ChargePrint: A Framework for Internet-Scale Discovery and Security Analysis of EV Charging Management Systems

Tony Nasr*, Sadegh Torabi†, Elias Bou-Harb‡, Claude Fachkha§ and Chadi Assi*
*The Cyber Security Research Centre, Concordia University, Montreal, Canada
†The Center for Secure Information Systems, George Mason University, Fairfax, VA, USA
‡The Cyber Center for Security and Analytics, University of Texas at San Antonio, San Antonio, USA
§The College of Engineering and IT, University of Dubai, United Arab Emirates and Steppa Cyber Inc., Canada
Emails: tony.nasr@mail.concordia.ca, storabi@gmu.edu, elias.bouharb@utsa.edu, cfachkha@ud.ac.ae, chadi.assi@concordia.ca

***Abstract*—Electric Vehicle Charging Management Systems (EVCMS) are a collection of specialized software that allow users to remotely operate Electric Vehicle Charging Stations (EVCS). With the increasing number of deployed EVCS to support the growing global EV fleet, the number of EVCMS are consequently growing, which introduces a new attack surface. In this paper, we propose a novel multi-stage framework, ChargePrint, to discover Internet-connected EVCMS and investigate their security posture. ChargePrint leverages identifiers extracted from a small seed of EVCMS to extend the capabilities of device search engines through iterative fingerprinting and a combination of classification and clustering approaches. Using initial seeds from 1,800 discovered hosts that deployed 9 distinct EVCMS, we identified 27,439 online EVCS instrumented by 44 unique EVCMS. Consequently, our in-depth security analysis highlights the insecurity of the deployed EVCMS by uncovering 120 0-day vulnerabilities, which shed light on the feasibility of cyber attacks against the EVCS, its users, and the connected power grid. Finally, while we recommend countermeasures to mitigate future threats, we contribute to the security of the EVCS ecosystem by conducting a Coordinated Vulnerability Disclosure (CVD) effort with system developers/vendors who acknowledged and assigned the discovered vulnerabilities more than 20 CVE-IDs.**

I. Introduction

Electric Vehicle Charging Management Systems (EVCMS) are a collection of specialized software that instrument the underlying Electric Vehicle Charging Stations (EVCS). They provide users/operators with advanced features, interfaces, and tools to remotely monitor, control, and manage the operations of their EVCS including scheduling charging sessions, billing/-payments, record-keeping, and user management/authentication [1]. Given the incentives that governments are offering to stimulate the purchase of EVs [2], the worldwide EV fleet is rapidly growing and currently includes 11 million units [3]. Similarly, the EV charging infrastructure is also expanding to support the growing client demands by deploying a large number of EVCS in public/private spaces [4]. This implies a growth in the number of deployed EVCMS to instrument the installed EVCS. As a result, the EV charging ecosystem's proliferation opens doors for a plethora of security challenges due to its large attack surface. Specifically, the diverse and mostly ad hoc vendor-specific development of EVCMS, in addition to the broad lack of regulations can expose the overall EV charging ecosystem to various threats. In fact, EVCMS constitute a major target for adversaries as they provide a remote attack vector for compromising the underlying EVCS, its charging operations and the connected critical infrastructure. Particularly, the impact of such attacks is amplified with its direct integration to the power grid [5].

Nevertheless, while prior work focused on investigating the security of the EV charging ecosystem [6], [7], there is a lack of knowledge about the security posture of the widely deployed EVCMS. Motivated by the large number of deployed EVCS and the extended remote capabilities of their EVCMS as a viable new attack surface, we aim at conducting a first exploration of this threat landscape by evaluating the security of the implemented EVCMS in the wild. This requires discovering and mapping the EVCMS instances and their hosts on the Internet at large, which is a challenging task with the lack of empirical data about the deployed EVCS. Additionally, unlike typical Internet-of-Things (IoT) devices, it is impractical to utilize existing fingerprinting approaches [8], [9] to accurately discover Internet-connected EVCS by identifying their EVCMS. This is mainly due to the fact that EVCMS are cloud-based and tend to have closed-source implementations. Moreover, while the deployed EVCMS have limited and non-trivial banners, they tend to have limited or no, search engine tags and banner rules for identifying them due to their diversity and lack of standardization among developers.

To address these challenges, we propose a multi-layer framework, ChargePrint, for fingerprinting and discovering Internet-connected EVCS and assessing the security of their EVCMS against remote exploitation. The framework relies on analyzing a seed of candidate EVCMS to extract identifiers from their banners, which are then utilized to conduct an iterative discovery/fingerprinting process by leveraging prominent online device search engines and a combination of classification and clustering approaches. Consequently, the framework is leveraged to perform an in-depth security analysis of the identified EVCMS and their host instances through a series of hybrid techniques, namely white-box analysis by examining the corresponding accessible EVCS firmware, and black-box analysis by examining online EVCMS instance endpoints. Additionally, with the outcome from our analyses using ChargePrint, we illustrate the feasibility of cyber attacks and discuss their implications against the EVCS, their users and operators, and the connected power grid. Further-

Network and Distributed System Security (NDSS) Symposium 2023
27 February - 3 March 2023, San Diego, CA, USA
ISBN 1-891562-83-5
https://dx.doi.org/10.14722/ndss.2023.23084
www.ndss-symposium.org

more, while we recommend mitigating countermeasures, we communicate our findings to the affected vendors to raise attention to the existing vulnerabilities. In fact, several system developers (e.g., Schneider Electric, Cornerstone technologies) have acknowledged our findings, and the vulnerabilities were assigned more than 20 Common Vulnerabilities and Exposures (CVE) IDs [10]. Finally, by demonstrating the feasibility of large-scale attacks against the EV charging ecosystem through vulnerable EVCMS implementations, we aim at motivating system developers towards re-evaluating and improving the security of their EVCMS and mitigating threats in future implementations.

