \left( \frac{1}{N} \sum_{i=1}^{N} (a_i - a'_i)^2 \right)
\]  

(7)

Where \((p, d, a)\) and \((p', d', a')\) are the controller outputs linked to \(\hat{O}\) and \(O\). \(\lambda_p\), \(\lambda_d\) and \(\lambda_a\) are the weights for each controller loss. After empirical analysis, we found the optimization process converges faster when \(\lambda_p\) is much higher than \(\lambda_d\) and \(\lambda_a\), since values of position coordinates (~ 3e + 5) are much larger than that of yaw (~ 1e + 2) and acceleration (~ 1e + 1).

Another difference to AS1 is that \((p, d, a)\) will not be fed back to train \(\hat{O}\) thereby training becomes non-recurrent. We make such change to avoid amplifying the error to ESKF caused by the inaccurate modeling of the controllers. Training ESKF under AS2 and AS3 follow the same workflow, except the input to ESKF and the output of controllers have noises.

**Anomaly filter.** We found some AD systems add another anomaly filter between MSF and controller, when the output of MSF is too noisy to direct the controller. For instance, Baidu Apollo takes the output of ESKF \(x_k\) and corrects it with other information, before feeding it to the controllers, as shown in Figure 3. Since the source code of the anomaly filter is not released by Baidu Apollo, we introduce a Multilayer Perceptron (MLP) model to replace it, which can be trained together with ESKF. We choose MLP because the input size is small (\(16 \times 1\)). Our MLP has 5 layers, and each layer has 16 neurons. The activation function is ReLU. The MLP model is initialized by identity matrices and all zero bias.

5 EVALUATION

In this section, we evaluate how **TaskMaster** recovers ESKF models with the real-world data. To evaluate **TaskMaster** against different models, we re-implemented one ESKF model based on [81] (termed SoLa–ESKF), and obtained the blackbox ESKF model of Baidu Apollo v2.5 (termed Apollo–ESKF), which is also the major evaluation platform for AD security research (e.g., [17, 48, 83]). For Stanley and PID controllers, we re-implemented them based on [78] and [8]. Our implementation of **TaskMaster** includes 756 LoC (lines of code) for ESKF and data pre-processing, 213 LoC for controller and 348 LoC for the training process.

We first describe the experiment settings. Then, we elaborate attack results on SoLa–ESKF and Apollo–ESKF under the three attack settings. We leave the evaluation of different parameters in Appendix G. We have also conducted experiments to compare **TaskMaster** against the baseline system identification, and evaluate how **TaskMaster** can help the spoofing attack proposed in [79], but due to the page limit, the results are not reported in this version.

### 5.1 **Experiment Settings**

**Evaluation datasets.** To evaluate **TaskMaster**, we use the KAIST Complex Urban sensor traces [46] (termed KAIST hereinafter). The authors of [46] collected sensor data, including Image, LiDAR, GPS, IMU and Encoder, from the complex urban areas of four different cities, with a mapping vehicle. In total there are 31 traces (each trace corresponds to one trip of the vehicle), and 12 are in highway and 19 are in downtown. Same as [79], we selected 5 traces from them for evaluation, which are labeled as local08, local31, local07, highway06 and highway17 by KAIST, because SoLa–ESKF and Apollo–ESKF cannot achieve reliable performance on the rest. According to Section 6.2 of [79], the 5 selected traces have the smallest average MSF state uncertainty in their categories (i.e., local and highway). According to the extended version of [79], these traces all have complete sensor data (e.g., some other traces do not have complete IMU data) and provide a complete motion history.

We use the KAIST sensor data as input and feed them to the two ESKF models to obtain two sets of ESKF states output as the ground-truth. We name the dataset consisting of the ESKF states output from SoLa–ESKF as SoLa–Output. The second dataset has the ESKF states output from Apollo–ESKF, and we name it Apollo–Output.

**Evaluation metrics.** We consider two metrics to evaluate **TaskMaster**, focusing on fidelity and accuracy.

- **Difference between the parameters (PER).** Since SoLa–ESKF is implemented by us, we have the ground-truth about the secret parameters. Therefore, we compute the difference between the parameter values learnt from the evaluation datasets and the ground-truth values, which we term Parameter Error Rate (PER) and show in the equation below:

\[
\text{PER} = \frac{|\hat{\theta} - \theta|}{|\theta|}
\]

(8)

Where \(\hat{\theta}\) represents the learnt parameter of the substitutional model and \(\theta\) represents the ground truth. The matrix mean is computed on their difference. This metric evaluates the fidelity of **TaskMaster**. In real-world settings, when the ESKF model parameters are proprietary, we cannot compute this metric. So we compute PER only for SoLa–ESKF.

