CPS Device-Class Identification via Behavioral Fingerprinting: From Theory to Practice

Leonardo Babun®, Member, IEEE, Hidayet Aksu®, and A. Selcuk Uluagac®

Abstract—Cyber-Physical Systems (CPS) utilize different devices to collect sensitive data, communicate with other systems, and monitor essential processes in critical infrastructure applications. However, in the ecosystem of CPS, unauthorized or spoofed devices may endanger or compromise the performance and security of the critical infrastructure. The unauthorized and spoofed devices may include tampered pieces of software or hardware components that can negatively impact CPS operations or collect vital CPS metrics from the network. Such devices can be outsider or insider threats trying to impersonate other real CPS devices via spoofing their legitimate identifications to gain access to systems, steal information, or spread malware. Device fingerprinting techniques are promising approaches to identify unauthorized or illegitimate devices. However, current fingerprinting solutions are not suitable as they disrupt critical real-time operations in CPS due to the nature of their intrusive data analysis or too much overhead on the devices’ computational resources. To address these concerns, in this work, we propose STOP-AND-FRISK (S&F), a novel fingerprinting framework to identify CPS device classes and complement traditional security mechanisms in CPS. S&F is based on a secure challenge/response mechanism that analyzes the behavior of the CPS devices at both the hardware and OS/kernel levels. Specifically, the proposed novel mechanism combines system and function call tracing techniques, signal processing, and hardware performance analysis to create specific device-class signatures. Then, the signatures are correlated against known behavioral ground-truth to identify the device types. To test the efficacy of S&F extensively, we implemented a realistic testbed that included different classes of CPS devices with a variety of computing resources, architectures, and configurations. Our experimental results reveal an excellent rate on the CPS device-class identification. Finally, extensive performance analysis demonstrates that the use of S&F yields minimal overhead on the CPS devices’ computing resources.

Index Terms—Cyber-physical systems, device-class fingerprinting, correlation, system calls, function calls, hardware performance.

I. INTRODUCTION

A THE core of the Cyber-Physical Systems (CPS) (e.g., transportation systems, smart grid, gas and oil plants), devices such as Remote Terminal Units (RTUs), Programmable Logic Controllers (PLCs), and Intelligent Electronic Devices (IEDs) are utilized to collect sensitive data from the infrastructure, provide two-way communication capabilities, and monitor the health of the CPS operations in real-time [1]–[3]. However, these devices also present an opportunity for attackers to have access to sensitive information and the critical CPS infrastructure [4]–[6]. For instance, insiders can impersonate real CPS devices via spoofing attacks to gain access to the systems, steal information, make other devices in the network behave erratically, or spread malware (Figure 1) [7]–[11]. Also, illegitimate CPS devices may have installed unauthorized pieces of software and hardware that could degrade the performance of the devices and compromise the integrity of the CPS network. Recent studies demonstrate that compromised manufacturing stages within the supply chain may facilitate malicious manipulation and modification of CPS devices’ components to deliver a downgraded product to the critical infrastructures [12].

Protecting against such attacks stemming from spoofed or unauthorized devices can be very challenging, considering the size and complexity of the CPS infrastructure. For instance, an attacker may use spoofed devices with software and hardware architectures very similar to real devices to increase the chances of stealthy malicious operations. In fact, the spoofed devices can perform the attacks while mimicking real CPS operations. Also, devices with unauthorized components are often capable of supporting most of the CPS operations, but are prone to under-performance and failure when in charge of more demanding and critical tasks. In these scenarios, device fingerprinting techniques are suitable to identify original devices and discriminate them from the unauthorized and the spoofed devices. However, current fingerprinting solutions either require extensive analysis of network packets and device features or study the behavior of very dynamic network metrics [11], [13]–[17]. Thus, in most cases, these solutions introduce significant overhead to devices and systems, putting the execution of critical time-sensitive CPS tasks at risk.
A. Goals and Contributions

In this paper, we address these challenges by proposing STOP-AND-FRISK (S&F),\(^1\) a novel signature-based fingerprinting framework intended to perform CPS device-class identification and complement other traditional security mechanisms in the CPS infrastructure. Specifically, the proposed approach combines system and function call tracing techniques, signal processing, and hardware performance analysis on the devices to implement a secure challenge/response-based fingerprinting solution. By using this approach, S&F studies the behavior of the devices within a CPS infrastructure, focusing on their software/hardware architecture and configuration to identify real CPS devices and discriminate them from unauthorized and spoofed devices. The benefits of S&F are two-fold. First, it combines a secure challenge-response approach with signature-based fingerprinting techniques to identify spoofed devices in the field. Second, it studies the performance of CPS devices to detect devices with degraded software and hardware that could compromise the CPS’s critical operations. To test the efficacy of S&F, we implemented a realistic testbed containing different classes of CPS devices with different resources (i.e., memory and CPU), architectures, computing capabilities, and configurations. Our extensive experimental results demonstrate that, by combining OS/kernel behavioral analysis and hardware performance analysis, S&F achieves an excellent rate in the identification of the CPS device classes. Finally, S&F yielded minimal overhead to the CPS devices’ computing resources.

The contributions and summary of this work are as follows:

- **Novel Features for CPS Behavioral Analysis:** We proposed a novel combination of features to study the behavior of CPS devices. These features are related to the OS/kernel and hardware performance behaviors of the devices.
- **STOP-AND-FRISK:** We designed a novel and lightweight signature-based fingerprinting approach that performs CPS device-class identification. The proposed framework combines system and function call information, signal processing, and hardware performance analysis to perform the identification of spoofed and unauthorized devices in the CPS network.
- **Realistic End-to-End Implementation:** We proposed and tested a secure end-to-end deployment mechanism for S&F using a realistic CPS testbed that includes 11 different classes of CPS devices with varying resources and configurations.
- **High Performance:** Our performance evaluation proved that the proposed framework achieves very high accuracy while introducing minimal overhead in the utilization of the computing resources available in the CPS devices.

B. Organization

In Section II, we present background information on the classification of CPS devices based on their behavior. Then, in Section III, we define the problem and present the threat model. Section IV details the architecture of S&F and discusses its main components and analysis approaches. Further, Section V presents implementation and deployment details of S&F. Then, in Section VI, we present and evaluate the experimental results. We discuss the performance of S&F under different compromised scenarios in Section VII. Finally, in Section VIII we discuss the related work and Section IX concludes the paper.

II. BACKGROUND INFORMATION

A. Cyber-Physical Systems

Cyber-Physical Systems (CPSs) permit the integration of virtual and physical processes. In this context, the physical domain refers to capabilities that act over physical objects. On the other hand, the virtual domain constitutes the set of software and embedded systems intended to guarantee two-way communications, monitor the realization of the physical processes, and provide control [18]. In general, one can characterize CPSs networks by using the following features:

- **Type of Task Performed:** Depending on the specific application and their logical location inside the CPS architecture, the type of task performed by CPS devices may range from just a simple service generated by a local host device to an essential component of a more complex and centralized process. In any case, individual CPS processes are assumed to be simple, deterministic, and very specific actions that support the entire system in a distributed topology [19].
- **Resource Availability:** The total amount of available computing resources to perform CPS processes depends on the type of device performing every particular task. In general, we can group CPS devices into resource-rich and resource-limited devices [8], [20]. Resource-limited devices have simple hard (e.g., single-core CPU and limited memory) and software architecture that allows for the execution of simple, specific tasks. On the other hand, resource-rich devices have more complex Operating System (OS) architecture and run with multi-core CPUs and plenty of memory. These capabilities allow them to execute more complex processes inside the CPS network.
- **Timing Properties:** As we noted before, one of the main goals of the cyber domain in CPS is the monitoring and control of physical processes, which is achieved through rigorous timing control mechanisms. In general, temporal behavior of CPS is expected to be precise, and should not change too much over time [19].