We summarize the main contributions as follows:

1) We propose a novel multi-layer framework (ChargePrint) to tackle the problem of EVCS discovery/fingerprinting and address the lack of empirical data about EVCMS by gaining and leveraging information about their unique banners and characteristics Further, to demonstrate the effectiveness of our approach, we implement ChargePrint and utilize it at a large scale to discover 27,439 Internet-connected EVCS host instances that are instrumented by 44 different EVCMS products. To the best of our knowledge, we are among the first to survey EVCMS instances in the wild.
2) We provide a first attempt to perform a comprehensive security analysis of EVCMS, which defines a new attack surface in the EV charging ecosystem. We highlight major security flaws in EVCMS by implementing a hybrid security analysis approach into ChargePrint to examine the various vendor-specific EVCMS products both in terms of their firmware and their online instances. In addition, we customize automated scanning modules to conduct a large-scale EVCMS vulnerability analysis, which identified 25,300 EVCMS hosts affected by 120 remotely exploitable 0-day vulnerabilities. Moreover, we communicate our findings to the respective system developers who acknowledged them and assigned more than 20 CVE-IDs.
3) We shed light on the insecurity of the existing EVCMS at scale, by discussing attack implications against the EV charging stakeholders namely the EVCS, users/operators, and power grid. More importantly, we recommend mitigating countermeasures to strengthen the deployed systems against cyber attacks. We contribute to the security of the broad ecosystem by motivating the developers, through our reported findings, to improve the security of existing and future EVCMS products. Finally, we re-assess the EVCMS state of security in 2022 by following-up with the manufacturers on the released patches, and surveying their deployment in the wild.

The remainder of the paper is organized as follows. We present in the next section relevant background information, and a detailed description of the proposed framework in Section III. The empirical evaluation of the implemented framework along with the experimental findings, their threat implications on the EV ecosystem, and possible mitigating tactics are presented in Section IV. Vulnerability disclosure information and patch follow-up along with lessons learned and future research directions of the work are noted in Sections V and VI, respectively. Finally, we examine related work in Section VII, before concluding the paper by summarizing the main takeaways in Section VIII.

II. Background

EVCS vs. IoT. When compared to IoT devices and their embedded software, EVCS along with their EVCMS have many distinctive characteristics, despite common ones such as Internet connectivity for remote management.

First, EVCS are equipped with larger processing units and storage capacity than that of IoT devices, to keep up with the need for EVCMS to store and process charging records and activity logs, and to ensure their heavy performance requirement and continuous service to deliver electric energy to EVs throughout the day and for long periods of charging cycles.

Second, EVCS's power-electronics circuitry is designed to sustain much larger power supplies than that of IoT devices, as they are built to feed electric power to EV battery from the grid [11]. EVCS classify into 3 major classes based on their maximum amount of power delivery: level 1 provide power through a standard $120V$ AC outlet with a power output of 1.5-2 kW, level 2 provide power through a 208-$240V$ connection with peak power of 19 kW and average 7.2 kW power output, and level 3 provide charging through a 480-$800V$ AC input with peak power of up to 240 kW and on average 110 kW.

Third, while IoT devices rely on a variety of protocols (e.g., BLE, ZigBee, etc) to operate and deliver certain functionalities [12], EVCS rely on proprietary protocols that have been devised by operators to allow for communication between the various ecosystem entities as well as EVCMS, which is essential for managing and controlling the various operations and functionalities of EVCS. Specifically, to standardize this communication, Open Charge Alliance (OCA) designed and introduced in 2012 the Open Charge Point Protocol (OCPP) [13] considered the de-facto standard application protocol for EVCS message exchange.

Fourth, EVCS are designed with the intent for regular and direct interaction with users/operators, unlike IoT devices which when deployed in public places (e.g., CCTV) are not used at close proximity nor are they meant to be as frequently interacted with on a daily basis as part of their setup.

Fifth, EVCMS present a larger and much more complex software surface than most IoT device firmware. EVCMS encompass a wide number of functionalities, thus a wider code base and structure, designed to accommodate the various operations offered and required by EVCS. In particular, EVCMS are devised to organize in real-time and keep track of EV scheduling, as well as manage charging records, and aside from remote management, provide support for physical interfaces such as human machine interface (HMI) on the EVCS itself which also require to handle charging ID tokens like near-field-communication (NFC) cards. EVCMS also implement complex access control mechanisms, user authentication and management, user role segregation and privilege matrices, which are not available in IoT software which typically operate with a single or limited user/privilege. Moreover, while most IoT device firmware use standard packaging compression formats for delivery and device deployment such as LZMA and Zip, many EVCMS use custom and proprietary compression formats like EPK. We also note that while IoT device firmware use more common file system types such as squashfs, EVCMS use less common file system types such as JFFS2.

EVCMS Assets. EVCMS can be deployed following different models, among which the firmware-based and cloud-based are the most prominent. The firmware-based model is implemented by embedding the application User Interface (UI) document collection (i.e., web interface files) directly into the EVCS firmware, which is a minimized operating system (OS) designed to provide low-level control over the EVCS hardware. The cloud-based model is implemented on a web server in the cloud, which connects and communicates with the EVCS. In this work, we examine both firmware-based and cloud-based EVCMS software products and conduct an in-depth vulnerability analysis on various instances of them.