- **Distance between the predicted states (SER).** The ultimate goal of the adversary is to build an ESKF model of high prediction accuracy, and the parameters stolen by **TaskMaster** should serve this purpose. Hence, for each trace, we use the inferred ESKF and the ground-truth ESKF to predict the vehicle states using the sensor inputs, and compute the Root Mean-Squared Error (RMSE) between the states, termed State Error Rate (SER), with the equation below:

\[
\text{SER} = \sqrt{\frac{1}{N} \sum_{i=1}^{N} ||y_i - y'_i||^2}
\]

(9)

Where \(N\) represents the number of positions, \(y_i\) and \(y'_i\) represent the predicted states (they are vectors) by the ground-truth ESKF \(\hat{O}\) and the inferred ESKF \(\hat{O}\). Since the states can be generated by a blackbox ESKF, we evaluate against both SoLa–ESKF and Apollo–ESKF.

**Experiment parameters.** We choose Adam as the optimizer. We set the maximum number of epochs to 200 for the training process. For the learning rate, we adopt step decay [55]. For \(\eta\), the learning rate is set to 1e – 3 for the first 10 epochs and then changed to 1e – 4,
we selected 2 different traces to train each model. The extracted when extracting the secret parameters from Sola-ESKF. Since we have white-box access to it, we assess both PER (it is averaged for Q, R_G, R_V and R_H) and SER. To show the impact of the training traces, we selected 2 different traces to train each model. The extracted models are tested on all 5 traces. Table 1 summarizes the results.

<table>
<thead>
<tr>
<th>AS</th>
<th>Training</th>
<th>PER</th>
<th>Testing (SER)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>lo08</td>
</tr>
<tr>
<td>AS1</td>
<td>lo08</td>
<td>0.01347</td>
<td></td>
</tr>
<tr>
<td></td>
<td>hi17</td>
<td>0.01297</td>
<td>0.043</td>
</tr>
<tr>
<td>AS2</td>
<td>lo08</td>
<td>0.00978</td>
<td></td>
</tr>
<tr>
<td></td>
<td>hi17</td>
<td>0.00206</td>
<td>0.024</td>
</tr>
<tr>
<td>AS3E</td>
<td>lo08</td>
<td>4.5431</td>
<td></td>
</tr>
<tr>
<td></td>
<td>hi17</td>
<td>4.6247</td>
<td>1.43</td>
</tr>
<tr>
<td>AS3G</td>
<td>lo08</td>
<td>3.9916</td>
<td></td>
</tr>
<tr>
<td></td>
<td>hi17</td>
<td>4.3111</td>
<td>1.51</td>
</tr>
<tr>
<td>AS3R</td>
<td>lo08</td>
<td>5.8869</td>
<td></td>
</tr>
<tr>
<td></td>
<td>hi17</td>
<td>4.2485</td>
<td>2.12</td>
</tr>
</tbody>
</table>

Table 1: Evaluation result on Sola-ESKF. The result of each testing trace is represented as SER. PER is the same for each training trace. PER could be larger than 1 if the error between the extracted value and the ground truth is larger than the ground truth itself. AS3E, AS3G and AS3R are AS3 under Exponential, Gamma and Rayleigh noises. “lo” and “hi” are short for “local” and “highway”.

1e - 5 and finally 1e - 6 for the following 30, 70 and 100 epochs respectively. For R, the learning rate is set to 1e - 5 initially for first 10 epochs, and then 1e - 6, 1e - 7 and finally 1e - 8 after 30, 70 and 100 epochs respectively. When the anomaly filter is emulated, its parameter values in training, initialization and controllers will be evaluated in Appendix G.

Experiment environment. The training and testing processes are done on a workstation of one NVIDIA RTX 2080 Ti GPU and one AMD 5950x with 32 GB memory. The code runs on PyTorch 1.11.0 with Nvidia CUDA 11.5 support. All the data used in the experiments are in f16 format.

5.2 Extracting Sola-ESKF

In this subsection, we evaluate the effectiveness of TaskMaster when extracting the secret parameters from Sola-ESKF. Since we have white-box access to it, we assess both PER (it is averaged for Q, R_G, R_V and R_H) and SER. To show the impact of the training traces, we selected 2 different traces to train each model. The extracted models are tested on all 5 traces. Table 1 summarizes the results.

Result on AS1 (Intrusive In-AV Attacker). The 2 training traces are local08 and highway06, which represents local (roads of downtown areas) and highway navigation respectively. As their environments are vastly different, the sensor noises and the AV kinetics (positions, velocities and heading poses) also have big differences. In general, local08 has a smaller value variation than highway06. For example, both x and y coordinate of local08 traces vary in 100-meter level (between 3.511e + 5 and 3.514e + 5, 4.022e + 5 and 4.023e + 5), respectively, while x and y coordinate of highway06 traces vary in 1000-meter level (between 3.49e + 5 and 3.53e + 5, 4.02e + 5 and 4.03e + 5, respectively). For trace length, local08 has 30,704 recorded data points while highway06 has 205,375 recorded data points.

