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

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

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

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

Haipeng Zhang
zhanghp@shanghaitech.edu.cn
ShanghaiTech University

ABSTRACT

The security of the Autonomous Driving (AD) system has been gaining researchers' and public's attention recently. Given that AD companies have invested a huge amount of resources in developing their AD models, e.g., localization models, these models, especially their parameters, are important intellectual property and deserve strong protection.

In this work, we examine whether the confidentiality of production-grade Multi-Sensor Fusion (MSF) models, in particular, Error-State Kalman Filter (ESKF), can be stolen from an outside adversary. We propose a new model extraction attack called TaskMaster that can infer the secret ESKF parameters under black-box assumption. In essence, TaskMaster trains a substitutional ESKF model to recover the parameters, by observing the input and output to the targeted AD system. To precisely recover the parameters, we combine a set of techniques, like gradient-based optimization, search-space reduction and multi-stage optimization. The evaluation result on real-world vehicle sensor dataset shows that TaskMaster is practical. For example, with 25 seconds AD sensor data for training, the substitutional ESKF model reaches centimeter-level accuracy, comparing with the ground-truth model.

CCS CONCEPTS

• **Security and privacy → Embedded systems security**.

KEYWORDS

autonomous driving; localization; model extraction

ACM Reference Format:
Qifan Zhang, Junjie Shen, Mingtian Tan, Zhe Zhou, Zhou Li, Qi Alfred Chen, and Haipeng Zhang. 2022. Play the Imitation Game: Model Extraction Attack against Autonomous Driving Localization. In *Annual Computer Security Applications Conference (ACSAC '22), December 5–9, 2022, Austin, TX, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3564625.3567977

ACSAC '22, December 5–9, 2022, Austin, TX, USA

ACM ISBN 978-1-4503-9759-9/22/12.
https://doi.org/10.1145/3564625.3567977

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[1], 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.

[1] TaskMaster is a fictional character in Marvel Comics who can mimic any fighting style of a superhero.

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 cyber-attacks. 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 $k-1$.

$$\begin{aligned} x_k &= Fx_{k-1} + Bu_{k-1} \\ P_k &= FP_{k-1}F^\top + Q \end{aligned} \tag{1}$$

$$
\begin{aligned}
K' &= P_k H_k^\top (H_k P_k H_k^\top + R)^{-1} \\
\hat{x}_k &= x_k + K'(z_k - H_k x_k) \\
\hat{P}_k &= P_k - K' H_k P_k
\end{aligned} \tag{2}
$$

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 to $\hat{x}_k$ and $\hat{P}_k$, 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 ($accel_{k-1}$) and angular velocity ($omega_{k-1}$) of the AV, which are used to construct the control vector $u_{k-1} = (accel_{k-1}, omega_{k-1})^\top$, for Prediction phase. The predicted state x_k is a vector consisting of 16 values. It is represented as $(pos_k, vel_k, quat_k, ba_k, bg_k)^\top$, 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 15×15 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 ($\hat{P}_k = G\hat{P}_k G^\top$) 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.

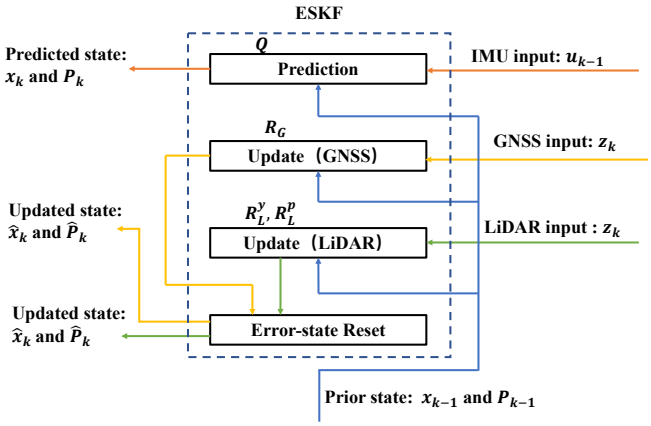

Figure 1: Workflow of ESKF. The flows of GNSS and LiDAR both generate Update states, but the values could be different.

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 by Baidu Apollo achieves 0.054 meters accuracy, which outperforms the academic KF 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^p and R_L^y (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.

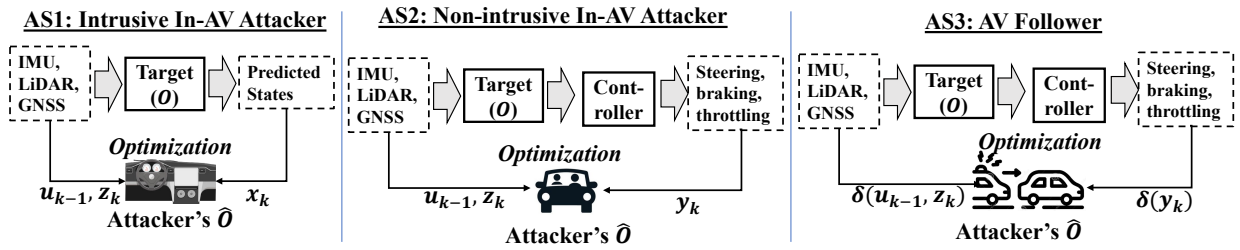

Figure 2: Adversary model. δ is the function adding noises.

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[2]. 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} and z_k), and the output from ESKF ($\hat{x}_k$). With such information, the attacker attempts to extract a victim AV's ESKF 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 accuracy 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, throttling, and braking, can be observed, by sniffing the command issued to those actuators. To notice, 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} and 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.

[2]Similar as blackbox model extraction attacks against DNN [89], the attacker needs to know the structure of KF ahead.