Fingerprinting and Discovery. In general, discovering Internet-connected devices relies on fingerprinting and banner analysis. Fingerprinting is the mapping of a device query:response to class labels such as the device type, while banner analysis is performing Internet-wide protocol scans (e.g., HTTP, SSH) and collecting application layer data (i.e., banners) to extract textual information and identify devices using a set of rules. In the literature, several frameworks have been introduced to discover and fingerprint Internet-facing devices [9], [14].

However, despite the promising results in identifying generic IoT and industrial control systems (ICS) remote management devices [8], these approaches are ineffective for annotating EVCS due to various reasons; (1) there are limited banners associated with online EVCMS; (2) it is difficult to locate information related to EVCMS as most of them are cloud-based or closed-sourced; (3) in contrast to the vast number of generic IoT device models, EVCMS specifications are harder to obtain in the absence of banner rules for identifying these systems using conventional fingerprinting techniques; (4) while ICS devices can be identified through built-in tags provided by device search engines, there are none for EVCMS; (5) EVCMS's diversity and lack of standardized development results in a wide range of banner representations that are hard to analyze and keep track of, which makes it difficult to extract and search for useful information about them due to the differences among the various products and their features. Such factors make it unfeasible to employ existing approaches to accurately and effectively locate and fingerprint EVCS. Therefore, we develop ChargePrint to overcome these challenges and propose an effective approach for EVCMS discovery and fingerprinting, which is a prerequisite for analyzing their security posture.

Device Search Engines. The majority of online device search engines collect information about Internet-connected devices/hosts by actively scanning the Internet (e.g., IPv4 address space) while performing consequent banner analysis to tag/label the identified hosts. Given such rich repository of host banners and identified device information, we rely on passive scanning (metascan–based) techniques by leveraging existing solutions through their Application Programming Interfaces (API) [15]. Specifically, we perform our experiments using the four most prominent device search engines: (1) Shodan [16]: performs continuous detection and monitoring to gather near real-time information about Internet-connected devices by indexing their service banner/metadata; (2) Censys [17]: monitors Internet-accessible devices by regularly probing public IP addresses and domain names; (3) Zoomeye [18]: collects information about online devices and services with an aim to provide threat detection at an Internet-scale; and (4) Fofa [19]: provides intelligence about Internet-connected device network assets, scope analysis, and application popularity statistics.

Vulnerability Analysis. We address the key gaps from the literature [20], [21] by providing the first attempt to analyze the security of firmware-based and cloud-based EVCMS products, specifically, the resilience of EVCMS against remote attacks. To perform this analysis, we utilize white-box techniques to examine obtainable EVCMS firmware and scripts, and black-box techniques for assets extracted from online EVCMS endpoints. We mainly focus on detecting and uncovering remotely exploitable EVCMS vulnerabilities that would allow access and control over the respective systems and its underlying EVCS. Particularly, we refer to OWASP [22] and MITRE for the top security issues [23], and for each we develop systematic methodologies to infer their existence in specific EVCMS.

III. Proposed Framework: ChargePrint

We design and implement ChargePrint, a multi-stage framework for exploring the EVCS threat landscape by discovering Internet-connected EVCS host instances and examining the security of their EVCMS. As illustrated in Figure 1, ChargePrint has two main core segments represented by the discovery and security analysis campaigns. We present the detailed internal architecture of ChargePrint, which consists of several interconnected components and layers that support iterative operations for discovery and security analysis.

A. Discovery

First, we conduct a discovery operation for finding and fingerprinting Internet-connected EVCMS host instances in the wild, then utilize them to extend knowledge of EVCMS products while paving the way towards identifying the attack surface for the security analysis. In what follows, we elaborate on the modules of the framework's discovery campaign.

1) Seed Storing: To bootstrap the discovery campaign, we build an initial database by collecting and storing EVCMS seeds from a preliminary lookup, in which we leverage indexed host scan data from querying Shodan, Censys, Zoomeye, and Fofa with a list of 23 keywords (e.g., EV charger, EVCS) related to EVCS and known EVCMS products. To validate these systems, we manually examine their web user interfaces banners for specific indicators such as product series/model and vendor name/logo, and correlate them with information from software package documentation of available respective EVCMS developers. Also, we select the hosts with the most information as initial set of EVCMS candidates, and store their banners into a database for analysis.

2) Identifier Extraction: In what follows, we elaborate on the analysis of seed EVCMS to extract identifiers that can be leveraged to distinguish among the various EVCMS products.

Collecting Assets. To obtain assets, we collect and examine EVCMS products that are either accessible through firmware packages, portals of online deployed web-based instances and those that are associated with a vendor/developer having their own website. We note that it is not always possible to obtain all these assets for a given EVCMS product, as some instances for