For a training trace, we select N points (set to 1000) to construct the training dataset, following this rule: from the first 15000 points, we randomly select a starting point from the points indexed in [1001, 10000], and consecutively collect the following 1000 points for training. We drop the first N points, as ESKF is in the “warm-up” stage in the beginning, and the ESKF output is less stable. The same strategy has been adopted by [79]. For each testing trace, we select the points indexed in [1001, 15000] to construct the dataset. The secret parameters of \( \hat{O} \) are all set to random values before training.

As shown in Table 1, the extracted \( \hat{O} \) is quite similar as the ground-truth \( O \): PER is 0.01347 and 0.01483 for the two traces. It indicates TaskMaster is able to learn the secret parameters at very high fidelity. SER ranges from 0.042 to 0.089 (the unit is m). Given that L4-standard AV asks for centimeter-level (0.01) RMSE, this result is satisfactory: if the targeted ESKF has reached centimeter-level RMSE, tuning \( \hat{O} \) to the same level is deemed much easier comparing to tuning from the scratch.

We also found the impact of the training traces is fairly small. The average SER resulted from local08 and highway17 are 0.0685 and 0.0458 separately. Our result also suggests the cost of attack is quite low: by just collecting 1000 data points on one trace (about 25 seconds of AV driving), the attacker can extract a high-fidelity and high-accuracy ESKF model. In Appendix G, we show that increasing N from 1000 to 5000, a small performance gain can be obtained but the training overhead also significantly increase.

Result on AS2 (Non-intrusive In-AV Attacker). Since the attacker cannot get the output of the ESKF model, but only the output of the followed controllers in this setting, the extracted \( \hat{O} \) is expected to be less accurate. We follow the same training and testing strategy as AS1 (N is also 1000 points).

Interestingly, the result shows AS2 can achieve even lower PER and SER compared to AS1, e.g., 0.013 vs 0.029 SER when training on highway17 and testing on local07. We speculate controller outputs actually enhance the training performance, as controllers also handle noises.

Result on AS3 (AV Follower). In this setting, we injected noises into the sensor input and controller output. Though noises following normal distributions are usually used in academic studies about AV security (e.g., [79]), real-world noises are often more complex. As such, we select Exponential, Gamma, and Rayleigh noises based on a survey of noises [13]. These noise models are widely used in radiation-based systems, such as X-ray, LiDAR, and MRI systems.
We define the model of the noise injection as follows.

\[
\begin{align*}
\delta(u_{k-1}) &= u_{k-1} + n_1 \\
\delta(z_k^p) &= z_k^p + n_2 \\
\delta(z_k^-) &= z_k^- + n_3 \\
\delta(y_k) &= y_k + n_4
\end{align*}
\]

(10)

where \(\delta(u_{k-1})\), \(\delta(z_k^p)\), \(\delta(z_k^-)\), and \(\delta(y_k)\) are measurements from \(u_{k-1}\), \(z_k^p\), \(z_k^-\), and \(y_k\) under the influence of noises \(n_1\), \(n_2\), \(n_3\), and \(n_4\). We show the three noise models in Appendix H.

For \(n_1\), \(n_2\), and \(n_3\), we select \(\lambda_1 = 10\) for Exponential noises, \(\alpha_1 = 0.1, \beta_1 = 2\) for Gamma noises, and \(\alpha_1 = 0.05\) for Rayleigh noises. For \(n_4\), we select \(\lambda_2 = 1\) for Exponential noises, \(\alpha_2 = 1, \beta_2 = 2\) for Gamma noises, and \(\alpha_2 = 0.5\) for Rayleigh noises. These settings are considered reasonable given that the extracted ESKF can tolerate meter-level errors and sensors can tolerate decimeter-level errors. According to Table 1, we found PER is significantly higher, however, this is expected, as the KF parameters have to be changed to tolerate the noises. SER ranges from 1.26 to 3.07, showing the impact of noises cannot be neglected. Yet, given that the academic implementations of ESKF have meter-level RMSE (see Section 3.1), our RMSE is comparable.

The impact of search space reduction. In Section 4.1, we propose a strategy to reduce the search space of \(Q\). Here we evaluate the impact of this strategy. Specifically, we implement another version of TaskMaster that optimizes the covariances within the same sensors (position, velocity, quaternion/pose, accelerometer error state and gyrometer error state). In this setting, in total 45 (5x3x3) variables are to be optimized, which is much more than the 4 variables under search space reduction. Since more variables are involved, we optimize the model for 400 epochs.

In Figure 4, we demonstrate the training loss (measured by SER) of the two settings on local08 on AS1. It turns out that, with search space reduction, the training loss reaches 0.026m after 200 epochs, while it takes 400 epochs to reach a similar level without reduction. Moreover, the training time is almost reduced to half (198.27s vs 399.93s, per epoch). This result suggests our reduction strategy significantly improves the efficiency of TaskMaster.