B. Device-Class Identification

Traditionally, device-class classification has been performed by considering the branch, model, specific device metrics characteristics, and the activities the devices should perform in the network [21]. S&F implements a more comprehensive approach to identify types of devices in the network that also considers (1) the device behavior at the OS or kernel level and (2) its performance metrics at the hardware level. The main advantage of an approach that includes behavior and hardware performance into its analysis is that it allows for a more secure identification approach that does not depend on device characteristics or metrics that can be spoofed by savvy (or even naive) attackers.

In this work, we consider the following features to define specific classes of CPS devices:

\(^1\)The term STOP-AND-FRISK makes reference to the architectural principles of the proposed fingerprinting mechanism and does not constitute an endorsement to any law enforcement practice.
- **Device Metrics**: We use well-known metrics like device’s branch, model, and expected functionality to perform initial classification of the devices. The expected functionality of the device mostly refers to the intended application of the device based on the device characteristics (e.g., routers and firewalls may be divided into two different groups based on their unique application and intended use). This preliminary analysis supports the labeling process performed before evaluating S&F (more details in Section V).

- **Device Behavior**: This feature characterizes the device response to specific challenges at the OS and kernel levels. S&F studies the device’s behavior based on the collection of system and function calls triggered while reacting to specific challenges.

- **Device Performance**: It characterizes the device’s response to specific challenges at the hardware level. S&F studies the device’s performance by evaluating the device’s memory and CPU utilization as well as the application execution time while reacting to specific challenges or stimuliants.

### III. Problem Definition and Threat Model

This work assumes a CPS network \( \mathcal{N} \) within a critical infrastructure (i.e., the smart grid) that contains devices with various functionalities and computational resources. First, we consider that the supply chain that provides the devices cannot be trusted and is assumed compromised during any of the sourcing, manufacturing, assembling, packing, and delivery processes. Hence, the devices being used in the critical infrastructure may contain unauthorized pieces of hardware and software that could either degrade their performance, or execute malicious or unexpected functionalities. At some point, a network administrator may install a Programmable Logic Controller (PLC) device that was manufactured with a low-cost Central Processing Unit (CPU). The low-end CPU adds additional delays to the device capabilities to react to inputs, which causes degradation to the device’s performance and to the CPS network’s effectiveness to respond to time-critical tasks [8]. Second, we consider that external attackers may have the capability of inserting fake (i.e., spoofed) devices into the network. These spoofed devices have similar computing characteristics and can execute real tasks with similar performance as legitimate devices. Also, these devices are capable of malicious activities allowing the attackers to gain unauthorized access to different regions of the CPS network and its critical data. Mallory, an insider that has access to the CPS network, could insert a spoofed device (e.g., a BeagleBoard) that has been programmed using open source IEC61850 libraries freely available online [22]. With this device, Mallory can establish communications with real CPS devices and implement GPS spoofing attacks [23].

#### A. Threat Model

This work considers an attacker (insider or outsider) capable of inserting spoofed devices into a CPS infrastructure. The unauthorized devices spoof real CPS devices and operations to gain access to restricted areas of the network and perform malicious activities. These malicious operations may include:

1. stealing sensitive information,
2. poisoning physical measurements, and
3. creating the conditions to facilitate new types of attacks in the future. We do not consider insiders that have access and can compromise original CPS devices with the same hardware and software configurations used in the field. Instead, we assume that attackers utilize spoofed devices that mimic real CPS network operations to gain access to the network. We also consider unauthorized or illegitimate CPS devices that contain unauthorized software or hardware added during any of the production stages (i.e., raw material sourcing, manufacturing, assembly, testing, and delivery) of the supply chain. The use of unauthorized hardware or software may cause degradation to the device’s performance and create unsafe states in the CPS critical infrastructure.

### IV. Host-Based CPS Device Class Fingerprinting

In this section, we overview STOP-AND-FRISK and present the details about its modules and processes. S&F is a novel and lightweight fingerprinting framework that uses a secure challenge-response approach to extract behavioral data from unknown CPS devices to create specific device signatures. These signatures are then correlated with known device profiles for identification purposes. Specifically, we take advantage of the fingerprinting and identification capabilities of S&F to solve the problems above. First, as S&F considers device performance into its analysis, we can identify devices that are under-performing due to unauthorized hardware of software. Second, the proposed fingerprinting framework can identify spoofed CPS devices that either fail the challenge-response process or that are incapable of generating the expected signature. Finally, S&F’s capabilities support automated configuration mechanisms for similar devices that share akin tasks in the network.

#### A. Overview of STOP-AND-FRISK

Assume that there is a CPS critical infrastructure where devices of different types interact to execute a task \( T \). The specific class of some of the devices in the setup is known (i.e., Type A and Type B); however, an Unknown device is also present. With S&F, the network operators may be able to verify that (1) the devices in the critical infrastructure are of the expected class (based on the specific tasks they are executing) and (2) they may be able to identify unknown devices and determine if they are authorized to be present or not. Since most of the CPS devices perform time-critical operations in the network, we envision our CPS device class fingerprinting framework to become active at the device’s patch- or maintenance-time (i.e., downtime). That way, S&F’s operations would focus on individual devices and would put minimal overhead on the systems. Such operations require the interaction of two different services: a server-based remote service (running from a remote server that monitors the CPS environment) and a host-based local service (running on the CPS devices).

Figure 2 depicts the general overview of the proposed device-class fingerprinting framework. First, a scheduler running in the remote S&F’s server sends a secure request containing a secret challenge to the unknown CPS device (i.e., localhost) at downtime (1). Such a challenge implements the host-based local service that activates the Device.
B. Device Feature Extraction

Device Feature Extraction module (②). This module is in charge of running the secret challenge and extracting software- and hardware-related data generated during the device’s reaction to the challenge. Specifically, it hooks into the device’s activity and extracts lists of system and function calls. Additionally, the Device Feature Extraction monitors the performance of the device regarding CPU utilization, memory utilization, and the execution time of the challenge while extracting the calls. Once finished, the module derives specific features from the collected data. These features are related to the set of functions and system calls triggered, the number of different call types, and their arguments, respectively. Additionally, it computes the CPU utilization, the amount of memory allocated to the device, and the total execution time. Once all the required features are acquired, the local service securely sends them to the remote server using a for further analysis (③). On the server side, the collected features are then utilized to generate the signature of the unknown CPS device inside the Signature Generation module (④). Further, the generated signature is correlated (⑤) with ground-truth data previously extracted from known-class CPS devices (i.e., ground-truth) included in the CPS network. Finally, a threshold-based decision algorithm defines the class of the unknown device (⑥).

B. Device Feature Extraction

The first step into the fingerprinting process is to collect the necessary features used to create the unique device-class signature. These features include OS/kernel behavior via system and function calls and hardware performance via memory/CPU utilization and execution time. Compared to current fingerprinting techniques, the implementation of S&F does not require extensive network traffic monitoring. Also, the devices are always monitored at downtime, so no critical CPS operation is interrupted.

1) The Challenge: S&F uses a challenge-response mechanism to generate data that accurately describe the behavior and performance of the unknown CPS devices. Such data is utilized to generate device-type specific signatures that are used later for identification purposes. The Device Feature Extraction module running in the host is the one in charge of securely storing and executing the challenge. There are some advantages associated with executing the challenge locally in the host devices. First, it eliminates the need for creating extra secure channels to deliver files from the remote server, especially at downtime, when connection capabilities may be limited. Second, S&F automatically flags as unauthorized those devices with the wrong reaction to the challenge or where the Device Feature Extraction module is unavailable at test time (which adds additional security layers in cases where capable attackers may try to mimic the performance of S&F). Third, even if the attackers can still implement S&F in the spoofed devices, the final decision depends on the device behavior rather than on metrics that can be easily spoofed. Assuming that the attackers can change the device’s behavior and also modify the hardware performance results in S&F’s signature, they still need to guess the right values to guarantee that the fake signature strongly correlates with the one stored in S&F’s server, for the specific device-type analyzed.