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 which the attacker uses the prediction APIs provided by the deployed model $O : \mathcal{X} \to \mathcal{Y}$ to issue queries (e.g., requesting classification of images) $X \subset \mathcal{X}$, and obtains the responses, including the labels $Y \subset \mathcal{Y}$, 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 $\mathcal{A}$ 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 $\mathcal{D}_{\mathcal{F}}$ over $\mathcal{X}$, $Pr_{x \sim D_F}[S(\hat{O}(x), O(x))]$ is maximized, where S is a similarity function; 3) High accuracy: given a true task distribution $\mathcal{D}_{\mathcal{A}}$ over $\mathcal{X} \times \mathcal{Y}$, $Pr_{(x,y)\sim D_A}[\text{argmax}(\hat{O}(x)) = y]$ 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_G, R_L^p and R_L^y, 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, as both RNN and ESKF have a feedback loop between output and input. As such, we can try to find the best parameters that minimize the error between the predicted states outputted by the targeted ESKF O (ground-truth) and the attacker's ESKF $\hat{O}$. Gradient-based optimization can be applied here because the ESKF functions are derivable.

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 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^p, z_k^y (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^p is obtained. By sniffing LiDAR locator output, z_k^p and z_k^y are obtained. In the end, the attacker obtains a time sequence $T = [t_1, ..., t_i, ...]$ as input, where t_i is u_{k-1}, z_k^p or z_k^y. 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 x_i' and its covariance P_i' . 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 x_i', as shown in Equation 3. The difference between P_i and P_i' is not integrated because P_i' is an internal variable that cannot be obtained when O is considered blackbox. We compute the logarithmic MSE to make the convergence process faster.

$$L(x, x') = \log(\frac{1}{N}\sum_{i=1}^{N}||x_i - x_i'||^2) \tag{3}$$

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_k' 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.

$$\begin{aligned} V_i &= \sigma_{a_n}^2 (\Delta t)^2 I \\ \Theta_i &= \sigma_{\omega_n}^2 (\Delta t)^2 I \\ A_i &= \sigma_{a_\omega}^2 \Delta t I \\ \Omega_i &= \sigma_{\omega_\omega}^2 \Delta t I \\ Q &= \mathrm{diag}(V_i, \Theta_i, A_i, \Omega_i) \end{aligned} \tag{4}$$

where V_i, Θ_i, A_i and Ω_i represent velocity, quaternion/pose, accelerometer error state, and gyrometer error state. Δt is the difference between timestamps. σ_{a_n}, σ_{ω_n}, σ_{a_ω} and σ_{ω_ω} 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.

Search-space reduction of R_G, R_L^p and R_L^y. Similar to Q, the search space of R_G, R_L^p, and R_L^y can be reduced based on the constraints of the physical world and control theory. As described in Section 2.2, R_G, R_L^p and R_L^y are covariance matrices describing the deviation of Gaussian noise injected into the related sensor observations, all of them have two properties: 1) it is a diagonal matrix, 2) elements on the diagonal of are non-negative. These properties are based on the related mathematical equations that hold universally [70, 97]. According to previous works on both GNSS and LiDAR sensor development [94], the variance of each dimension in the measurements is independent of other elements. Hence, we can fix the values of the elements not on the diagonal and add a check to avoid updating the diagonal elements to negative values. The number of elements to be optimized is reduced from 27 (R_G, R_L^p and R_L^y are all 3×3 matrices) to 9 (the diagonal elements), and the search space of each element is cut to half.

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 decide the prediction. In addition, the data generation frequencies of IMU, LiDAR, and GNSS are vastly different (roughly 50:6.5:1 on the KAIST dataset we use [46]), resulting in unstable input dimensions that cannot be easily handled by MLE.

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.

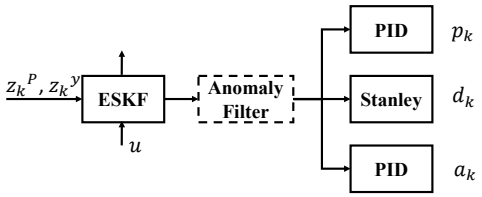

Figure 3: Workflow of ESKF combined with controllers. Anomaly filter is an optional component included by some AD like Baidu Apollo.

- **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_L^p and R_L^y are partitioned into two groups, *i.e.*, $G_1 = [Q]$ and $G_2 = [R_G, R_L^p, R_L^y]$. 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_L^p, R_L^y 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_k') of O, so the loss L cannot be directly computed for training. On the other hand, x_k' 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-1}, z_k^p, z_k^y) and the observable controller output(y_k). Then she can readily extract the ESKF O from the trained series. Noticeably, 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:

$$\begin{aligned} p_k &= K_{pp} \times \text{MinDist}(tp^*, pos_k) + K_{ip} \times \sum_{m=k-M}^{k-1} (p_m) \\ a_k &= K_{pa} \times (max\_speed - vel_k) + K_{ia} \times \sum_{n=k-N}^{k-1} (a_n) \end{aligned} \tag{5}$$

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, p_m and a_n 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.

$$\begin{aligned} front\_axle\_vec &= (\cos(yaw_k + \frac{\pi}{2}), -\sin(yaw_k + \frac{\pi}{2}))^\top \\ error\_front\_axle &= (x_{min^*}, y_{min^*})^\top \cdot front\_axle\_vec \\ \theta_e &= \text{normalize\_angle}(cyaw_{min^*} - yaw_k) \\ \theta_d &= \arctan 2(k \times error\_front\_axle, vel_k) \\ d_k &= \theta_e + \theta_d \end{aligned} \tag{6}$$

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 $[-\pi, \pi]$, θ_d and θ_e are the cross track error and the heading error, and d_i is the resulted yaw control vector. k is the only tuning parameter and it can be optimized along with ESKF in our method.