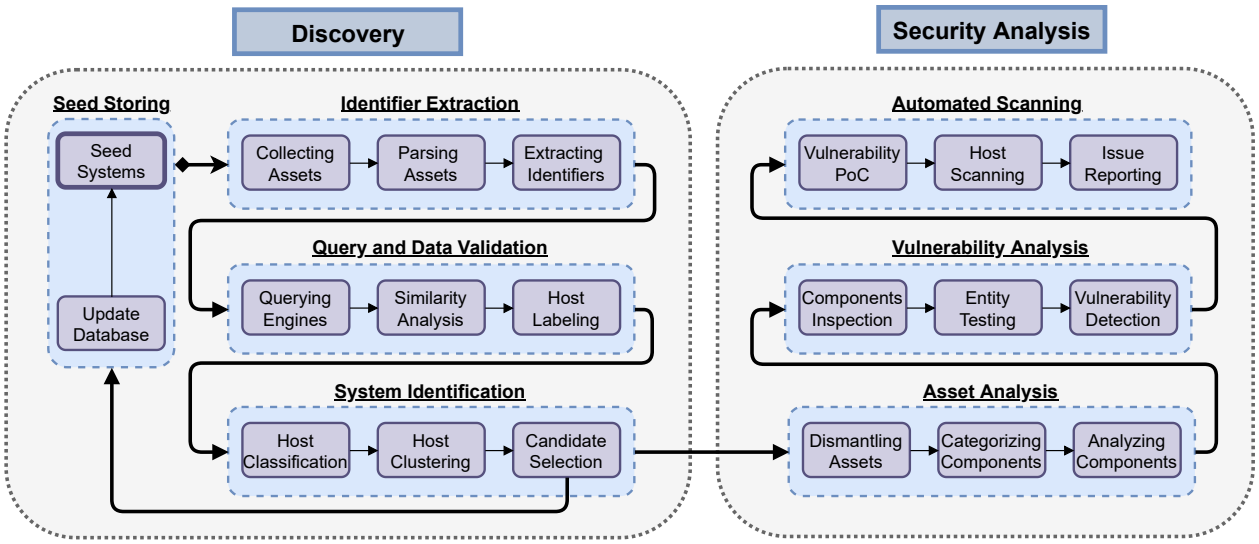


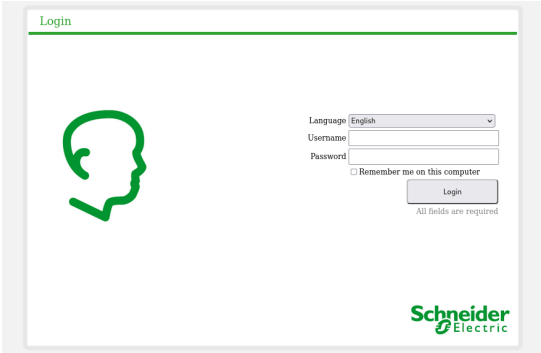
Fig. 1: Overview of ChargePrint and its main components/operations.

example, do not have their corresponding firmware available for download on the vendor/developer website.

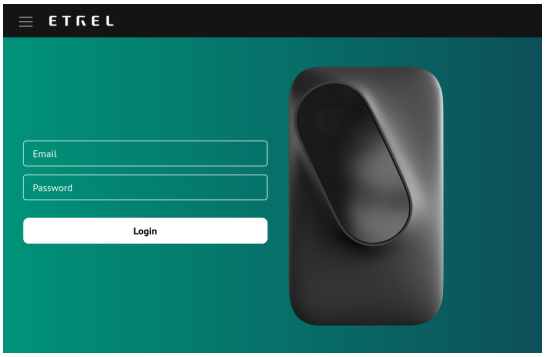
To collect EVCMS firmware, we query web search engines using dorks made of keywords such as product and developer name and website link, and package extension, in search for indexed websites belonging to the product vendor/developer. Then, we examine these websites' sitemaps for potential endpoints that could hold firmware (e.g., /download), by recursively scraping hyperlinks, using Scrapy [24], crawling web pages until a potential firmware acquisition endpoint is reached. To collect web-based EVCMS, we probe the host instances' main portal endpoints and download their web documents (i.e., HTML and CSS files) as well as the files of other accessible endpoints along with their language variants.

Parsing Assets. To parse the downloaded firmware, we extract their packages and explore the filesystem by carving and dumping the embedded files, binaries, and directories using a custom utility that we built with binwalk's API [25]. To parse the acquired portals, we examine the portal web documents by inspecting their document object model (DOM) structure. To parse the collected vendor websites, we crawl their web pages and accumulate the text strings enclosed within HTML tags.

Extracting Identifiers. Next, we extract a set of identifiers that allow us to accurately represent the respective EVCMS candidate and distinguish it from other products. To extract identifiers from firmware, we locate unique file/directory path names within the filesystem tree (e.g., /cgi-bin/cgiServer) which will serve as direct indicators for the EVCMS product. To extract identifiers from web portals, we search the HTML documents' DOM, using regular expressions, for special elements that act as indicators for the EVCMS such as the product name, version, vendor/developer name and logo. In Figure 2, we show the screen captures of the main portals for a few examples of EVCMS that were discovered with ChargePrint. As with the vendor/developer websites, we compile from the previously accumulated strings a list of EVCS-related text (e.g., EV Charging Solution) that enrich our EVCS terminology knowledge for the extended discovery of EVCMS



(a) EVlink



(b) CSWI Etrel

Fig. 2: Screen captures of the main web portals for (a) EVlink, and (b) CSWI Etrel EVCMS.

instances. As an example in Table I, we present a set of identifiers for a candidate running on EVlink EVCMS product.

3) Query and Data Validation: In this phase, we utilize the extracted identifiers for the given EVCMS products to discover host instances that that are instrumented by the corresponding EVCMS whose identifiers we already have, as well as dis-

TABLE I: Sample set of identifiers for EVlink.

Identifier	Value
Title	EVSE Web Interface
Name	EVlink
Version	3.3.0.12
Vendor	Schneider Electric
Logo	/images/schneider_head.png
Port	5001
Path	/cgi-bin/cgiServer
Parameter	worker, time, error, lang
Language	English (en), Deutsch (de)
Keyword	wallbox, ev charging solution

cover new EVCMS products by utilizing combinations of the extracted EVCS string wordlists.