2) Parametric Call List (PCL): S&F utilizes system and function call hooking techniques [24] to collect all the system and function calls that a specific CPS device-class triggers as a response to the pre-determined challenge. From the call lists, the Device Feature Extraction module extracts distinctive device metrics such as (1) the set of specific triggered calls, (2) the total number of calls by type (e.g., malloc, free, open), and (3) the value of specific call’s arguments (e.g., the amount of memory allocated by malloc). We refer to this list of parameters extracted from the system and function calls as Parametric Call List (PCL), which is defined as:

$$PCL_i = \{ x_i \in X_i : \exists X_i \wedge X_i \neq \emptyset \},$$  \hspace{1cm} (1)$$

where $PCL_i$ represents the PCL data extracted from device $i$, $x_i$ represents the arguments of the calls extracted from device $i$, and $X_i$ represents the call lists extracted from device $i$. In general, the hooking technique utilized to extract the system and function calls is a configurable attribute of S&F, and it may be specific to every device’s architecture and OS [24]. Also, the effectiveness of the PCL in identifying CPS classes does not depend on the amount of system and function calls included in the PCL. S&F does not impose a specific sampling rate to collect the calls, but relies on well-known hooking techniques to collect the call data. That is, as opposed to a rate of system and function calls collected over time, the number of function and system calls is determined by the specific device’s response to the challenge, and would always characterize the device behavior.

During our implementation of S&F, we used library interposition and ptrace to hook into the challenge execution and collect the function and system calls. While library interposition is a hooking technique that can be applied to wide range of operating systems to collect system calls, ptrace is a UNIX-based system call that is used to hook into specific function calls [24]. Specifically, we instrumented relevant system call definitions and hooked into the process that executes the S&F’s challenge in the CPS devices. From there, we were able to build the PCLs with relevant information extracted from every call triggered by the challenge process. In Listing 1, we provide an example of instrumentation of the system call malloc, and detail the extra code that we added to enable the collection process.

Finally, to reduce complexity on the host and minimize overhead, once the PCL generation is completed, the collected data is sent to the S&F remote server for processing using secure communication channels. We discuss how to secure S&F’s communications in later sections.

3) Device Performance Index (DPI): The second feature used by the proposed framework to identify CPS device classes is the Device Performance Index (DPI). Since call lists can be
This text involves mathematical expressions, algorithms, and discussions on device-class identification and behavior fingerprinting in cyber-physical systems (CPS). It discusses the use of metrics such as memory utilization, CPU utilization, and execution time to characterize device behavior. The text mentions the use of the Leibniz formula and the scalar triple product in defining a metric for device identification, referred to as DPI (Device Fingerprint Index).

Mathematical expressions include:

\[ \text{DPI} = \frac{1}{n!} |\epsilon_{ijk}| \sum_{i,j,k=1}^{n} \mu_{\text{PCL}}(\text{PCL lists}) \cdot \mu_{\text{DPI}}(\text{DPI lists}) \]  

\[ \mu_{\text{PCL}} = \frac{1}{\text{CPU time}} \sum_{t=0}^{t_{n}} \text{CPU utilization} \]  

The text also mentions the use of tools like GNU time and m allocate to capture system performance information and discusses the integration of these metrics into the DPI analysis. The effectiveness of such identification methods is evaluated, especially in the context of securing CPS devices against spoofing or unauthorized access.
region, and (2) they must perform stationary deterministic operations inside the CPS infrastructure over time. The first rule guarantees that the device’s metrics and the type of activity (i.e., the device’s specific application inside the CPS network) performed by the ground-truth device are both considered to define its class (Section II). On the other hand, the second rule guarantees reliability. S&F requires that ground-truth devices behave in a deterministic way to guarantee that, if the same challenge is applied, every device class always generates the same signature over time. Steady behavior constitutes a realistic requirement since previous research works have highlighted the deterministic behavior of CPS [19]. Finally, the mechanism of obtaining signatures from ground-truth devices is known as learning phase. Once S&F completes this process, it stores the ground-truth device-class signatures into a signature database (SDB).

Ground-truth devices characterize a CPS network region. As discussed in Section II, device-class classification has been traditionally focusing on the branch, model, and the specific application of the devices. We also utilize these metrics to perform preliminary classification of the potential ground-truth devices and to determine how many different classes of devices may be present in the network. We also utilize this information to better organize signatures in the SDB. Thus, we first group the different classes of devices in the network based on device-specific metrics. Further, we apply behavioral and performance analysis to extract the signatures from every device class. To evaluate the reliability of the ground-truth devices, we calculate the autocorrelation of different PCLs and DPs obtained while the devices execute the same process (i.e., challenge) but at different time intervals. We use Equations 5 and 6 to calculate PCL and DPI autocorrelation, respectively, as follows:

\[
\rho_{PCL_{i}, PCL_{i+1}} = \frac{\sum PCL_{i} PCL_{i+1} - \bar{PCL}_{i} PCL_{i+1}}{\sqrt{\sum PCL_{i}^2 - \bar{PCL}_{i}^2}}
\]

\[
\rho_{DPI_{i}, DPI_{i+1}} = \frac{\sum DPI_{i} DPI_{i+1} - \bar{DPI}_{i} DPI_{i+1}}{\sqrt{\sum DPI_{i}^2 - \bar{DPI}_{i}^2}}
\]

where \( PCL_{i}, PCL_{i+1}, DPI_{i}, \) and \( DPI_{i+1} \) represent PCL and DPI metrics extracted from the same CPS process, but executed at different time intervals, \( n \) represents the size of the arrays PCL and DPI, and \( s \) represents the standard deviation.

Algorithm 1 details the process of obtaining the ground-truth signatures during the learning phase. Initially, in Line 1, the number of iterations (for averaging purposes) is defined, and the local variables \( PCL_{lists} \) and \( DPI_{lists} \) are declared and initialized. These variables contain the list of parameters (i.e., PCL and DPI) from every iteration \( i \) learning iteration. The goal of running several iterations of the challenge on the ground-truth devices is two-fold. First, we can study the behavior of the devices over time (devices with a random behavior that would not guarantee a stable signature would be rejected). Second, we compute and average all the data from all the iteration to so we consider potential fluctuations of the device behavior on the final signature. Going back to Algorithm 1, in Lines 5 and 6, system and function call tracing techniques are applied to obtain the PCL and DPI at different time intervals.

To calculate autocorrelation between different challenge iterations, S&F converts the PCL lists into random variables \( R_{PCL} \). To do so, the framework assigns weights \( \delta_{PCL} \) to every different type of function and system call in the PCL. Stop-and-frisk follow a specific pattern to assign the weight values, which depends on the type of system and function call in the PCL. For instance, if the PCL list that characterizes the device behavior to the challenge contains sys calls of type malloc and free, the weight assigned to these calls are desire to have certain statistical relationship to preserve the correlation among these calls. In regular system tasks, malloc and free calls are frequently invoked as part of the same process. Thus, S&F tries to preserve such a correlation by assigning numerical weight values that are also correlated (e.g., \( \delta_{PCL_{malloc}} = 2 \) and \( \delta_{PCL_{free}} = 4 \)). The result of the weight assignment is a random variable \( R_{PCL} \) that takes values between \( \delta_{min} \) and \( \delta_{max} \), and that statistically describes the reaction (i.e., behavior) of the CPS device class to the challenge. The rest of the collected parameters (i.e., calls arguments, call amount, and DPI information), are considered without modification for the autocorrelation calculation as they constitute numerical values.

In Lines 9 and 12, the autocorrelation vector between the different time intervals of PCL and DPI is calculated. Later, in Lines 14 and 15, the average of all autocorrelation values is computed. Finally, if the autocorrelation values of PCL and DPI are greater than the threshold \( \zeta \) (Line 16), the algorithm accepts the evaluated CPS device as ground-truth and stores its signature into the SDB (Line 17). In practice, the value of the threshold \( \zeta \) is agnostic and can be determined based on the specific characteristics of the operations in the ground-truth device.