Extracting $\hat{O}$. 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.

$$L(p,d,a,p',d',a') = \lambda_p \log(\frac{1}{N}\sum_{i=1}^{N}||p_i - p_i'||^2) + \lambda_d \log(\frac{1}{N}\sum_{i=1}^{N}(d_i - d_i')^2) + \lambda_a \log(\frac{1}{N}\sum_{i=1}^{N}(a_i - a_i')^2) \quad (7)$$

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

Another difference to AS1 is that $\langle p,d,a\rangle$ 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 (x_k is 16x1). 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 the 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:

$$PER = \frac{\overline{|\hat{\theta} - \theta|}}{\overline{|\theta|}} \quad (8)$$

  Where $\hat{\theta}$ represents the learnt parameter of the substitutional model and θ 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:

$$SER = \sqrt{\frac{1}{N}\sum_{i=1}^{N}||y_i - y_i'||^2} \quad (9)$$

  Where N represents the number of positions, y_i' and y_i represent the predicted states (they are vectors) by the ground-truth ESKF 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 Q, the learning rate is set to $1e-3$ for the first 10 epochs and then changed to $1e-4$,

AS	Training	PER	Testing (SER)				
			lo08	*lo07*	*lo31*	*hi06*	*hi17*
AS1	*lo08*	0.01347	-	0.067	0.042	0.076	0.089
	hi17	0.01297	0.043	0.029	0.073	0.038	-
AS2	*lo08*	0.00978	-	0.018	0.028	0.026	0.036
	hi17	0.00206	0.024	0.013	0.018	0.036	-
AS3E	*lo08*	4.5431	-	1.76	1.24	1.33	3.15
	hi17	4.6247	1.43	2.88	2.28	4.19	-
AS3G	*lo08*	3.9916	-	1.53	1.89	2.57	1.21
	hi17	4.3111	1.51	0.58	1.43	5.67	-
AS3R	*lo08*	5.8869	-	4.24	2.10	1.11	1.33
	hi17	4.2485	2.12	1.87	4.11	0.88	-

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 replacement MLP also needs to be trained, and we set the learning rate to $1e-9$ initially and then decreased to $1e-10$ and $1e-11$ after 30 and 80 epochs respectively.

We select a subset of one trace (its positions and corresponding ESKF output) for training, because 1) using the whole trace is time-consuming (it contains hundreds of thousands of samples), and 2) the initialization stage (usually the first 40 seconds) of an MSF module yields unreliable output which should be removed. All the other traces are used for testing (the first 40 seconds' data are removed similarly). We term the number of selected positions for training as N, and set it to 1000 after empirical analysis. Other parameters for training, including the three loss weights of Equation 7 (λ_p, λ_a and λ_d) are set to 100, 1, and 0.1 respectively. The impact of different 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 `float64` 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_L^p and R_L^y) 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.

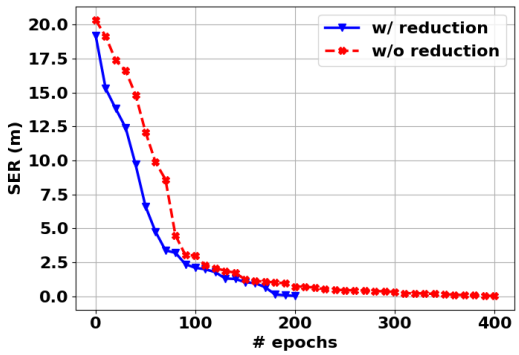

Figure 4: Training loss with and without search space reduction. The training trace is *local08*. The number of maximum training epochs is 400. We stop the training with space reduction after 200 epochs as the loss is sufficiently low.

We define the model of the noise injection as follows.

$$\begin{aligned} \delta(u_{k-1}) &= u_{k-1} + n_1 \\ \delta(z_k^p) &= z_k^p + n_2 \\ \delta(z_k^y) &= z_k^y + n_3 \\ \delta(y_k) &= y_k + n_4 \end{aligned} \tag{10}$$

where $\delta(u_{k-1})$, $\delta(z_k^p)$, $\delta(z_k^y)$ and $\delta(y_k)$ are measurements from u_{k-1}, z_k^p, z_k^y 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 $\sigma_1 = 0.05$ for Rayleigh noises. For n_4, we select $\lambda_2 = 1$ for Exponential noises, $\alpha_2 = 1, \beta_2 = 2$ for Gamma noises, and $\sigma_2 = 0.5$ 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.

AS	Training	Testing (SER)				
		lo08	*lo07*	*lo31*	*hi06*	*hi17*
AS1	*lo08*	-	0.37	0.42	0.91	1.18
	hi17	0.95	0.72	0.68	0.89	-
AS2	*lo08*	-	1.26	1.01	1.03	0.62
	hi17	1.12	0.96	0.88	0.79	-
AS3E	*lo08*	-	1.72	1.82	2.03	1.96
	hi17	1.92	1.48	2.41	1.27	-
AS3G	*lo08*	-	1.45	1.63	2.11	3.07
	hi17	1.26	1.43	1.55	1.90	-
AS3R	*lo08*	-	1.78	2.08	2.13	1.49
	hi17	1.71	1.82	1.56	1.33	-

Table 2: Evaluation result on `Apollo-ESKF`. The result in each cell is represented as SER, as `Apollo-ESKF` is blackbox.