Querying Engines. We utilize the extracted identifiers, following two methods, with the objective to broaden the search scope and perform an extended system collection by querying device search engines to find more host instances instrumented by the currently known EVCMS products and new EVCMS candidates. First, we perform a targeted search using the extracted product-specific identifiers to filter hosts that have their banners matching the queried information. Second, we perform a generic search for EVCS-related banner host data using the wordlists that were compiled from the extracted EVCMS strings/keywords found on the vendor/developer websites.

Similarity Analysis. Since our targeted and generic search results may contain a variety of hosts that belong to different EVCMS, we require further validation before associating them to existing previously detected or new EVCMS. Thus, we introduce a new similarity measure (DOMetric) to compare a given host's HTML pages (H) with known EVCMS candidates (C) from the database to identify host instances that are running services assigned to the same EVCMS product family based on the similarity of their services and interfaces; since host instances running variants of the same EVCMS product would have identical or highly similar web documents.

To determine the similarity score using DOMetric, we start by parsing the EVCMS portals' HTML page to extract their structure, style, and text content. We determine structural similarities $D_1(H, C)$ by performing pair-wise comparison using the Gestalt pattern matching method [26] on the sequence of tags in the HTML pages, then we leverage Eq. 1 to find the Longest Common Sequences (LCS) of tags between those of an identified host (S_1) and known EVCMS candidates from the database (S_2). For style similarity $D_2(H, C)$, we collect embedded style declaration blocks and selectors of common tags from documents' HTML and find the largest amount of common declarations between the two documents, A (host) and B (candidate EVCMS), using the Jaccard index (Eq. 2). Lastly, we determine text similarities (D_3) by vectorizing the enclosed text within the tags for the host (T_1) and candidate (T_2) in the order they appear in the HTML, then comparing them using the cosine similarity index (Eq. 3).

$$D_1(H,C) = \frac{2 \times \sum_{i=0}^{max(|S_1|,|S_2|)} |LCS(s_{1i}, s_{2i})|}{|S_1| + |S_2|} \quad (1)$$

$$D_2(H,C) = \frac{\sum_{i=0}^{m} \frac{|a_i \cap b_i|}{|a_i \cup b_i|}}{m := min(|A|,|B|)} \quad (2)$$

$$D_3(H,C) = \frac{\sum_{i=0}^{m} \frac{t_{1i} \cdot t_{2i}}{|t_{1i}| \times |t_{2i}|}}{m := min(|T_1|,|T_2|)} \quad (3)$$

Host Labeling. With these similarity measures, we calculate the total DOMetric score for each pair of host (H) and candidate EVCMS (C) using Eq. 4, where w is a scaling factor (e.g., $w = \frac{1}{3}$). The remaining unlabeled hosts are kept for further investigation using the subsequent binary classifier.

$$DOMetric(H,C) = \sum_{i=1}^{3} w.D_i(H,C) \quad (4)$$

4) System Identification: In this phase, we detect host instances that belong to already known EVCMS products (i.e., having seed candidates in the database), and determine instances that are new EVCMS products from which we select candidates for the database as well as remove generic (IoT) devices that do not represent EVCMS host instances.

Host Classification. With the large number of unlabeled hosts from the extended queries and data validation, we employ a binary classifier that leverages 14 features (Table II) to determine whether these hosts are EVCMS.

TABLE II: Dataset features for the binary classifier.

Name	Description	Type	Value
auth	Presence of authentication form	Bool.	0/1
settings	Presence of settings form	Bool.	0/1
evbrand	Presence of EV keywords	Bool.	0/1
evbrand_cnt	Number of EV keywords	Int.	[0...N]
evcskey	Presence of EVCS keywords	Bool.	0/1
evcskey_cnt	Number of EVCS keywords	Int.	[0...N]
transport	Type of network protocol	Cat.	TCP/UDP
http	Usage of HTTP	Bool.	0/1
logos_cnt	Number of images	Int.	[0...N]
tags_cnt	Number of HTML tags	Int.	[0...N]
styles_cnt	Number of CSS selectors	Int.	[0...N]
strings_cnt	Number of text strings	Int.	[0...N]
dometric	Highest DOMetric score	Real	[0...1]
system	EVCMS of highest DOMetric	Cat.	$[C_1 \ldots C_n]$

We evaluate four common classifiers (logistic regression, k-nearest neighbors, naive bayes and support vector machine) by extracting features from a dataset of 1,000 host instances (ground truth) that we manually examined and labeled, having 500 EVCMS and 500 non-EVCMS hosts, partitioned into training (70%) and testing (30%) for validation purposes. Based on the evaluation metric results presented in Table III, we select the logistic regression (LR) algorithm as the preferred classifier model as it outperforms the other models with significantly higher accuracy (93.1%) and F1-Measure (94.2%) values. However, for future work, the classification outcome could be improved by evaluating a wider range of machine learning models.

Host Clustering. To identify new products among the EVCMS instances that were identified through binary classification, we design and implement Algorithm 1 to cluster newly identified

TABLE III: Evaluation of binary classification algorithms.