E. Signature Correlation and Decision - Prediction Phase

During decision, S&F correlates the signature obtained from unknown CPS devices against the ground-truth signatures
Algorithm 2 Identify Device Class (Prediction Phase)

1: \text{CP

Algorithm 2 Identify Device Class (Prediction Phase)

1: \text{CP} \text{SignList} \leftarrow \text{SDB}
2: \text{iterations} \leftarrow N
3: \text{PCL lists}, \text{DPI lists}, \text{CPSdeviceID} \leftarrow \text{null}
4: \text{signature} \leftarrow \text{null}
5: \text{for} \ i = 0 \text{ to iterations} - 1 \ \text{do}
6: \text{PCL lists}[i] \leftarrow \text{getParamList}()
7: \text{DPI lists}[i] \leftarrow \text{getDPIindex}()
8: \text{end for}
9: \text{signature} \leftarrow \{\mu(\text{PCL lists}), \mu(\text{DPI lists})\}
10: \text{corrXYmax} \leftarrow 0
11: \text{for} \ i = 0 \text{ to size(\text{CP} \text{SignList})} - 1 \ \text{do}
12: \text{corrXY} \leftarrow \rho_{x,y}(\text{CPSSignList}(i), \text{signature})
13: \text{if} \ \text{corrXY} > \delta \text{ and } \text{corrXY} > \text{corrXYmax} \ \text{then}
14: \text{CPSdeviceID} \leftarrow i
15: \text{corrXYmax} \leftarrow \text{corrXY}
16: \text{end if}
17: \text{end for}

STOP-AND-FRISK algorithm for CPS device class identification.

stored in the SDB. This process is known as prediction phase and is detailed in Algorithm 2.

1) Signature Correlation: The process for obtaining the signature of the unknown CPS device follows similar steps as in Algorithm 1. However, this time the system is not required to calculate autocorrelation, as S&F assumes that all devices in the network are capable of generating a valid signature (Lines 2, 6, and 7 in Algorithm 2). Once the unknown signature is finally generated in the server (Line 9), S&F calculates the correlation between signature and all the unique CPS ground-truth signatures from the SDB (Line 12) using Equation 7 and Equation 8:

\[
\rho_{\text{PCL}X,\text{PCL}Y} = \frac{\sum_{i=1}^{n} \text{PCL}_X(i) \cdot \text{PCL}_Y(i) - n \cdot \mu(\text{PCL}_X) \cdot \mu(\text{PCL}_Y)}{\sqrt{\text{nsPCL}_X \cdot \text{spCL}_Y}}, \tag{7}
\]

\[
\rho_{\text{DPI}X,\text{DPI}Y} = \frac{\sum_{i=1}^{n} \text{DPI}_X(i) \cdot \text{DPI}_Y(i) - n \cdot \mu(\text{DPI}_X) \cdot \mu(\text{DPI}_Y)}{\sqrt{\text{nsDPI}_X \cdot \text{spDPI}_Y}}, \tag{8}
\]

where \(n\) represents the size of \(\text{PCL}_X\) (i.e., ground truth PCL), \(\text{PCL}_Y\) (i.e., unknown device PCL), \(\text{DPI}_X\) (i.e., ground-truth’s DPI), and \(\text{DPI}_Y\) (i.e., unknown device’s DPI), \(\mu(\text{PCL}_X), \mu(\text{PCL}_Y), \mu(\text{DPI}_X), \mu(\text{DPI}_Y)\) represent the mean value, and \(\text{spCL}_X, \text{spCL}_Y, \text{spDPI}_X, \text{spDPI}_Y\) represent the standard deviation, respectively.

After computing both \(\rho_{\text{PCL}X,\text{PCL}Y}\) and \(\rho_{\text{DPI}X,\text{DPI}Y}\) correlations, the decision process starts. The logical condition in Line 16 evaluates that (1) the correlation between the unknown device and signature \(i\) from the database is over a certain threshold \(\delta\) and (2) this value of correlation is a maximum obtained from all the iterations in Algorithm 2. If such a condition holds, the unknown CPS device is deemed to be the same CPS device class as CPS device \(i\) from the database (Line 14). On the other hand, if the condition in Line 13 is never satisfied, the unknown device is classified as \text{Unknown}, and flagged by S&F.

2) Decision: One can observe that, from Algorithms 1 and 2, the value of the correlation threshold \(\delta\) is a configurable parameter that can be inferred based on the threshold \(\xi\) used to generate the ground-truth device signature. That is, the value of \(\xi\) that achieves the highest accuracy during the learning phase is selected as \(\delta\) value for the prediction phase. With this, we guarantee the highest accuracy for S&F decisions. In addition, since the signatures are generated as a result of the device’s response to a controlled stimulus (i.e., challenge), S&F’s approach minimizes potential decision errors due to signature deviations. A device’s response that clearly deviates from a known signature should be considered as from a different device (known or unknown), as opposed to a signature error due to different random processes running in the device. The signature generation process hocks into the specific challenge execution and does not consider other processes running in the device. Also, as we explain in Section IV-D, we extracted the ground-truth signatures after averaging different learning iterations. With this, we guarantee that the final device signatures considers “expected” deviations that may occur to the device signature over time, due to random kernel and operating system operations.

Finally, the S&F’s processes described in Algorithms 1 and 2 are summarized in Figure 4. One can notice that, for both the learning and prediction phases, S&F reuses the first two modules of the proposed architecture since they contain similar operation steps. In both phases, S&F needs to extract features and create signatures from ground truth or unknown devices, respectively.

F. Space- and Time-Complexity Analysis

We analyze the time and space complexity of S&F’s algorithms, which highly depends on their specific implementation strategies. For instance, the correlation calculation can be performed either in time or frequency domains, which would differently impact their overall complexity. In addition, highly efficient correlation implementations can be used to reduce the use of computational resources and time [25], [26]. We perform analysis on the space- and time-complexity of S&F’s implementation (see Algorithms 1 and 2). We do not consider the encryption and signing steps used to guarantee the secrecy and freshness of S&F’s operations, but the
mathematical operations and analysis performed during data extraction, signature generation, correlation, and decision (see Figure 4). Recall that within the host, S&F only collects device data to generate the corresponding signature. The worst case performance scenario would create and query lists of \( n \) PCL and DPI elements, yielding an upper bound time and space complexity of \( O(n) \). On the other hand, the space complexity depends on the terms needed to calculate the correlation. For devices with storage limitations, it is possible to only save the value of the average calculations, thus yielding an overall space complexity of \( O(n) \). On the server side, the two most complex operations, autocorrelation and correlation, have similar time and space complexity. In total, considering PCL and DPI datasets of size \( n \), S&F computes the correlation calculations within control flow statements of \( n \) iterations. These calculations can be executed in constant time, yielding an upper bound time complexity of \( O(n) \). On the other hand, the space complexity depends on the terms needed to calculate the correlation. For devices with storage limitations, it is possible to only save the value of the average calculations, thus yielding an overall space complexity of \( O(\log M) \); where \( M \) would be the maximum value within the PCL and DPI lists. In this case, however, it would be necessary to spend more time in calculating a higher number of subtractions. In a different approach, the algorithm could store the average values and the differences, allowing for a faster computation but yielding an upper bound space complexity of \( O(n \log(M)) \).

V. IMPLEMENTATION DETAILS OF S&F

In this section, we assume a realistic CPS network and present implementation and deployment details of S&F. Also, we detail the key characteristics of the devices included in the testbed.

A. Realistic CPS Testbed Implementation

We implemented a CPS testbed considering the characteristics of the CPS device classes described in Section III (Table I). These characteristics were included in our testbed as follows:

1) Diversity in Hardware and Software Resources: We included 11 different classes of CPS devices with a variety of available computing resource and different hardware/software configurations. This diversity makes our testbed representative of a large population of real CPS devices; from small devices with limited resources to resource-rich devices [8], [20]. Despite the expected diversity, we allow certain similarities among the different device classes. We explain later in this section how we use these similarities to challenge the identification capabilities of S&F.

2) Discriminate then Regroup: We expect the proposed classification technique to be effective in discriminating different classes of CPS devices (to avoid false positives outcomes). However, we also expect S&F to be capable of grouping devices of the same class together (to avoid false negatives outcomes). To evaluate both metrics, we allowed more than one device of the same class in some cases (Table I). With this, we may determine how S&F performs on classifying types of devices into different classes and devices of the same type into a common class.