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 speculate 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 `Sola-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 (EKF) 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 x_k, P_k and K' [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 `Sola-ESKF`. We believe the main reason is that `Apollo-ESKF` is more complex than `Sola-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 `Sola-ESKF` and comparable accuracy for a complex, industry-grade model `Apollo-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

[1] D Alspach. 1974. A parallel filtering algorithm for linear systems with unknown time varying noise statistics. *IEEE Trans. Automat. Control* 19, 5 (1974), 552–556.

[2] Brilian Putra Amiruddin, Eka Iskandar, Ali Fatoni, and Ari Santoso. 2020. Deep Learning based System Identification of Quadcopter Unmanned Aerial Vehicle. In *2020 3rd International Conference on Information and Communications Technology (ICOIACT)*. IEEE, 165–169.

[3] ars technica. 2017. Google's Waymo invests in LIDAR technology, cuts costs by 90 percent. https://arstechnica.com/cars/2017/01/googles-waymo-invests-in-lidar-technology-cuts-costs-by-90-percent/. (2017).

[4] Emil Artin. 2015. *The gamma function*. Courier Dover Publications.

[5] TaskMaster authors. 2022. Code repository for TaskMaster. https://github.com/qifan-sailboat/eskf-acsac22-ae. (2022).

[6] Autoware. 2020. Autoware-AI/autoware.ai: Open-source software for self-driving vehicles. https://github.com/Autoware-AI/autoware.ai. (2020).

[7] Baidu. 2020. ApolloAuto/apollo: An open autonomous driving platform. https://github.com/ApolloAuto/apollo. (2020).

[8] Baidu. 2020. Baidu apollo controller module. https://github.com/ApolloAuto/apollo/tree/r2.5.0/modules/control. (2020).

[9] Vinay A Bavdekar, Anjali P Deshpande, and Sachin C Patwardhan. 2011. Identification of process and measurement noise covariance for state and parameter estimation using extended Kalman filter. *Journal of Process control* 21, 4 (2011), 585–601.

[10] Matt C Best, Andrew P Newton, and Simon Tuplin. 2007. The identifying extended Kalman filter: parametric system identification of a vehicle handling model. *Proceedings of the Institution of Mechanical Engineers, Part K: Journal of Multi-body Dynamics* 221, 1 (2007), 87–98.

[11] T Bohlin. 1976. Four cases of identification of changing systems. In *Mathematics in Science and Engineering*. Vol. 126. Elsevier, 441–518.

[12] Adith Boloor, Karthik Garimella, Xin He, Christopher Gill, Yevgeniy Vorobeychik, and Xuan Zhang. 2020. Attacking vision-based perception in end-to-end autonomous driving models. *Journal of Systems Architecture* (2020), 101766.

[13] Ajay Kumar Boyat and Brijendra Kumar Joshi. 2015. A review paper: noise models in digital image processing. *arXiv preprint arXiv:1505.03489* (2015).

[14] Jason Brownlee. 2019. A Gentle Introduction to RNN Unrolling. https://machinelearningmastery.com/rnn-unrolling/. (2019).

[15] Business Line. 2018. Velodyne LiDAR awarded Perception System Contract from Mercedes-Benz. https://www.thehindubusinessline.com/business-wire/velodyne-lidar-awarded-perception-system-contract-from-mercedesbenz/article9856916.ece. (2018).

[16] Yulong Cao, Ningfei Wang, Chaowei Xiao, Dawei Yang, Jin Fang, Ruigang Yang, Qi Alfred Chen, Mingyan Liu, and Bo Li. 2021. Invisible for both camera and lidar: Security of multi-sensor fusion based perception in autonomous driving under physical-world attacks. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 176–194.

[17] Yulong Cao, Chaowei Xiao, Benjamin Cyr, Yimeng Zhou, Won Park, Sara Rampazzi, Qi Alfred Chen, Kevin Fu, and Z Morley Mao. 2019. Adversarial sensor attack on lidar-based perception in autonomous driving. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 2267–2281.

[18] CBINSIGHTS. 2018. Android of the Auto Industry? How Baidu May Race Ahead Of Google, Tesla, And Others In Autonomous Vehicles. https://www.cbinsights.com/research/baidu-china-autonomous-vehicles/. (2018).

[19] Varun Chandrasekaran, Kamalika Chaudhuri, Irene Giacomelli, Somesh Jha, and Songbai Yan. 2020. Exploring connections between active learning and model extraction. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*. 1309–1326.

[20] Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, Tadayoshi Kohno, et al. 2011. Comprehensive experimental analyses of automotive attack surfaces.. In *USENIX Security Symposium*, Vol. 4. San Francisco, 447–462.

[21] Kyong-Tak Cho and Kang G Shin. 2016. Fingerprinting electronic control units for vehicle intrusion detection. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*. 911–927.

[22] CISOMAG. 2020. Tesla Offers US$1 Million and a Car as Bug Bounty Reward. https://cisomag.eccouncil.org/tesla-offers-us1-million-and-a-car-as-bug-bounty-reward/. (2020).

[23] DARPA. 2014. The DARPA Grand Challenge: Ten Years Later. https://www.darpa.mil/news-events/2014-03-13. (2014).

[24] Matthew DeBord. 2018. Waymo has launched its commercial self-driving service in Phoenix - and it's called 'Waymo One'. https://www:businessinsider:com/waymo-one-driverless-car-servicelaunches-in-phoenix-arizona-2018-12. (2018).

[25] David Di Ruscio. 2003. Subspace system identification of the Kalman filter. (2003).

[26] ETH Zürich. 2018. Ethzasl MSF Framework. https://github.com/ethz-asl/ethzasl_msf. (2018).

[27] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. 2018. Robust physical-world attacks on deep learning visual classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1625–1634.

[28] Frida. 2022. Dynamic instrumentation toolkit for developers, reverse-engineers, and security researchers. https://frida.re/. (2022).