Classifier	Accuracy	F1-Measure	Precision	Recall
Logistic Regression	**93.1%**	**94.2%**	89.2%	99.9%
k-Nearest Neighbors	86.1%	88.8%	81.2%	98.1%
Naive Bayes	79.3%	84.4%	73.3%	99.5%
Support Vector Machine	66.1%	86.8%	62.6%	99.5%

Algorithm 1: Clustering EVCMS host instances

```
Input: β = List of instance tuples {i, p, cId, cnd}
Output: β′ = Cluster assignment list
newcId = 1
for x ∈ β do
    if x.cId ≠ 0 then
        continue
    end if
    for y ∈ β do
        if x.i = y.i then
            continue
        end if
        if y.cId ≠ 0 then
            if DOMetric(x.p, y.p) > α then
                x.cId = y.cId = newcId
                larger = maxContent(x.p, y.p)
                larger.cnd = 1
                newcId++
            end if
        end if
        else if y.cnd = 1 then
            if DOMetric(x.p, y.p) > α then
                x.cId = y.cId
                larger = maxContent(x.p, y.p)
                if larger.i = x.i then
                    x.cnd = 1
                    y.cnd = 0
                end if
            end if
        end if
    end if
end for
end for
```

EVCMS hosts into correlated groups that belong to the same EVCMS products, based on their page HTML similarity using the DOMetric score (recall Eq. 4). The algorithm takes a list of instance tuples as input (β) and returns an updated list $\beta\prime$ having the respective cluster assignments. The tuple for each instance contains an index *i* for referencing and identifying it among other instances, the portal HTML content *p*, the cluster identifier *cId* which by default is initialized to 0 indicating that the host has not been assigned to a cluster yet, and a boolean flag *cnd* indicating whether this instance is a candidate. We define a function *DOMetric()* that calculates the DOMetric score of two web pages and a function *maxContent()* that compares two web pages to determine the one with larger content and newer product version string.

Selecting DOMetric Threshold. For the algorithm to accurately correlate hosts into the same clusters, an effective DOMetric similarity threshold value (α) must be selected. Based on our observations, deployed systems belonging to the same product family always have a very similar structure and interface, with the exception of differences that arise due to product version variants that integrate new components. EVCMS product version variants are instances of a specific EVCMS product family whose builds are different due to release distinctions, thus, while these variants are different than each other, they still share very similar structure and filesystem, and belong to the same EVCMS product family.

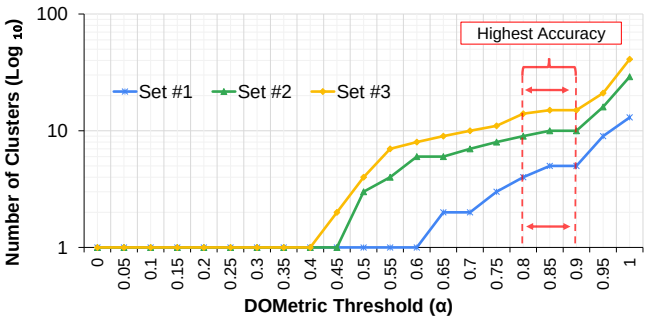


Fig. 3: Finding DOMetric threshold via experiments with Algorithm 1.

Therefore, we have to select an appropriate threshold that accounts for the differences among the distinct EVCMS product families as well as version variants belonging to the same product family. To determine an optimized DOMetric threshold value, we conduct several experiments by running Algorithm 1 with a span of DOMetric threshold values (α) that range from 0 to 1 inclusive, over three custom datasets (i.e., Set #1, Set #2, and Set #3), containing 500, 2,000, and 3,500 of EVCMS product host instances, and made of 5, 10, and 15 EVCMS product families, and 13, 29, and 41 EVCMS product version variants, respectively.

As shown in Figure 3, we note three trends with the plot lines. First, when running the algorithm with low α values ($\alpha \leq 0.8$), we observe that the number of clusters remains at 1 and then steadily increases. Since these cluster numbers are incorrect, this indicates that lower threshold values lead to a large overlap among the examined systems; any compared systems that are significantly different (i.e., belong to different EVCMS product families) will be grouped together due to the low clustering condition (small α), which ultimately renders the clustering process erroneous. Second, when running the algorithm with α values ranging from 0.8 to 0.9, we observe that the number of clusters for Set #1, Set #2, and Set #3 are almost the exact values pertaining to the correct number of clusters that are found in each. In fact, the threshold value should be motivated by the need for a high similarity rate among the analyzed/compared system host instances. This corresponds to a grouping of systems that share significant similarities. Therefore, we selected the highest DOMetric similarity threshold value ($\alpha = 0.9$) in order to maximize the accuracy of the number of correct clusters obtained with Algorithm 1. Third, when running the algorithm with α values that are higher than 0.9 and up to 1, the outcome yields a larger number of clusters for each set, which represent clusters of EVCMS product version variants.

Candidate Selection. Once the clusters are identified, we examine them and select representative candidates (new seed

systems) for each cluster. We determine the corresponding candidates by choosing the host instances that deploy different versions of the respective EVCMS product and whose content/configuration is larger than the remaining instances (recall function *maxContent()* in Algorithm 1). Subsequently, we store the EVCMS candidates into the seed EVCMS database which we then utilize to perform a new iteration of the framework's discovery campaign. This iterative approach allows ChargePrint to extend the fingerprinting and discovery process by identifying a wide range of host instances that deploy the same products of the candidates from the EVCMS database, which includes, with respect to the iteration, both new and old systems. For each framework iteration, we perform the new search using the seed EVCMS candidate whose assets were collected in the first stage of discovery.

B. Security Analysis

With the newly obtained EVCMS product candidates (seeds), we conduct an in-depth security analysis using a series of systematic methodologies, as highlighted in Figure 1.