5.3 Extracting Apollo-ESKF

In this subsection, we report our results of extracting Apollo-ESKF, which is blackbox and might have components not modeled by us. In Table 2, we show the SER of the 5 testing traces, paired to the two training traces (local08 and highway17). We only measure SER, as we have no knowledge about the ground-truth parameters of Apollo-ESKF.

It turns out for AS1 and AS2, the error rate significantly increased. There are 6 cases with SER more than 1 in testing, e.g., training on local08 and testing on highway17 (1.18) in AS1 / local07 (1.26), local31 (1.01) and highway06 (1.03) in AS2. All the other SERs are in decimeter level. We conjecture the rise of SER is caused by the additional procedures of Apollo-ESKF not implemented by us. We also show the error distribution in Appendix I.

With noises injected in AS3, errors of attacker’s observation rise to meter-level: the average SER training on local08 under Exponential, Gamma, and Rayleigh noises rise to 1.88, 2.67, and 1.87, respectively. Interestingly, the SERs between S01a-ESKF and Apollo-ESKF are much closer compared to AS1 and AS2.

To assess whether the extracted ESKF model is useful, we conduct another experiment and the result is shown in Appendix G.

6 DISCUSSION

Generalization to other KF models. AD may use variants of KF models in localization. For example, unscented Kalman filter and Extended Kalman filter (EFK) are good candidates as they work better with non-linear systems [30].

We believe TaskMaster can also be used to extract these variants when the structure is approximately known by the attacker. As for the reasons, 1) these variants have a similar structure with ESKF (for instance, EKF differs from ESKF only in the equations deriving \(K^t\) [60]), and 2) these models are derivable. With the above reasons, the attacker can adopt a similar methodology of TaskMaster.

Limitations. 1) We did not deploy the extracted ESKF models on real AV and evaluate them on the real roads, given we have no access to the AV testing and manufacturing process. Yet, we use the real-world traces (KAIST) and target ESKF models (Apollo-ESKF),
and the result shows TaskMASTER is effective. We are discussing with the AV vendors to obtain their feedback on our study, as an indicator of the attack’s practicality. 2) When the AD of the attacker uses a set of sensors with very different settings as the targeted AD, the stolen parameters cannot be directly used. However, as described in Section 3.1, industry-grade models like Apollo-ESKF is able to achieve good performance on different AVs even without re-tuning. Hence, we believe stealing such models should be useful to attackers in most cases. 3) TaskMASTER is less effective against Apollo-ESKF comparing to So1a-ESKF. We believe the main reason is that Apollo-ESKF is more complex than So1a-ESKF, and we are unable to emulate all components surrounding Apollo-ESKF. 4) Except Baidu Apollo, we have not found another AD system to verify if obfuscation is applied on the localization module. Though Autoware [6] is another popular open-source AD system, we found it only uses LiDAR for localization by default. Though extracting binaries from an AV can help us get another ground-truth MSF, it is impossible without vulnerability exploitation. In the meantime, we will keep inquiring the AD community. We evaluated TaskMASTER against Baidu Apollo v2.5 while the latest version is v6.0. The ESKF version is upgraded from v1.0.3 to v1.0.4. We plan to test the latest version, but we expect the changes to be small.

Defense. Though there are a number of extraction attacks against ML models, recent works show defenses are possible. An example is PRADA [52], which analyses the distribution of clients’ queries and detects the ones that deviate from the normal ones. However, these works assume MLaaS (Machine-learning as a Service) settings, where the queries can be audited. This is very difficult in our setting, as the attacker’s observation cannot be blocked under AS3, and there is no need to actively query ESKF under AS1 and AS2. A partial solution, which is practical under AS1 and AS2, could be mediating the access to ECUs. Specifically, AV vendors may add “self-destruction” modules to ECUs. Once an attacker tries to break the ECU to get the output of ESKF, the ECUs can wipe out the parameters of ESKF. Additionally, AV vendors can encrypt all internal messages.

7 RELATED WORK

Security of AD. The research into the security of modern vehicles has been started a decade ago [20, 21, 54, 71], and the attack surface on wireless protocols, CAN bus, etc. were explored. The security of AD has attracted attention from the research community only recently. One direction is to study the sensors leveraged by AD, including LiDAR, IMU, perception, etc. [12, 16, 17, 27, 49, 50, 58, 83, 90, 95, 100], and defense based on physical invariants was proposed [73]. Attacks against the traditional computing architecture, like cache side-channel attacks [59] and malware attacks [47], were examined and found feasible against the software stack of AD. Regarding the security of MSF, only Shen et al. [79] demonstrated its integrity can be tampered with, while TaskMASTER looks into the confidentiality issues of MSF models.

Model Extraction. Similar to KF models, the parameters of machine-learning models, including the classic models like logistic regression, and DNN, can be considered secret. Tramer et al. proposed an equation-solving attack and a path-finding attack [89] to guess the model parameters. Wang et al. studied how the model hyper-parameters used to balance between the loss function and the regularization terms can be stolen [96]. Juuti et al. improved on the existing attacks by generating synthetic queries and proposed defenses [52]. The attack precision is further improved with semi-supervised learning [44] and active learning [19]. While prior works focused on CNN when attacking DNN, recently, the attack against RNN was demonstrated feasible [85]. But as described in Section 3.3, successful model extraction against KF has to overcome new challenges, which are addressed by the new design of TaskMASTER.