3) CPS-specific Tasks and Processes: During the learning phase, the devices included in our testbed performed real CPS networking operations following the IEC61850 communication standard [27]. The IEC61850 is a protocol-suite that defines the communication standards for electrical substation automation systems. To implement this functionality, we utilized an open-source version of the IEC1850 standard (i.e., libiec61850 that is freely available online [22]. In addition, we designed challenge-response approaches for the prediction phase that are also suitable for CPS operations (e.g., extract specific device information, write it into a file, and further delete the file from memory).

4) Multiple But Similar OSes: The CPS devices included in the testbed run 11 different versions of Linux-based OSes. Using different versions of Linux constitutes a realistic approach since most of the legitimate CPS devices used in the field include some variant of Unix-based OS [28]. Additionally, the open-source approaches in S&F enable the implementation of flexible solutions that would not impact the evaluation process of S&F. Indeed, previous research works have detailed system and function call hooking techniques that can be applied to all major operating systems [24]. In fact, obtaining the PCL and DPI data from devices with different operating systems is independent of S&F’s architecture and more related to specific implementation challenges. Finally, despite their noted differences, we purposely kept similarities among the different devices classes in the testbed. For instance, as shown in Table I, most of the devices are Debian-based systems using ARM CPU architecture. Such an implementation approach would additionally challenge S&F into identifying device classes based on small deviations of software and hardware-based features instead of taking advantages of very noticeable architectural differences.

<table>
<thead>
<tr>
<th>Class</th>
<th>Device Id</th>
<th>Hardware Specifications</th>
<th>Operating System</th>
<th>Release</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BB1</td>
<td>AM335x Cortex-A8 @ 1GHz 512MB D3 RAM</td>
<td>Linux Beaglebone 4.15</td>
<td>Debian 8.3 jessie</td>
</tr>
<tr>
<td>2</td>
<td>BB2</td>
<td>AM335x Cortex-A8 @ 1GHz 512MB D3 RAM</td>
<td>Linux Beaglebone 3.8.13</td>
<td>Debian 7.9 Wheezy</td>
</tr>
<tr>
<td>3</td>
<td>BB3</td>
<td>AM335x Cortex-A8 @ 1GHz 512MB D3 RAM</td>
<td>Linux Beaglebone 3.8.13</td>
<td>Debian 7.9 Wheezy</td>
</tr>
<tr>
<td>4</td>
<td>G2</td>
<td>AMD G41/100A @ 2GHz 1GB D3 RAM</td>
<td>Linux Ubuntu 4.4.0</td>
<td>Ubuntu 16.04 xenial</td>
</tr>
<tr>
<td>5</td>
<td>LAF1</td>
<td>A10 Cortex-A8 @ 1GHz 512MB D3 RAM</td>
<td>Linux A10/Live 3.4.90</td>
<td>Debian 3.4.90</td>
</tr>
<tr>
<td>6</td>
<td>LM2</td>
<td>A10 Cortex-A8 @ 1GHz 512MB D3 RAM</td>
<td>Linux A10/Live 3.4.90</td>
<td>Debian 3.4.90</td>
</tr>
<tr>
<td>7</td>
<td>ODR</td>
<td>A15 Cortex Quad core @ 2GHz A7 Quad core @ 900MHz 2GB D3 RAM</td>
<td>Linux Odroid 3.10.96</td>
<td>Ubuntu-Mate 16.04</td>
</tr>
<tr>
<td>8</td>
<td>OP1</td>
<td>H3 Quad core Cortex-A7 @ 1GHz 2GB D3 RAM</td>
<td>Linux Orange Pi Kali 3.4.39</td>
<td>Kali 2.0</td>
</tr>
<tr>
<td>9</td>
<td>OP2</td>
<td>H3 Quad core Cortex-A7 @ 1GHz 2GB D3 RAM</td>
<td>Linux Orange Pi Kali 3.4.39</td>
<td>Kali 2.0</td>
</tr>
<tr>
<td>10</td>
<td>RP1</td>
<td>Cortex-A7 @ 900MHz 1GB D3 RAM</td>
<td>Linux Raspberry Pi 4.1.7</td>
<td>Raspbian 8.0 jessie</td>
</tr>
<tr>
<td>11</td>
<td>RP2</td>
<td>Cortex-A7 @ 900MHz 1GB D3 RAM</td>
<td>Linux Raspberry Pi 4.1.7</td>
<td>Raspbian 8.0 jessie</td>
</tr>
<tr>
<td>12</td>
<td>RP3</td>
<td>Cortex A53 Quad core @ 1.2GHz 1GB D3 RAM</td>
<td>Linux Raspberry Pi 4.4.11</td>
<td>Raspbian 8.0 jessie</td>
</tr>
<tr>
<td>13</td>
<td>RP4</td>
<td>Cortex A53 Quad core @ 1.2GHz 1GB D3 RAM</td>
<td>Linux Raspberry Pi 4.4.11</td>
<td>Raspbian 8.0 jessie</td>
</tr>
<tr>
<td>14</td>
<td>RP5</td>
<td>ARM1176 @ 700MHz 512MB D3 RAM</td>
<td>Linux Raspberry Pi 4.1.13</td>
<td>Raspbian 7.0 wheezy</td>
</tr>
<tr>
<td>15</td>
<td>LTP1</td>
<td>Intel Core i7-2760QM @ 2.4GHz 6GB D3 RAM</td>
<td>Linux 3.19.0</td>
<td>Ubuntu 14.04 xenial</td>
</tr>
<tr>
<td>16</td>
<td>LTP2</td>
<td>Intel Core i5-5200 @ 2.2GHz 6GB D3 RAM</td>
<td>Linux 4.4.0</td>
<td>Ubuntu 16.04 xenial</td>
</tr>
</tbody>
</table>
For fairness, we followed a black-box approach where an independent third-party assigned specific labels (i.e., IDs) to the devices in the testbed based on their hardware and OS characteristics. Table I details the results of the labeling process. For instance, devices with CPU of type AM355x Cortex-A8 1GHz clock and 512MB of DD3 RAM, which also has installed the same type of OS (Linux Beaglebone 3.8.13) are considered as from the same “Class #2” (e.g., BB2 and BB3). However, devices with the same hardware characteristics but different OS are considered as from different classes (e.g., BB1 receives the label “Class #1”). With this labeling strategy, we demonstrate that S&F is capable of fine-grained classification, where devices with very similar OS and hardware characteristics can fall into different classes as their behavior differ. The rationale behind including many different devices classes in the testbed is justified as CPS infrastructures may contain a high diversity of devices that behave differently, but that share similar software and hardware characteristics. Also, the hardware and software characteristics of the devices may vary considerably in real scenarios. Specifically, in the CPS infrastructure, one can find devices with limited and rich computing resources, various software configurations, and different architectures. In this context, the hardware and software characteristics of the CPS devices are specific to their functionalities and applications. As a consequence, small changes in the CPS devices’ configuration should be highly noticeable in their general behavior [8]. In this work, we exploit these characteristics of CPS networks and devices to propose a fingerprinting technique that identifies different CPS device classes based on their behavior and performance.