[29] Bernard Friedland. 2012. *Control system design: an introduction to state-space methods*. Courier Corporation.

[30] Quan Bo Ge, Wen Bin Li, Ruo Yu Sun, and Zi Xu. 2013. Centralized fusion algorithms based on EKF for multisensor non-linear systems. *Zidonghua Xuebao/Acta Automatica Sinica* 39, 6 (2013), 816–825.

[31] RMO Gemson. 1991. Estimation of aircraft aerodynamic derivatives accounting for measurement and process noise by EKF through adaptive filter tuning. *Bangalore, India: Department of Aerospace Engineering, Indian Institute of Science* (1991).

[32] Christopher Goodin, Daniel Carruth, Matthew Doude, and Christopher Hudson. 2019. Predicting the Influence of Rain on LIDAR in ADAS. *Electronics* 8, 1 (2019), 89.

[33] GreenValley International. 2020. LiBackpack - Mobile Handheld LiDAR - 3D Mapping System. https://greenvalleyintl.com/hardware/libackpack/. (2020).

[34] Hex Rays. 2022. Ida Pro. https://hex-rays.com/ida-pro/. (2022).

[35] HEXAGON. 2020. EasyMile and Velodyne Lidar Announce Three-Year Agreement. https://insideunmannedsystems.com/easymile-and-velodyne-lidar-announce-three-year-agreement/. (2020).

[36] Charles G Hilborn and Demetrios G Lainiotis. 1969. Optimal estimation in the presence of unknown parameters. *IEEE Transactions on Systems Science and Cybernetics* 5, 1 (1969), 38–43.

[37] Gabriel M Hoffmann, Claire J Tomlin, Michael Montemerlo, and Sebastian Thrun. 2007. Autonomous automobile trajectory tracking for off-road driving: Controller design, experimental validation and racing. In *2007 American control conference*. IEEE, 2296–2301.

[38] Bernhard Hofmann-Wellenhof, Herbert Lichtenegger, and Elmar Wasle. 2007. *GNSS–global navigation satellite systems: GPS, GLONASS, Galileo, and more*. Springer Science & Business Media.

[39] Tehrani Nik Nejad Hossein, Seiichi Mita, and Han Long. 2010. Multi-sensor data fusion for autonomous vehicle navigation through adaptive particle filter. In *2010 IEEE Intelligent Vehicles Symposium*. IEEE, 752–759.

[40] Steve Corrigan HPL. 2002. Introduction to the controller area network (CAN). *Application Report SLOA101* (2002), 1–17.

[41] Informed Infrastructure. 2012. Velodyne's LiDAR Division Announces Agreement With Caterpillar for Laser Imaging Technology. https://informedinfrastructure.com/25630/velodynes-lidar-division-announces-agreement-with-caterpillar-for-laser-imaging-technology-2/. (2012).

[42] Inside Unmanned Systems. 2020. EasyMile and Velodyne Lidar Announce Three-Year Agreement. https://insideunmannedsystems.com/easymile-and-velodyne-lidar-announce-three-year-agreement/. (2020).

[43] Intel. 2022. Pin - A Dynamic Binary Instrumentation Tool. https://www.intel.com/content/www/us/en/developer/articles/tool/pin-a-dynamic-binary-instrumentation-tool.html. (2022).

[44] Matthew Jagielski, Nicholas Carlini, David Berthelot, Alex Kurakin, and Nicolas Papernot. 2020. High Accuracy and High Fidelity Extraction of Neural Networks. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*.

[45] Matthew Jagielski, Nicholas Carlini, David Berthelot, Alex Kurakin, and Nicolas Papernot. 2020. High Accuracy and High Fidelity Extraction of Neural Networks . In *Proceedings of the 29th USENIX Security Symposium (USENIX Security '20)*. Boston, MA.

[46] Jinyong Jeong, Younggun Cho, Young-Sik Shin, Hyunchul Roh, and Ayoung Kim. 2019. Complex urban dataset with multi-level sensors from highly diverse urban environments. *The International Journal of Robotics Research* (2019), 0278364919843996.

[47] Saurabh Jha, Shengkun Cui, Subho S. Banerjee, James Cyriac, Timothy Tsai, Zbigniew Kalbarczyk, and Ravishankar K. Iyer. 2020. ML-Driven Malware that Targets AV Safety. In *50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2020, Valencia, Spain, June 29 - July 2, 2020*. IEEE, 113–124. https://doi.org/10.1109/DSN48063.2020.00030

[48] Saurabh Jha, Shengkun Cui, Subho S Banerjee, Timothy Tsai, Zbigniew Kalbarczyk, and Ravi Iyer. 2020. ML-driven Malware that Targets AV Safety. *arXiv preprint arXiv:2004.13004* (2020).

[49] Yunhan Jia, Yantao Lu, Junjie Shen, Qi Alfred Chen, Hao Chen, Zhenyu Zhong, and Tao Wei. 2019. Fooling detection alone is not enough: Adversarial attack against multiple object tracking. In *International Conference on Learning Representations*.

[50] Pengfei Jing, Qiyi Tang, Yuefeng Du, Lei Xue, Xiapu Luo, Ting Wang, Sen Nie, and Shi Wu. 2021. Too good to be safe: Tricking lane detection in autonomous driving with crafted perturbations. In *30th USENIX Security Symposium (USENIX Security 21)*. 3237–3254.

[51] Jer-Nan Juang, Minh Phan, Lucas G Horta, and Richard W Longman. 1993. Identification of observer/Kalman filter Markov parameters-Theory and experiments. *Journal of Guidance, Control, and Dynamics* 16, 2 (1993), 320–329.