System Identification. System identification (SI) [57] builds a mathematical model for a dynamic system with statistical analysis. The related methods can be divided into 4 categories [65], but we found directly using SI to steal ESKF parameters does yield satisfactory results. 1) The Bayesian Method treats parameter update as a Bayesian update [1], but a Bayesian optimal estimation might be unrealizable [36]. 2) Maximum Likelihood carries out non-linear, gradient-based optimization to minimize the difference between model prediction and measurement [1, 11, 99], and Expectation Maximization [9, 80] attempts to avoid the non-linear optimization. However, the optimization process is time-consuming. 3) Covariance Matching runs Monte Carlo simulation and checks if the sampled statistics are internally consistent [9, 31, 64, 67]. Though faster, Covariance Matching leads to sub-optimal results. 4) Correlation Techniques assume the sequence of prediction error is zero-mean white Gaussian noise when the model is optimal, and tune the control parameters towards this criteria [61, 62, 69]. However, this assumption does not always hold. Some works have applied SI on simple KF models [10, 25, 51], but none of them are applicable to the MSF models adopted by AVs, especially ESKF. In fact, ESKF is an LTV (linear time-variant) system while KF is an LTI (linear time-invariant) system, and many properties from LTI systems do not hold in LTV systems. Recently, deep-learning models have been used for SI [2, 63] but again none of them work on ESKF.

8 CONCLUSION

In this paper, we systematically studied the confidentiality issues underlying the AD control models, in particular the ESKF model used for localization, which has been considered intellectual property by AD companies. We designed TaskMASTER, a novel optimization-based framework to infer the secret parameters by observing the input and output of an AD. Under 3 practical adversarial settings, we found TaskMASTER can achieve very high accuracy for So1a-ESKF and comparable accuracy for a complex, industry-grade model Apollo-10-ESKF. As the first study on the AD model confidentiality, we hope our findings can attract attention from AD industries and security community in addressing this new threat.

ACKNOWLEDGMENTS

We thank our shepherd Prof. Ming Li and the anonymous reviewers for their valuable suggestions. The authors from University of California, Irvine were supported by NSF DGE-2039634, CNS-1929771, CNS-2145493, and USDOT UTC Grant 69A3552047138. The author from ShanghaiTech University was supported by ShanghaiTech Start-up Fund 2018F0203-000-07.
REFERENCES


An AD controller is normally divided into 2 sub-components: a lateral controller and longitudinal controller. The lateral controller makes decisions on the angular velocity change (i.e., steering) while the longitudinal controller makes decisions on the acceleration (i.e., throttle and brake). Both the lateral controller and the longitudinal controller share the same input data, and their output, including steering, throttle and brake, forms a complete control vector.

The AD controller solves the optimal control problem in a dynamic system. As such, it uses the existing classic control algorithms. For Baidu Apollo [8], the LQR (linear-quadratic regulator) algorithm [29] is used for lateral control while the PID (proportional-integral-derivative) algorithm [77] is used for longitudinal control. LQR controller uses a cost function defined by human, in the form of a sum of the deviations of key measurements, and finds the controller settings that minimizes the cost. PID controller calculates proportional, integral, and derivative responses after reading the sensors, and sums them to compute the actuator output. LQR controller produces better response compared to PID controller, as it aims to achieve optimal control states, though at the cost of higher complexity.

B MORE DETAILS OF ESKF

$G$ is the matrix to reset the $\hat{P}_k$, defined as follows:

$$G = \begin{pmatrix} I_n & 0 & 0 \\ 0 & I_3 - \frac{1}{2} \delta \Theta \% & 0 \\ 0 & 0 & I_3 \end{pmatrix}$$

Where $I_n$ represents an $n \times n$ identity matrix and $\delta \Theta$ represents the error state of the AD car heading (in euler angles).

The Update phase is changed from Equation 2 to the equation below in ESKF.

$$K' = P_k H_G^T (H_G P_k H_G + R_G)^{-1}$$

$$\hat{x}_k = x_k + K'(z_k - H_G x_k)$$

$$\hat{P}_k = P_k - K' H_G P_k$$

$$K'' = P_k (H_L^P)^T (H_L^P P_k (H_L^P)^T + R_L^P)^{-1}$$

$$\hat{x}_k = x_k + K''(z_k^y - H_L^y x_k)$$

$$\hat{P}_k = P_k - K'' H_L^y P_k$$

Where $R_L^P, H_L^P, z_k^P$ and $R_L^y, H_L^y, z_k^y$ are related to position and yaw observations separately.