B. Secure Deployment of S&F

We guarantee secure deployment and implementation of STOP-AND-FRISK via device attestation [29]. Figure 5 depicts a high-level representation of the end-to-end attestation process implemented to support S&F. First, a centralized scheduler running in the remote S&F server sends a secure request R containing the specific challenge C_H to the unknown CPS device. The request is secured via the use of encryption, digital signature, and timestamps. On the one hand, the use of encryption prevents passive observers (i.e., eavesdroppers) from learning the structure and content of the challenge. On the other hand, digitally signing the requests allows the S&F device to verify that the challenge comes as part of a legitimate S&F request that has not been illegally modified (i.e., preserving integrity). Finally, the use of timestamps prevents attackers from using old S&F requests to implement replay attacks. The encryption step of S&F uses a combination of symmetric and asymmetric encryption to protect the challenge and the device responses from external attackers. First, the requester generates a session key S that is used to encrypt the challenge C_H. Then, the session key is also encrypted, along with a timestamp T using the public key PB of the targeted host, and appended to the C_H. Finally, the entire request is signed using the targeted host’s public certificate. Once the CPS device receives the request, it is re-directed to the isolated secure environment where the host S&F resides. There, the signature of R is verified, and the part of the request containing the session key is decrypted using the private key of the device host PK. Then, the rest of the request corresponding to the challenge C_H is decrypted using the session key. We use software and, where possible, hardware-based isolation to prevent unauthorized access and modification of S&F’s modules. As we consider Linux-based devices in our CPS testbed, we first set up standard UNIX access control mechanisms, POSIX capabilities to limit access to the isolated area. Further, we ran the S&F’s modules in Virtual Machines (VM) hosted in hypervisors implemented using the Kernel-based Virtual Machine (KVM) [30]. We envision that, in cases where the architecture of the CPS device allows, S&F may take advantage of the principle of Platform Security Architecture (PSA) featuring Trusted Execution Environment (TEE) and TrustZone [31] to guarantee software isolation via hardware separation.

Once the CPS host processes the challenge, it generates a response R_C that contains specific system and function calls triggered by the challenge in the host CPS device. These calls are collected to create the PCL. In addition, the device’s hardware reaction to the challenge is collected to generate the DPI. The PCL and DPI information are combined in a form of a response R_C to S&F’s challenge, which is encrypted and signed using the same session key S and the public key and certificate of the STOP-AND-FRISK server. Finally, the response R_C is sent back to the S&F server which processes it and decides if the host device class is authorized or not.

VI. PERFORMANCE EVALUATION

In this section, we present experimental results that demonstrate the effectiveness of S&F to fingerprint and identify different classes of CPS devices. With this performance evaluation, we aim to answer the following research questions:

• RQ1: Learning Phase. How the proposed framework performs during the learning phase? (Section VI-B).

• RQ2: Prediction Phase. What is the accuracy of S&F in fingerprinting CPS devices while using (1) PCL correlation only, (2) DPI correlation only, and (3) both PCL and DPI analysis simultaneously? (Section VI-C).

• RQ3: Overhead. What is the overhead introduced by S&F to the CPS devices? (Section VI-D).

In all the evaluation experiments, we computed the results after averaging 30 different runs for all the covered scenarios. Every scenario comprised the execution of the challenge-response process. Further, we applied Algorithm 1 and Algorithm 2 on the devices included in
our testbed (Table I) to (1) generate a trustworthy signature database, (2) evaluate the correlation between the signatures and the devices' behavior, (3) identify different classes of CPS devices, and finally, (4) evaluate the overhead that S&F introduces to the CPS devices' computing resources.

During the learning phase, we studied the PCL and DPI behavioral characteristics of the CPS devices included in the testbed. We expect the devices to have a PCL and DPI behavior that is deterministic enough to guarantee repeatability (i.e., test-retest reliability) during the signature generation step. To evaluate determinism, we calculated the statistical autocorrelation among signatures extracted from the same device while it executed similar CPS processes at different time intervals. For the cases where a deterministic behavior was identified, the devices were accepted as ground-truth. Finally, for this phase of the evaluation, we set the threshold \( \xi = 0.7 \) in Algorithm 1 (Section IV), which marks the point from moderate to strong statistical autocorrelation that is widely accepted in the literature [32].

Further, S&F applied the challenge-response approach discussed in Section IV to extract data and create the devices' signatures. These signatures were then stored in the SDB for identification purposes during the prediction phase. For the prediction phase, we also set the threshold \( \delta \) to 0.7. In real-life scenarios, this value of \( \delta \) may be adjusted depending on the specific behavioral characteristics of the devices in the CPS network. For instance, in practical applications of S&F, the analysis over a group of well-known devices (i.e., control group) may give the best decision threshold value for the specific network region. Finally, since we are working with UNIX-based OSes, we utilized library interposition [33] and ptrace function [34] to extract the lists of system and function calls, respectively and generate the PCL. Also, we utilized the top and GNU time commands to extract information related to execution time as well as CPU and memory utilization for the DPI analysis. Finally, we used the same challenge to trigger responses from all the tested devices. That way, S&F detects and flags the differences between devices classes based not on the CPS tasks they process, but only on their relative differences in specific behavior of kernel and hardware performance. Finally, for statistical evaluation of the PCL, we converted the list of system and function calls into random variables. We followed the process of assigning \( \lambda_i \) weights to every specific type of system and function call. To maintain the statistical correlation among similar processes, we assign close weight values to system and function calls that are related to a similar process. For instance, we may assign the values \( \lambda \) and \( \lambda + 1 \) to system calls of the type malloc and calloc, respectively.

### A. Performance Metrics

To evaluate the performance of S&F, we compute the standard statistical metrics of accuracy, recall, precision, and specificity. We define these metrics as follows:

\[
A_{CC} = \frac{(TP + TN)}{(TP + TN + FP + FN)},
\]

\[
R_{EC} = \frac{TP}{(TP + FN)},
\]

\[
S_{PEC} = \frac{TN}{(TN + FP)}.
\]

where \( TP \) stands for true positive or the case where a CPS device is correctly classified as of some specific class; \( TN \) stands for true negative or the case where a CPS device is correctly classified as of not from some specific class; \( FP \) stands for false positive or the case where a CPS device is identified using the wrong signature. Finally, \( FN \) stands for false negative or the case where a CPS device whose signature has been previously stored in the database cannot be correctly identified.

### B. Performance of S&F During the Learning Phase

As described in Section IV, the first step towards applying S&F is to find a reliable set of unique signatures that characterize the different CPS device classes. The signature generation process uses statistical autocorrelation between different realizations of PCL and DPI to determine if deterministic behavior can be inferred from different time interval realizations of a similar process in the devices. Moderate to high values of autocorrelation (typically over 0.7 [32]) indicate that the specific CPS device (which is assumed to be a trusted CPS device with no prior tampering or unauthorized components) can be used as ground-truth to create a reliable signature for its class.

Figure 7(a) depicts the evaluation results after applying Algorithm 1 (Section IV) over randomly selected devices from all the different classes included in the testbed. One can observe that, in all the cases, the autocorrelation values are over the threshold \( \xi \), which indicates a deterministic behavior of the devices over time. Again, we obtained these results after 30 different PCL and DPI runs in every device at different time intervals. These results constituted a strong indicator that ground-truth signatures can be obtained for all the devices in the testbed. Finally, once the ground-truth CPS devices were identified, we generated the signatures and stored them into the SDB.

### C. Performance of S&F During the Prediction Phase

The primary goal of S&F is to classify CPS devices to the right class based on similarities in OS’s behavior, hardware performance, and configuration. Additionally, S&F must be able to cluster devices from the same class effectively. Before executing the prediction phase, S&F securely sent a challenge to the unknown devices, collected its features, and created their unique signatures. S&F applied system and function call hooking techniques (i.e., library interposition and ptrace) to generate the PCL of the unknown devices. Similarly, S&F extracted the hardware performance features used to calculate the DPI of every single device. Once these processes were completed, the host-based portion of S&F (Figure 2) sent this information to the remote server for processing. The prediction phase was then initiated by applying Algorithm 2 (Section IV) to the collected data.

To thoroughly test the efficacy of S&F and evaluate the real contribution of every fingerprinting feature that we have chosen, we first analyzed the performance of the framework by using PCL- and DPI-based correlation only. Then, we evaluated how the results improved after combining both analyses.
Fig. 6. Evaluation of the experimental results after considering PCL-based correlation only: (a) accuracy, (b) precision, (c) recall, and (d) specificity. One can observe that, in some cases, lower accuracy results were obtained due to false positives among some device classes. These results were improved after combining PCL-based correlation with DPI analysis (Figure 9).