[52] Mika Juuti, Sebastian Szyller, Samuel Marchal, and N Asokan. 2019. PRADA: protecting against DNN model stealing attacks. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 512–527.
[53] Evgueny Khartchenko. 2018. Vectorization: A Key Tool To Improve Performance On Modern CPUs. https://software.intel.com/content/www/us/en/develop/articles/vectorization-a-key-tool-to-improve-performance-on-modern-cpus.html. (2018).
[54] Karl Koscher, Alexei Czeskis, Franziska Roesner, Shwetak Patel, Tadayoshi Kohno, Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, et al. 2010. Experimental security analysis of a modern automobile. In *2010 IEEE Symposium on Security and Privacy*. IEEE, 447–462.
[55] Suki Lau. 2017. Learning Rate Schedules and Adaptive Learning Rate Methods for Deep Learning. https://towardsdatascience.com/learning-rate-schedules-and-adaptive-learning-rate-methods-for-deep-learning-2c8f433990d1. (2017).
[56] Steffen L Lauritzen. 1981. Time series analysis in 1880: A discussion of contributions made by TN Thiele. *International Statistical Review/Revue Internationale de Statistique* (1981), 319–331.
[57] Lennart Ljung. 1999. System identification. *Wiley encyclopedia of electrical and electronics engineering* (1999), 1–19.
[58] Jiajun Lu, Hussein Sibai, and Evan Fabry. 2017. Adversarial examples that fool detectors. *arXiv preprint arXiv:1712.02494* (2017).
[59] Mulong Luo, Andrew C Myers, and G Edward Suh. 2020. Stealthy Tracking of Autonomous Vehicles with Cache Side Channels. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*. 859–876.
[60] Simon Lynen, Markus W Achtelik, Stephan Weiss, Margarita Chli, and Roland Siegwart. 2013. A robust and modular multi-sensor fusion approach applied to mav navigation. In *2013 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, 3923–3929.
[61] Raman Mehra. 1970. On the identification of variances and adaptive Kalman filtering. *IEEE Transactions on automatic control* 15, 2 (1970), 175–184.
[62] Raman Mehra. 1972. Approaches to adaptive filtering. *IEEE Transactions on automatic control* 17, 5 (1972), 693–698.
[63] Srinivas Soumitri Miriyala and Kishalay Mitra. 2020. Deep learning based system identification of industrial integrated grinding circuits. *Powder Technology* 360 (2020), 921–936.
[64] AH Mohamed and KP Schwarz. 1999. Adaptive Kalman filtering for INS/GPS. *Journal of geodesy* 73, 4 (1999), 193–203.
[65] Shyam Mohan M, Naren Naik, RMO Gemson, and MR Ananthasayanam. 2015. Introduction to the Kalman filter and tuning its statistics for near optimal estimates and Cramer Rao bound. *arXiv* (2015), arXiv–1503.
[66] Kevin P Murphy. 2012. *Machine learning: a probabilistic perspective*. MIT press.
[67] Kenneth Myers and BD Tapley. 1976. Adaptive sequential estimation with unknown noise statistics. *IEEE Trans. Automat. Control* 21, 4 (1976), 520–523.
[68] Sen Nie, Ling Liu, and Yuefeng Du. 2017. Free-fall: Hacking tesla from wireless to can bus. *Briefing, Black Hat USA* 25 (2017), 1–16.
[69] Brian J Odelson, Alexander Lutz, and James B Rawlings. 2006. The autocovariance least-squares method for estimating covariances: application to model-based control of chemical reactors. *IEEE transactions on control systems technology* 14, 3 (2006), 532–540.
[70] Kun Il Park. 2017. *Fundamentals of Probability and Stochastic Processes with Applications to Communications* (1st ed.). Springer Publishing Company, Incorporated.
[71] Jonathan Petit and Steven E Shladover. 2014. Potential cyberattacks on automated vehicles. *IEEE Transactions on Intelligent transportation systems* 16, 2 (2014), 546–556.
[72] South China Morning Post. 2020. Chinese internet giant Baidu offers free trial robotaxi rides through search and map apps in Changsha. https://www.scmp.com/tech/apps-social/article/3080712/chinese-internet-giant-baidu-offers-free-trial-robotaxi-rides. (2020).
[73] Raul Quinonez, Jairo Giraldo, Luis Salazar, Erick Bauman, Alvaro Cardenas, and Zhiqiang Lin. 2020. {SAVIOR}: Securing Autonomous Vehicles with Robust Physical Invariants. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*. 895–912.
[74] Tyler GR Reid, Sarah E Houts, Robert Cammarata, Graham Mills, Siddharth Agarwal, Ankit Vora, and Gaurav Pandey. 2019. Localization requirements for autonomous vehicles. *arXiv preprint arXiv:1906.01061* (2019).
[75] Renovo.auto Blog. 2017. Renovo selects Velodyne as Reference LiDAR provider for advanced automotive development projects. https://medium.com/renovo-auto-blog/renovo-selects-velodyne-as-reference-lidar-provider-for-aware-automated-mobility-operating-system-f8001b91c6c. (2017).
[76] Reuters. 2021. Ford dissolves its 7.6% stake in Velodyne Lidar. https://www.reuters.com/article/us-velodyne-lidar-stake/ford-dissolves-its-7-6-stake-in-velodyne-lidar-idUSKBN2AF1B7?il=0. (2021).