1) Asset Analysis: We analyze the EVCMS product assets by examining their components, which represent the attack surface for uncovering vulnerabilities.

Dismantling Assets. To conduct the security analysis campaign, we dismantle and dissect the collected assets for further analysis. For dissecting the firmware images, we dump and mount the embedded filesystem to explore the directories/-files that they contain, specifically, we search to locate the EVCMS document collection which contains the EVCMS web service files. These files provide various interfaces and controls containing entry points to the EVCMS that are accessible to the user/operator as well as to an external adversary. To dissect the assets of candidate EVCMS instances with no firmware images, we determine the main web user interface service of the EVCMS along with all other available web paths, and uncover resources/scripts from which we gather forms & parameters. Moreover, we create offline mirrors of the respective EVCMS host instances by downloading web document collection such as web directories & JavaScript files.

Categorizing Components. To organize the analysis process, we arrange the obtained components into three categories: compiled files (e.g., binaries, executables), non-compiled files (e.g., scripts) and web endpoints. We extract the compiled files from the EVCMS products' firmware images, while we gather non-compiled files from both the firmware images and the web portals of the EVCMS host instances, extract web endpoints from the document collection of the EVCMS firmware images as well as the candidate host instances. In particular, we extract endpoints from available firmware through custom scripts which utilize tree-listing and pre-compiled custom word dictionaries, by searching the filesystem and identifying the web document collection directory and recursively locating valid server back-end and front-end files that represent endpoints, then parse them to extract parameters.

Analyzing Components. We perform two types of analysis: white-box and black-box on the gathered components. We utilize white-box procedure to examine the EVCMS components to which we have direct and complete access to such as binaries and scripts. Specifically, we perform static analysis (i.e., disassembly) on compiled server-side binaries (e.g., `cgiServer`) and conduct source code review on non-compiled client-/server-side files and scripts (e.g., JavaScript, PHP). The disassembly and decompilation processes are automated through specialized software (e.g., Cutter [27]), while the corresponding code paths and execution flows are reviewed manually, since the discovered vulnerabilities are unique per EVCMS product and design flow. The source code review process is automated through custom scripts that we wrote to search for potentially vulnerable functions and entry points, then we conducted a manual inspection and validation, after the automation has narrowed down the analysis surface. We utilize black-box analysis to investigate the EVCMS components to which we do not have direct access to such as the endpoints that were identified on the EVCMS host instances' portals whose back-end binaries and scripts are inaccessible to us, due to the inability to obtain related firmware images.

2) Vulnerability Analysis: As the discovery of EVCMS vulnerabilities is not possible by simply employing commercial scanners, we develop a collection of custom modules, scripts, tools, and testing procedures. We also utilize manual efforts to correctly engage the EVCMS architecture as this requires in-depth investigation, extensive security domain expertise and secure programming experience to identify security bugs and efficiently direct the investigation process.

Components' Inspection. When analyzing compiled binaries through disassembly and decompilation, we focus on tracing their execution flow to identify bugs and business logic flaws. As when reviewing source code files, we focus on finding entry points that lack input cleansing and validation allowing injection and execution of arbitrary code and queries, which is the main objective to achieve for an adversary to take control of the system. For instance, when inspecting client-side JavaScript files in the EVCMS document collection, we locate vulnerable functions that failed to properly sanitize data variables, allowing arbitrary JavaScript execution within the context of the affected EVCMS, subverting its logic and functionalities to attack the users/operators by hijacking their sessions and the system.

For instance, in Listing 1, vulnerable function `goToTerminal` does not implement mechanisms to sanitize data passed through variable `termNum`, which is directly employed at lines 3/6/8, allowing an attacker that controls the data to pass and execute malicious code, breaking the legitimate sequence and overriding the EVCMS logic flow. Finally, when investigating the collected endpoints, we intercept HTTP request/response traffic using Burpsuite [28] to find insertion points (e.g., GET/POST parameters).

Entity Testing. The firmware images and their underlying entities are available/found for some EVCMS products, but not for the others. Thus, a large portion of the codes and binaries that operate these EVCMS are inaccessible during the study. Instead, we heavily rely on black-box testing techniques for conducting the security analysis, by detecting vulnerabilities with crafted approaches on the endpoints of these EVCMS host instances whose back-end logic is relatively unknown. The process consists of tracing both random and crafted data into the discovered entry points (e.g., inputs, parameters) to elicit an unexpected outcome, while finding indications of vulnerabilities. For that, we generate payloads that are tailored

```
function goToTerminal(termNum) {
 var menuTabs=parent.frames.menuTabs;
 menuTabs.document.getElementById('selectedDescSlave
    ').value='descSlave'+termNum; var ip;
 if (isRouter)
  ip=document.getElementById('ipRoute'+termNum).
     value;
 else
  ip=document.getElementById('ip'+termNum).value;
 var div=menuTabs.document.getElementById('MenuTabs'
    );
 var paramItemSelected='';
 if (div.title=='EVSE')
  paramItemSelected=getMenuItemSelected(menuTabs); }
```

Listing 1: Weakness in JavaScript function leads to EVCMS code execution.

for the specific testing context of the analyzed EVCMS, and run these lists on the corresponding entry points using custom tools to collect the returned output for examination.