1) PCL-Based Correlation Analysis: Figure 6 depicts details of the performance evaluation metrics after applying PCL-based correlation only. S&F achieved lower accuracy values of slightly over 86% for devices RPi4, RPi2 and BB1. Also, it obtained accuracy results of over 94% for devices of classes GZ, BB3, LPT2 and BB2, respectively. We believe that these results are caused by similarities in kernel behavior from different classes of devices while processing the challenge. The metrics of precision, recall, and specificity were also affected by these false-positive events. Figure 7(b) represents the confusion matrix (NxN PCL-based correlation matrix) among all the device classes in the testbed for PCL-based analysis only. A darker color indicates a high correlation, while lighter colors indicate lower correlation values between PCLs from different devices. As per Table I, one should expect a total of 11 different CPS device classes based on the different computing resources and software/hardware configurations. However, in this case, S&F was only able to identify 9 out of a total of 11 different classes of devices. From Figure 7(b), one can observe that, for instance, the proposed framework mistakenly confused GZ and LPT2 as of the same class. Also, the devices BB1, BB2, and BB3 were wrongly grouped together.

2) DPI-Based Correlation Analysis: We calculated the DPI values for every CPS device in the testbed using the approach proposed in Equation 3. In Figure 7(c), we show the results of the DPI calculation. As observed, several DPI values from different devices were very similar, which negatively impacts the feasibility of using this feature for identification purposes. However, for some specific devices, DPI values were significantly different among device classes. To better understand this analysis, we further represent the obtained DPI values versus the average DPI of all the devices classes in the testbed in Figure 7(d). From this figure, it is clear that DPI analysis may not significantly contribute to discriminating devices from different classes. For instance, one can observe that devices from different classes like ODR and LPT1 are wrongly overlapping in Figure 7(d).

3) PCL and DPI Analysis Combined: We further combined both PCL- and DPI-based correlation analysis and obtained a new decision map in Figure 8. This time, S&F was able to identify 11 different device classes by avoiding the false positives events obtained in previous results. Also, performance metrics significantly improved after combining PCL and DPI, if compared with PCL-only analysis. In Figure 9, we detail the new performance metrics results. One can observe that all the different metrics achieved excellent results if compared with the values presented in Figure 6. For the four different metrics, S&F obtained excellent performance with values close to 100%. The excellent detection performance of STOP-AND-FRISK is a direct consequence of some of the design strategies of S&F’s architecture. First, the combination of two different features, the PCL and the DPI, into the analysis permits monitoring the behavior of the CPS devices across two different low-correlated dimensions. While the PCL provides information regarding how the OS and kernel handle the applications running on the devices, the DPI characterizes...
Fig. 9. Evaluation of the experimental results after considering correlation and device performance index for decision: (a) accuracy, (b) precision, (c) recall, and (d) specificity. One can notice how the overall metrics improved if compared results shown in Figure 6.

The hardware configuration used to support such applications. Second, S&F combines function and system calls into the PCL, increasing the sensitivity of the proposed fingerprinting technique for cases in which similar devices have different OS or kernel configurations. These configurations may define distinctive device classes with different permission levels for users and applications that S&F would not be able to detect without considering kernel- and application-level calls simultaneously. Third, the challenge-response strategy followed by S&F permits the analysis of the devices’ behaviors on a controlled environment, which minimizes errors introduced by differences in device tasking over time. Furthermore, we summarize the average performance of S&F for all considered threshold values in Table II. In addition to the already define metrics, we include True Positive Rate (TPR), False Positive Rate (FPR), True Negative Rate (TNR), and False Negative Rate (FNR). The results depicted in Table II demonstrate the effectiveness of S&F in fingerprinting CPS device classes with different behavior.

Finally, we compare S&F’s accuracy with the performance of other existing similar tools in Table III. Although there exist tools that combine software and hardware features to implement fingerprinting mechanisms, not all published works share their performance metrics. One can observe that despite some of the tools implement sophisticated analysis based on machine learning algorithms, S&F’s performance is competitive, if not better, in all the cases.

VII. DISCUSSION

In this section, we discuss the performance of S&F under different compromised scenarios.

A. Fingerprinting vs. Anomaly Detection

We propose S&F as a host-based signature-based fingerprinting mechanism capable of identifying unauthorized device
Babun et al.: CPS Device-Class Identification via Behavioral Fingerprinting: From Theory to Practice

The implementation and further evaluation of S&F (Sections V and VI) consider the behavior of CPS devices while reacting to known challenges that are securely sent from a centralized secure server. S&F’s signatures do not characterize the behavior of the devices while performing regular networking activities in the field. As explained before, the requests are signed, encrypted, and timed to prevent potentially compromised devices from uncovering the challenge. With the challenge, attackers would be able to modify the OS or kernel to generate a fake signature that could potentially bypass S&F’s analysis, hiding the real malicious behavior of the compromised device. Also, signing the requests makes it harder for potential attacker to perform illegal request which may also reveal the specific device-class signatures. Although the design approach of the proposed fingerprinting approach could also support anomaly-based intrusion detection systems (IDSs) [8], S&F does not monitor the devices’ behaviors while they are performing regular network operations. Thus, S&F does not consider specific attack vectors that may deviate the devices’ behavior from their expected signature [8], [20]. Detecting and classifying specific anomalies coming from compromised devices is outside the scope of this work. However, we analyze S&F’s performance under potentially compromised environments below.

B. S&F on Compromised Environments

We discuss on the performance of STOP-AND-FRISK on compromised devices. The main goal of S&F is to fingerprint CPS devices based on their behavior. Since we combine software and hardware characteristics of the devices into our analysis, there are several steps an attacker must go through to properly bypass S&F.

1) The Attacker Knows the Challenge: Since S&F is expected to run on a secure environment, the attacker must be able to break the encryption scheme and take control of the challenge. At this point, S&F relays on the secrecy of modern symmetric and asymmetric encryption schemes. Most modern microcontrollers incorporate low-power state-of-the-art encryption capabilities into their design [40]. Thus, it is nowadays feasible to perform long-key encryption on resource-limited devices, enabling the implementation of a secure S&F in practice.

2) The Attacker Controls the Device: Our threat model assumes that an attacker is able to introduce fake devices into a critical infrastructure. In this case, we should assume that the attacker has control over the device’s OS and apps and can generate fake PCL and DPI sequences. However, the attacker still needs to infer the device signature which resides in a secure server, outside the edge devices. Under these circumstances, the attacker’s only capability is to infer the exact behavior of the devices at both hardware and software levels to create a counterfeit signature that looks exactly like the ground-truth signature of the device class is being impersonated.

3) The Device Under Attack: An special case of the previous threat assumes that a legitimate device is compromised and the attacker is able to perform stealthy attacks. In this case, S&F would not be able to directly detect the attacks as the signatures only computes the device behavior while reacting to a specific challenge and not while the devices perform malicious activities. However, previous works have demonstrated that approaches similar to S&F can be used to detect stealthy attacks (e.g., stealing sensitive information), as the attacks would change the normal behavior of the devices [8]. S&F can be adapted to these scenarios if the right signature is collected. For this, the learning phase simply needs to study the behavior of the devices while their perform their expected operations over time, and not while reacting to a specific stimulus.

4) S&F Under Attack: It is possible for a legitimate device to get compromised. In this case, the attacker may be able to detect the S&F’s requests and modify the PCL and DPI parameters, causing the device not to correctly react to the challenge. As a result, S&F would flag the device as unauthorized and reject the device, potentially causing a Denial-of-Service (DoS) situation. Although the DoS attack is the result of S&F’s analysis, it is definitely positive to reject a compromised device from the network.