C REVERSE-ENGINEERING KF PARAMETERS

The GitHub repo [7] of Baidu Apollo embeds ESKF in a binary file 1ibllocalization.msf. so, though other parts have source code. We have attempted to reverse engineer this binary file for 5 weeks but were unable to extract its ESKF parameters. First, we try to decompile this binary, and found SIMD vectorization [53] is heavily used, which makes the decompiled code (including the pre- and post-processing) less readable. We have used reverse-engineer tools including IDA Pro [34], Snowman [98], and McSema [88] (an LLVM IR lifting tool), and all failed. Though McSema claims that AVX instructions can be handled, SIMD Instructions still cannot be decomposed. Second, we also tried binary analysis tools like Intel Pin [43] and Frida [28] to recover the secret values at the runtime, but were also unsuccessful. Regarding the information learnt from the 5 weeks’ efforts, we roughly know the execution steps of ESKF, such as IMU prediction, measurement update, outlier detection, after reading the disassembled code. We also discovered that the ESKF prediction and updates occurred asynchronously with multi-threading, which increases the difficulty for reverse engineering.

D GENERALIZABILITY OF THE EXTRACTED MODEL

Here we explain how the extracted KF parameters can be used in AVs different from the target. $R$ (including $R_G, R_L^P$ and $R_L^y$) depends on the sensors (e.g., GNSS and LiDAR). For two AVs, if their sensors are similar, their $R$ can be similar. As a supporting evidence, we collected a trace (local08) from KAIST Complex Urban [46] (described in Section 5.1), which contains the sensor input and localization results when the tested AV is driven by a human. We consider its localization results as the ground truth, and compare to the localization results generated by Baidu Apollo’s ESKF (using the sensor input from local08), in order to assess how well the ESKF can adapt to different vehicles (the default vehicle supported by Baidu Apollo is different from the KAIST vehicle) with similar sensors (e.g., their LiDAR sensors are the same). We found that the root mean squared error (RMSE) between the ground truth and Apollo’s output is only 0.074m, reaching cm-level error, suggesting the industry-grade MSF has good adaptability.

When the sensors are quite different, directly using the stolen $R$ parameters for attacker’s ESKF model might yield sub-optimal result. We consider the re-tuning of $R$ in certain circumstances as limitation (also described in Section 6), but we expect heavy re-tuning is unnecessary in most case. In fact, a few sensor providers are very popular among the car manufacturers. For instance, (1) the default LiDAR sensors supported by Baidu Apollo were manufactured by Velodyne [93], which were also integrated by AVs from Google/Alphabet [3], Ford [75, 76, 91], Toyota [41], Mercedes-Benz [15], Hyundai Mobis [42], ThroDrive [92], etc; (2) most car manufacturers purchase GNSS/IMU sensors from Novatel [35].

Aside from $R$ and $Q$, the other matrices, including $F$, $B$, $H_G$, $H_L^P$ and $H_L^y$ are determined by the vehicle kinematics and sensor measurement models, which can be obtained from textbook or tutorials [81]. Since sensors do not have lots of measurement variations — they typically measure vehicle positions in global coordinate systems such as longitude/latitude/altitude, these matrices usually will not change for different sensors.

E SYMBOL NOTATIONS

The symbols used by Section 4 is shown in Table 3.

F ALGORITHM OF AS1 ATTACK FLOW

The algorithm is shown in Algorithm 1.
Table 3: Symbols used in Section 4. "Dim" is for Dimension. "cov" is for covariance matrix. ** marks the secret to be inferred by TaskMASTER.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Dim</th>
</tr>
</thead>
<tbody>
<tr>
<td>( u_{k-1}^p )</td>
<td>IMU measurement</td>
<td>2 × 1</td>
</tr>
<tr>
<td>( z_k^p )</td>
<td>Position measurement</td>
<td>3 × 1</td>
</tr>
<tr>
<td>( z_k^y )</td>
<td>Yaw measurement</td>
<td>4 × 1</td>
</tr>
<tr>
<td>( x_k )</td>
<td>Predicted state</td>
<td>16 × 1</td>
</tr>
<tr>
<td>( P_k )</td>
<td>Predicted cov</td>
<td>15 × 15</td>
</tr>
<tr>
<td>( y_k )</td>
<td>Controller output</td>
<td>4 × 1</td>
</tr>
<tr>
<td>( p_k )</td>
<td>Position control</td>
<td>2 × 1</td>
</tr>
<tr>
<td>( a_k )</td>
<td>Acceleration control</td>
<td>1 × 1</td>
</tr>
<tr>
<td>( d_i )</td>
<td>Yaw control</td>
<td>1 × 1</td>
</tr>
<tr>
<td>( *Q )</td>
<td>Observation noise cov</td>
<td>15 × 15</td>
</tr>
<tr>
<td>( *R^G )</td>
<td>GNSS noise cov</td>
<td>3 × 3</td>
</tr>
<tr>
<td>( *R^p )</td>
<td>LiDAR position noise cov</td>
<td>3 × 3</td>
</tr>
<tr>
<td>( *R^y )</td>
<td>LiDAR yaw noise cov</td>
<td>3 × 3</td>
</tr>
<tr>
<td>( O )</td>
<td>Ground-truth model</td>
<td>-</td>
</tr>
<tr>
<td>( \hat{O} )</td>
<td>Extracted model</td>
<td>-</td>
</tr>
</tbody>
</table>