[77] Daniel E Rivera, Manfred Morari, and Sigurd Skogestad. 1986. Internal model control: PID controller design. *Industrial & engineering chemistry process design and development* 25, 1 (1986), 252–265.
[78] Atsushi Sakai, Daniel Ingram, Joseph Dinius, Karan Chawla, Antonin Raffin, and Alexis Paques. 2018. PythonRobotics: a Python code collection of robotics algorithms. *CoRR* abs/1808.10703 (2018). arXiv:1808.10703 http://arxiv.org/abs/1808.10703
[79] Junjie Shen, Jun Yeon Won, Zeyuan Chen, and Qi Alfred Chen. 2020. Drift with Devil: Security of Multi-Sensor Fusion based Localization in High-Level Autonomous Driving under GPS Spoofing. In *Proceedings of the 29th USENIX Security Symposium (USENIX Security '20)*. Boston, MA.
[80] Robert H Shumway and David S Stoffer. 2017. *Time series analysis and its applications: with R examples*. Springer.
[81] Joan Sola. 2017. Quaternion kinematics for the error-state Kalman filter. *arXiv preprint arXiv:1711.02508* (2017).
[82] Jae Kyu Suhr, Jeungin Jang, Daehong Min, and Ho Gi Jung. 2016. Sensor fusion-based low-cost vehicle localization system for complex urban environments. *IEEE Transactions on Intelligent Transportation Systems* 18, 5 (2016), 1078–1086.
[83] Jiachen Sun, Yulong Cao, Qi Alfred Chen, and Z Morley Mao. 2020. Towards Robust LiDAR-based Perception in Autonomous Driving: General Black-box Adversarial Sensor Attack and Countermeasures. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*. 877–894.
[84] System Plus Consulting. 2018. Bosch's 6-Axis IMU in the Apple iPhone X. https://www.systemplus.fr/reverse-costing-reports/boschs-6-axis-imu-in-the-apple-iphone-x/. (2018).
[85] Tatsuya Takemura, Naoto Yanai, and Toru Fujiwara. 2020. Model Extraction Attacks against Recurrent Neural Networks. *arXiv preprint arXiv:2002.00123* (2020).
[86] Tencent Keen Security Lab. 2020. Exploiting Wi-Fi Stack on Tesla Model S. https://keenlab.tencent.com/en/2020/01/02/exploiting-wifi-stack-on-tesla-model-s/. (2020).
[87] Nils Ole Tippenhauer, Christina Pöpper, Kasper Bonne Rasmussen, and Srdjan Capkun. 2011. On the requirements for successful GPS spoofing attacks. In *Proceedings of the 18th ACM conference on Computer and communications security*. 75–86.
[88] Trail of Bits. 2022. McSema. https://github.com/lifting-bits/mcsema. (2022).
[89] Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. 2016. Stealing machine learning models via prediction apis. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*. 601–618.
[90] Yazhou Tu, Zhiqiang Lin, Insup Lee, and Xiali Hei. 2018. Injected and delivered: Fabricating implicit control over actuation systems by spoofing inertial sensors. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*. 1545–1562.
[91] United States Securities and Exchange Commission. 2020. Schedule 13G Under the Securities Exchange Act of 1934 (Amendment No. 1) – Velodyne Lidar, Inc. https://www.reuters.com/article/us-velodyne-lidar-stake/ford-dissolves-its-7-6-stake-in-velodyne-lidar-idUSKBN2AF1B7?il=0. (2020).
[92] Unmanned Systems Technology. 2018. Velodyne Lidar Partners with Nikon for Autonomous Vision. https://www.unmannedsystemstechnology.com/2018/12/velodyne-lidar-partners-with-nikon-for-autonomous-vision/. (2018).
[93] Velodyne. 2022. Velodyne Lidar: Envision the Future. https://velodynelidar.com/. (2022).
[94] Guowei Wan, Xiaolong Yang, Renlan Cai, Hao Li, Yao Zhou, Hao Wang, and Shiyu Song. 2018. Robust and precise vehicle localization based on multi-sensor fusion in diverse city scenes. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 4670–4677.
[95] Ziwen Wan, Junjie Shen, Jalen Chuang, Xin Xia, Joshua Garcia, Jiaqi Ma, and Qi Alfred Chen. 2022. Too Afraid to Drive: Systematic Discovery of Semantic DoS Vulnerability in Autonomous Driving Planning under Physical-World Attacks. In *Network and Distributed System Security (NDSS) Symposium, 2022*.
[96] Binghui Wang and Neil Zhenqiang Gong. 2018. Stealing hyperparameters in machine learning. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 36–52.
[97] Greg Welch, Gary Bishop, et al. 1995. An introduction to the Kalman filter. (1995).
[98] yegord. 2022. Snowman. https://github.com/yegord/snowman. (2022).
[99] Megan A Zagrobelny and James B Rawlings. 2014. Identification of disturbance covariances using maximum likelihood estimation. *Technical report, No. 2014–02* (2014).
[100] Yue Zhao, Hong Zhu, Ruigang Liang, Qintao Shen, Shengzhi Zhang, and Kai Chen. 2019. Seeing isn't Believing: Towards More Robust Adversarial Attack Against Real World Object Detectors. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 1989–2004.