VIII. RELATED WORK

A. Device Fingerprinting

Device fingerprinting is an appealing research area that follows two main paths: device-class and device-host fingerprinting. In [41] the feasibility of large-scale host fingerprinting...
via motion sensors is analyzed with 90% accuracy. Other works use microscopic deviations in clock skews to identify specific devices [42]–[44]. However, these approaches are vulnerable to simple countermeasures and require the analysis of several network packets for accurate results. Authors in [45] use embedded acoustic devices (microphone and speakers) on smartphones to fingerprint individual devices. Even though they report accuracy values in the range of 98%, these results are only possible in close-range distances (0.1 meters). Similar research paths were followed in [46], [47] where frequency responses of devices’ speakers are used to identify individual devices. The authors in [48] use RF-based fingerprinting to identify attacks to key-less entry systems. Also, the work in [49] uses the inertial measurement unit sensors found in iOS and Android devices to create globally unique fingerprints. Similarly, the work in [50] performs authentication of different hardware based on checksum extracted from micro-architectural implementation differences. However, this work limits its evaluation to simulation environments. The work in [51] analyzes hardware imperfections from 3D printers to identify unlawful 3D printed products. The recent impact of the Internet of Things (IoT) has motivated researchers to look into mechanisms to fingerprint IoT devices. One interesting approach uses a stimulation-response mechanism to identify specific devices via magnetic signals emitted from the CPU module [52]. Finally, Channel State Information (CSI) has also been proposed to fingerprint WLAN devices [53].

As for the identification of different classes of devices, in [54], the authors propose a passive blackbox technique for determining the type of access point (AP) connected to a network based on its behavior. In [19], [55], the authors use time as a baseline for device type fingerprinting. In this case, the proposed fingerprinting methods are mainly based on (1) the response time to network-based interactions (cross-layer fingerprinting) and (2) the response time to physical operations (physical fingerprinting). Although their results are promising, the first approach highly depends on configurable network attributes like the level of priority of TCP messages and ACK implementation. Further, the second proposed method also depends on the SCADA system configuration. In different works, passive device-class fingerprinting is proposed by using the timing distributions between network packets as the fingerprinting features [38], [56]. In similar approaches applied to domains other than CPS, researchers propose the analysis of network dynamics to infer IoT device classes [37], [57]–[60].

### B. Function and System Calls

Several security approaches make use of system and function call analysis to regulate and monitor the behavior of specific applications [24], [61], [62]. For instance, researchers have proposed the use of system and function call analysis for the design of intrusion detection systems (IDS) [63], the identification of operating system functions [64], sandboxing [65], and the implementation of software portable packages. Also, some works have demonstrated that similar approaches are suitable for the classification of behavioral anomalies [66], [67]. Although these last works report high overhead introduced to the systems, other similar implementations are more lightweight [68].

### C. Behavioral Analysis

The authors in [69] use OS and hardware features extracted from computing systems to fingerprint browser users. In [35], behavioral (i.e., network traffic’s temporal features) and static features are combined to fingerprint mobile device apps. Other works report the use of the devices’ behavior as a response to stimulant network packets [70], [71]. In spite of their positive results, these types of fingerprinting techniques also come with some limitations. For instance, the proposed approaches only apply for specific types of network protocols (e.g., transport layer protocols like UDP, and TCP), or they are vulnerable to network dynamics such as WiFi channel characteristics and traffic delay. Also, the behavior of API functions in computer systems has been proposed to fingerprint specific devices [72].

The work in [36] proposes the utilization of sensor behaviors from mobile devices to create fingerprints used to authenticate users.

### D. Differences From Existing Works

In Table V, we further compare S&F against fingerprinting techniques discussed before based on specific design and implementation criteria. The existing works (1) require extensive analysis, (2) depend on network dynamics, (3) consider physical metrics, (4) consider OS/kernel metrics, or (5) yield high overhead. We selected these comparison features as we believe they directly impact the performance of the time-critical CPS infrastructure. For instance, as opposed to S&F, other fingerprinting techniques require extensive data analysis or depend on specific network metrics to achieve their results. Surprisingly, these solutions do not offer specific overhead performance analyses even though fingerprinting solutions that focus on the behavior of the network dynamics may impact the performance of the CPS infrastructure. Additionally, their effectiveness either depends on the network’s configuration, the analysis of extensive amount of data, or the observation of fingerprinting features over long periods. S&F is different from other discussed solutions since it is host-based and device-centered. Also, it does not require traffic monitoring, the study of the interaction between CPS devices and other network equipment, nor the need to overcome inevitable errors or overhead (e.g., latency) that can be introduced from changes in network dynamics. Our framework implements a signature-based device type fingerprinting mechanism that studies the behavior and performance of the CPS devices at both hardware and kernel levels. S&F utilizes a challenge-response approach where the devices perform standard CPS functionalities and operations. Finally, our technique achieves excellent identification results while introducing very little overhead to the CPS devices at downtime.

<table>
<thead>
<tr>
<th>Fingerprinting Tool</th>
<th>Reference</th>
<th>Requires Extensive Analysis</th>
<th>Utilizes Network Dynamics</th>
<th>Analyzes Physical Metrics</th>
<th>Analyzes OS’s Metrics</th>
<th>Yield High Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device Fingerprinting</td>
<td>[19], [38], [42], [54]</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Behavioral Fingerprinting</td>
<td>[35], [36], [69], [70], [72]</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>S&amp;F</td>
<td>-</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>

- Yes, ✗ - No, and ✗ - Not Discussed
IX. CONCLUSION

CPS critical infrastructure networks use different devices to collect data and monitor critical operations. However, these devices can be spoofed by attackers to get access to systems, steal information, or disrupt critical operations. Also, legitimate CPS devices may include unauthorized pieces of software and hardware that could degrade critical CPS tasks. In this paper, we presented STOP-AND-FRISK (S&F), a novel and lightweight CPS signature-based fingerprinting framework used to identify CPS device classes based on their behavior. Specifically, our novel approach combined system and function call tracing techniques, signal processing, and hardware performance analysis to implement a secure challenge/response-based device-class identification solution. Moreover, we evaluated the efficacy of S&F on a realistic testbed that included different classes of CPS devices with different hardware, software resources, and configurations. Our extensive experimental results demonstrated that S&F achieves an excellent rate in the identification of CPS devices classes. Also, our analysis revealed that the use of the proposed framework does not yield a significant overhead on the CPS devices’ computing resources.

ACKNOWLEDGMENT

The views expressed are those of the authors only, not of the funding agencies.

REFERENCES

A. Das, N. Borisov, and M. Caesar, "Do you hear what i hear?:
Z. Zhou, W. Diao, X. Liu, and K. Zhang, "Acoustic fingerprinting
T. Kohno, A. Broido, and K. C. Claffy, "Remote physical device
A. Das, N. Borisov, E. Chou, and M. H. Mughees, "Smartphone
F. Adamsky, T. Retunskaia, S. Schiffner, C. Köbel, and T. Engel,
K. Joo, W. Choi, and D. H. Lee, "Hold the door! Fingerprinting your
K. Gao, C. Corbett, and R. Beyah, "A passive approach to wireless
H. Aksu, A. S. Uluagac, and E. Bentley, "Identification of wearable

Leonardo Babun (Member, IEEE) received the Ph.D. degree in electrical and computer engineering from Florida International University in 2020. His research interests include cyber-physical systems (CPS) and the Internet of Things (IoT) security and privacy. He is currently a CyberCorps Scholarship for Service Alumnus and a member of the Cyber-Physical Systems Security Lab (CSL), Florida International University.

Hidayet Aksu received the Ph.D. degree from Bilkent University in 2014. His research interests include security for cyber-physical systems, the Internet of Things, security for critical infrastructure networks, security analytics, big data analytics, distributed computing, wireless networks, wireless ad hoc and sensor networks, localization, and P2P networks.

A. Selcuk Uluagac received the M.S. and Ph.D. degrees from the Georgia Institute of Technology. He leads the Cyber-Physical Systems Security Lab at the Florida International University, focusing on security and privacy of the Internet of Things and cyber-physical systems. He received the U.S. National Science Foundation CAREER Award and the U.S. Air Force Office of Sponsored Research’s Summer Faculty Fellowship. He currently serves on the editorial boards for the IEEE TRANSACTIONS ON MOBILE COMPUTING, Computer Networks (Elsevier), and the IEEE COMMUNICATIONS AND SURVEYS AND TUTORIALS.