Algorithm 1: Attack workflow under AS1

Input: Inferred \( Q, R^G, R^p, R^y \) and myStates;
\( x_{k-1} \leftarrow x_0; P_{k-1} \leftarrow P_0; \) cnt \( \leftarrow 1; \)
for \( i \leftarrow 1 \) to MaxEpoch do
while cnt \( \leq N \) do
| \( t_k \) \( \leftarrow \) get(T,cnt);
if \( t_k \) is from IMU then
| \( x_k, P_k \) \( \leftarrow \) predictIMU(\( x_{k-1}, P_{k-1}, t_k \));
end
if \( t_k \) is from GNSS then
| \( x_k, P_k \) \( \leftarrow \) predictGNSS(\( x_{k-1}, P_{k-1}, t_k \));
end
if \( t_k \) is from LiDAR then
| \( x_k, P_k \) \( \leftarrow \) updateLiDAR(\( x_{k-1}, P_{k-1}, t_k \));
end
add \( x_k \) into myStates;
\( x_{k-1} \leftarrow x_k; P_{k-1} \leftarrow P_k; \) cnt \( \leftarrow \) cnt + 1;
end
Loss \( \leftarrow \) log(MSE(myStates, refStates));
\( \hat{O} \) \( \leftarrow \) optimize(\( Q, \) Loss);
\( \hat{O} \) \( \leftarrow \) optimize(\( R^G, R^p, R^y, \) Loss);
end
Return \( Q, R^G, R^p, R^y \);

G IMPACT OF PARAMETERS

The number of points for training \( (N) \). We set \( N \) to 1000 as the default setting. Here, we evaluate how TaskMASTER is influenced by different \( N \). Intuitively, a small \( N \) will introduce more errors to each state, as ESKF might not be stabilized yet. Though a large \( N \) can overcome this issue, the training process will be longer, and more storage will be consumed to store the prior states.

We vary \( N \) from 100 to 5000 and evaluate Sola-ESKF. It turns out PER and SER are significantly decreased (from 0.4377 to 0.01347 and from 0.59 to 0.03) when \( N \) is increased from 100 to 1000, as shown in Table 4. When \( N > 1000 \), PER and SER remain the same level. However, the time overhead increases greatly (from 198.27s to 254.77s per iteration) with increasing \( N \) from 1000 to 5000. The benefit brought by a larger \( N \) is diminished by the overhead it incurs, and therefore we use \( N = 1000 \) as default.

Initialization and controller parameters. We initialize the values of \( Q \) and \( R \) based on the results of existing works [81]. Here we assess the effectiveness of TaskMASTER when different initial values are chosen.

We first assess the impact on Sola-ESKF, where the ground-truth of \( Q \) and \( R \) are known. We tested with 3 different ranges: \( [1e-3, 1e-2], [1e-2, 1e-1] \) and \( [1e-1, 1e0] \) for \( Q \), and \( [1e-6, 1e-5], [1e-5, 1e-4] \) and \( [1e-4, 1e-3] \) for \( R \). In each range, we draw 50 random values in uniform distribution, and run the experiment 50 times with the same setting as Section 5.2. Table 5 compares the average PER in AS1 and AS2. As we can see, the impact is relatively small for both settings, except when \( Q \) falls in \( [1e-1, 1e0] \) and \( R \) falls in \( [1e-4, 1e-3] \). The impact by \( Q \) is larger, and we speculate this is because the ground-truth \( R \) are in smaller range \( (1e-5 \) level) compared with \( Q \) \( (1e-2 \) level).

We then assess Apollo-ESKF in AS2. As the initial values of \( Q \) and \( R \) are not far away from the values of Sola-ESKF, we can learn whether starting from another ESKF model is important to get closer to an industry-grade ESKF. The answer seems to be negative. For \( Q \) and \( R \), the same ranges are tested as the previous experiment.

The result is summarized in Table 6.

Comparison between Sola-ESKF and Apollo-ESKF on the same KAIST dataset. Here we assess how useful the extracted ESKF model would be to the attacker. In particular, we run Sola-ESKF and Apollo-ESKF on KAIST and derive SER on their output (Sola-Output and Apollo-Output), under AS2. Table 7 shows the SER of all 5 traces. The best case has 5.31 SER while the worst case has as high as 12.60 SER, and the average is 8.63. In the meantime, our extracted \( \hat{O} \) trained on local08 has 0.98 SER in average. Hence, starting from the extracted model, parameter tuning is expected to be much easier for an adversary.