A AD CONTROLLER

After localization, the estimated vehicle status is combined with the output of path planning, collision avoidance, etc. in the navigation stack, and sent to the controller module to guide actuators/command executors. We briefly overview this module here, since it has to be emulated for the attack scenarios when the output to ESKF is not directly observable to the adversary (see Section 3.2).

An AD controller is normally divided into 2 sub-components: 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:

$$G = \begin{pmatrix} I_6 & 0 & 0 \\ 0 & I_3 - [\frac{1}{2}\delta\hat{\theta}] & 0 \\ 0 & 0 & I_9 \end{pmatrix} \tag{11}$$

Where I_n represents an $n \times n$ identity matrix and $\delta\hat{\theta}$ 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.

$$\begin{aligned} K' &= P_k H_G^\top (H_G P_k H_G^\top + R_G)^{-1} \\ \hat{x_k} &= x_k + K'(z_k - H_G x_k) \\ \hat{P_k} &= P_k - K' H_G P_k \end{aligned} \tag{12}$$

$$\begin{aligned} K' &= P_k (H_L^p)^\top (H_L^p P_k (H_L^p)^\top + R_L^p)^{-1} \\ x_k' &= x_k + K'(z_k^p - H_L^p x_k) \\ P_k' &= P_k - K' H_L^p P_k \\ K'' &= P_k (H_L^y)^\top (H_L^y P_k (H_L^y)^\top + R_L^y)^{-1} \\ \hat{x_k} &= x_k' + K''(z_k^y - H_L^y x_k) \\ \hat{P_k} &= P_k' - K'' H_L^y P_k \end{aligned} \tag{13}$$

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 `liblocalization_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 decompiled. 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.

Symbol	Description	Dim
u_{k-1}	IMU measurement	2×1
z_k^p	Position measurement	3×1
z_k^y	Yaw measurement	4×1
x_k	Predicted state	16×1
P_k	Predicted cov	15×15
y_k	Controller output	4×1
p_k	Position control	2×1
a_i	Acceleration control	1×1
d_i	Yaw control	1×1
${}^*Q$	Observation noise cov	15×15
${}^*R_G$	GNSS noise cov	3×3
${}^*R_L^p$	LiDAR position noise cov	3×3
${}^*R_L^y$	LiDAR yaw noise cov	3×3
O	Ground-truth model	-
$\hat{O}$	Extracted model	-

Table 3: Symbols used in Section 4. "Dim" is for Dimension. "cov" is for covariance matrix. "*" marks the secret to be inferred by TaskMaster.

Algorithm 1: Attack workflow under AS1

Input : N measurement $T = [t_1, ..., t_N]$, the output of O refStates, MaxEpoch
Output: Inferred Q, R_G, R_L^p, R_L^y

```
Initialize x_0, P_0, Q, R_G, R_L^p, R_L^y, myStates;
x_{k-1} ← x_0; P_{k-1} ← P_0; cnt ← 1;
for i ← 1 to MaxEpoch do
    while cnt ≤ N do
        t_k ← get(T,cnt);
        if t_k is from IMU then
            x_k, P_k ← predictIMU(x_{k-1}, P_{k-1}, t_k);
        end
        if t_k is from GNSS then
            x_k, P_k ← updateGNSS(x_{k-1}, P_{k-1}, t_k);
        end
        if t_k is from LiDAR then
            x_k, P_k ← updateLiDAR(x_{k-1}, P_{k-1}, t_k);
        end
        add x_k into myStates;
        x_{k-1} ←x_k; P_{k-1} ←P_k;cnt ← cnt + 1;
    end
    Loss ← log(MSE(myStates, refStates)) ;
    optimize(Q,Loss);
    optimize(R_G, R_L^p, R_L^y,Loss);
end
Return Q, R_G, R_L^p, R_L^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 R 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 (λ_p, λ_d and λ_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]. λ, α, β and σ are model parameters.

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

$$f(x;\lambda) = \lambda e^{-\lambda x} \tag{14}$$

where $\lambda > 0$. Mean of Exponential noise is $\frac{1}{\lambda}$, and standard deviation is also $\frac{1}{\lambda}$.

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

Table 4: The impact of N on AS1. The targeted model is `Sola-ESKF`. *local08* and *highway17* are for training and testing.

		Q			R		
AS	Original	$[1e-3, 1e-2]$	$[1e-2, 1e-1]$	$[1e-1, 1e0]$	$[1e-6, 1e-5]$	$[1e-5, 1e-4]$	$[1e-4, 1e-3]$
AS1	0.01347	0.01220	0.01265	0.14031	0.03868	0.10514	0.92062
AS2	0.00978	0.09540	0.12428	0.17274	0.12237	0.16641	0.50272

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.

	Q			R		
Original	$[1e-3, 1e-2]$	$[1e-2, 1e-1]$	$[1e-1, 1e0]$	$[1e-6, 1e-5]$	$[1e-5, 1e-4]$	$[1e-4, 1e-3]$
0.85	0.97	1.03	1.73	0.88	2.43	5.64

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*.

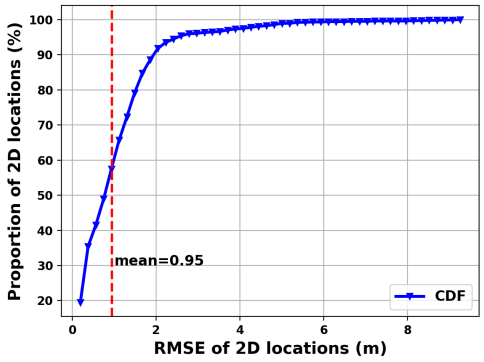

Figure 5: Distribution of RMSE on 2D locations in AS1 on `Apollo-ESKF`. The training trace is *highway17* and the testing trace is *local08*.

Testing (SER)				
local08	*local07*	*local31*	*highway06*	*highway17*
12.60	8.15	9.18	7.90	5.31

Table 7: Comparison between `Sola-ESKF` and `Apollo-ESKF` on the same `KAIST` dataset.

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

$$f(x;\alpha,\beta) = \frac{\beta^{\alpha}}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x} \tag{15}$$

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:

$$f(x;\sigma) = \frac{x}{\sigma^2} e^{-x^2/2\sigma^2} \tag{16}$$

where $\sigma > 0$. Mean of Rayleigh noise is $\sigma\sqrt{\frac{\pi}{2}}$, and standard deviation is $\sigma\sqrt{\frac{4-\pi}{2}}$.

I 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).