Other evaluations. We have also evaluated the impacts of the weights in loss function \( (\lambda_p, \lambda_d, \) and \( \lambda_a) \) and compared PID and Stanley controllers. The results are omitted due to page limit.

H NOISE MODELS

We adapt three noise models for noise injection in AS3 based on a survey [13]. \( \lambda, \alpha, \beta \) and \( \sigma \) are model parameters.

Exponential noise. Exponential noise follows a distribution whose probability density function (PDF) is:

\[
f(x; \lambda) = \lambda e^{-\lambda x}
\] (14)

where \( \lambda > 0 \). Mean of Exponential noise is \( \frac{1}{\lambda} \), and standard deviation is also \( \frac{1}{\lambda} \).
Table 4: The impact of $N$ on AS1. The targeted model is Sola-ESKF. local08 and highway17 are for training and testing.

<table>
<thead>
<tr>
<th>$N$</th>
<th>PER</th>
<th>SER (m)</th>
<th>Training Time (s)</th>
<th>Time per Iteration (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0.4377</td>
<td>0.59</td>
<td>11762.11</td>
<td>168.03</td>
</tr>
<tr>
<td>500</td>
<td>0.1274</td>
<td>0.27</td>
<td>13270.62</td>
<td>189.58</td>
</tr>
<tr>
<td>1000</td>
<td>0.01347</td>
<td>0.03</td>
<td>13878.90</td>
<td>198.27</td>
</tr>
<tr>
<td>2000</td>
<td>0.01562</td>
<td>0.08</td>
<td>14957.63</td>
<td>213.68</td>
</tr>
<tr>
<td>3000</td>
<td>0.01285</td>
<td>0.05</td>
<td>15724.83</td>
<td>224.64</td>
</tr>
<tr>
<td>4000</td>
<td>0.02014</td>
<td>0.03</td>
<td>16765.74</td>
<td>239.51</td>
</tr>
<tr>
<td>5000</td>
<td>0.009974</td>
<td>0.04</td>
<td>17833.91</td>
<td>254.77</td>
</tr>
</tbody>
</table>

Table 5: Comparison of PER under different parameter initialization ranges in Sola-ESKF. The training trace is local08. “Original” are the values (matrices) used in previous evaluation.

<table>
<thead>
<tr>
<th>$Q$</th>
<th>$R$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[1e-3, 1e-2]$</td>
<td>$[1e-4, 1e-3]$</td>
</tr>
<tr>
<td>AS Original</td>
<td>0.01347</td>
</tr>
<tr>
<td>AS1</td>
<td>0.01220</td>
</tr>
<tr>
<td>AS2</td>
<td>0.01265</td>
</tr>
<tr>
<td></td>
<td>0.14031</td>
</tr>
<tr>
<td></td>
<td>0.03868</td>
</tr>
<tr>
<td></td>
<td>0.10514</td>
</tr>
<tr>
<td></td>
<td>0.92062</td>
</tr>
</tbody>
</table>

Table 6: Comparison of SER under different parameter initialization ranges in Apollo-ESKF in AS2. The training trace is local08 and the testing trace is highway17.

<table>
<thead>
<tr>
<th>$Q$</th>
<th>$R$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[1e-3, 1e-2]$</td>
<td>$[1e-4, 1e-3]$</td>
</tr>
<tr>
<td>AS Original</td>
<td>0.85</td>
</tr>
<tr>
<td>AS1</td>
<td>0.01220</td>
</tr>
<tr>
<td>AS2</td>
<td>0.01265</td>
</tr>
<tr>
<td></td>
<td>0.14031</td>
</tr>
<tr>
<td></td>
<td>0.03868</td>
</tr>
<tr>
<td></td>
<td>0.10514</td>
</tr>
</tbody>
</table>

Gamma noise. Gamma noise follows a distribution whose PDF is:

$$f(x; \alpha, \beta) = \frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x}$$

where $\alpha > 0$, $\beta > 0$, and $\Gamma(\cdot)$ is gamma function [4]. Mean of Gamma noise is $\frac{\alpha}{\beta}$, and standard deviation is $\frac{\sqrt{\alpha}}{\beta}$.

Rayleigh noise. Rayleigh noise follows a distribution whose PDF is:

$$f(x; \sigma) = \frac{x}{\sigma^2} e^{-x^2/2\sigma^2}$$

where $\sigma > 0$. Mean of Rayleigh noise is $\sigma \sqrt{\frac{\pi}{2}}$, and standard deviation is $\sigma \sqrt{\frac{\pi}{2}}$.

1 ERROR DISTRIBUTION BY TRACE LOCATIONS

In addition to inspecting SER on a whole trace, we also take a closer look at the error distribution on different trace locations. In Figure 5, we show the distribution of RMSE on 2D locations of local08 under AS1, and it turns out 60% locations have less than 0.95 SER, and only a small fraction (around 5%) of locations have high SER (more than 2).