Vulnerability Detection. We examine the EVCMS products for the OWASP's top software weaknesses [23] by utilizing custom testing procedures that we tailored to detect occurrences of these specific classes of vulnerabilities. To detect SQL injection (SQLi), we utilize sqlmap [29] to inject sleep delay queries on EVCMS endpoints/parameters. To detect external XML entity injection (XXE) and server-side request forgery (SSRF), we append in EVCMS HTTP request message/parameters/endpoints crafted XML entities and addresses that contain callback to a server that we control. To detect hard-coded credentials, we examine the EVCMS firmware filesystem for embedded credentials. To detect Comma-Separated Values injection (CSVi), we inspect the EVCMS functionalities and endpoints by supplying crafted CSV payloads and examining the response.

Moreover, to detect cross-site scripting (XSS), we inject string-marked JavaScript payloads into HTTP request insertion points (i.e., parameters, endpoints, headers), then parse the responses to validate the injection. To detect cross-site request forgery (CSRF), we inspect EVCMS `GET/POST`-based configuration/settings requests for the absence of request-specific randomized tokens. To detect Cross-Origin Resource Sharing (CORS) and Flash Cross-Domain Policy (FCDP) misconfigurations, we inspect EVCMS endpoints for permissive rules in their cross-domain policy files. To detect information exposure, we parse EVCMS endpoints for sensitive information related to the underlying EVCS and EVCMS settings/configurations. To detect forced browsing and missing authentication, we inspect EVCMS functionalities by validating their access control mechanisms, and searching for logic flaws to request post-authentication endpoints/resources without login. To detect missing rate-limit, we send a cycle of HTTP requests to the EVCMS endpoints then compare the returned responses to determine if they contain differences.

3) Automated Scanning: To make the security analysis efficient, we build automated scanning modules (Figure 1).

Vulnerability Proof-of-Concepts (PoCs). Once we detect vulnerabilities within the respective EVCMS product candidates, we craft custom vulnerability-specific PoCs for each, which consist of a request containing vulnerability payload and an anticipated response with indicators that prove the vulnerability, and we correlate the scanned host instance product model and version strings with those of the examined systems. This allows to minimize the number of requests to send, and explore indexed Internet scans with limited host interaction, by relying on the data and details already collected by the third-party device search engines like Shodan. Thus, whenever a set of vulnerabilities is linked to a specific EVCMS product model and version numbers, we configure the PoC to extract the scanned host instance version string from the HTTP banners, then correlate it to the details associated with the vulnerable product within ChargePrint's database. This allows to prove the existence of vulnerabilities on a given EVCMS product, and determine automatically the number of host instances instrumented by the EVCMS that are affected.

Host Scanning. By iterating over the collected database of candidate host instances that represent the various clusters of EVCMS products, we leverage the generated vulnerability PoCs to determine vulnerable instances automatically. For every candidate EVCMS in the database, which represents a given cluster of products, we craft a collection of vulnerability PoCs to determine whether other host instances belonging to the same cluster are vulnerable.

To report the exact number of Internet-connected EVCMS host instances that suffer from the discovered vulnerabilities, we perform a targeted scan on each of these hosts by sending a precise number of PoC requests and parsing the responses for indicators to count the events when the tested vulnerability is confirmed. We build this automated scan on the basis that host instances instrumented by a given EVCMS product must run similar or identical builds and services, hence, will suffer from the same vulnerability classes. Thus, instead of re-conducting the thorough analysis for each host instance that deploys a variant of a given EVCMS product, we conduct the in-depth security analysis on fewer instances, while generalizing our discoveries across other similar hosts.

Ethical Considerations. As with scan-based security measurement studies, we abide by best practices [30]–[35]. In particular, we took several steps to ensure no harm towards the operation of the examined EVCMS.

Non-repudiation: In our scans, we set a custom user-agent string within the outgoing requests to signal benign intentions and supply contact information to permit system owners to communicate with us and have the option to be removed from our study. We also provided reverse DNS records to our machines' public IP addresses to allow targets to obtain additional information about the study.

Correlated Scans and Non-Exploit Payloads: To perform large-scale scans, we relied on previously collected Internet scan data by third-party device search engines (e.g., Shodan, Censys), and we utilized non-exploit payloads that yield no real-life exploitation and active interaction. We used side-channel techniques to infer vulnerabilities by carefully crafting proof-of-concepts, that leverage version strings and banner correlation, to verify the existence of vulnerabilities without causing any damage or persistent effects to the examined EVCMS, analogous to how device search engines, such as Shodan, determine if hosts are affected by a specific CVE vulnerability. Additionally, we developed our own approach for

clustering the large number of hosts into versioned products such that the analysis is conducted on the representative candidates or corresponding firmware, and finding issues in specific versions allows correlating to all related hosts, without interacting with them.

Client-Side Analysis and Test Environments: We note that out of 13 discovered vulnerabilities, 8 are client-side which do not affect nor cause any impact on the analyzed EVCMS as they were uncovered through offline client-side scripts analysis. As for the remaining 5 server-side vulnerabilities, they have been discovered through firmware analysis and coordinated testing in environments provided by the vendors, notably Cornerstone Technologies, Bluesky Energy and Etrel. We also validated that the PoCs do not cause harm to the analyzed EVCMS by communicating with local and national EV operators running a comprehensive list of vendor-specific EVCMS and testing the PoCs in a controlled environment while performing monitoring tasks of such assets, and we did not observe any damaging outcome. While we did not test our PoCs on all the international vendors that we scanned for the existence of vulnerabilities, we did work with few (e.g., Cornerstone Technologies, Bluesky Energy) where we utilized vendor-provided techniques and tools to verify that the asset is intact. We note that the PoCs contain passive payloads which do not exploit the vulnerabilities, and at worse-case scenario any unexpected side-effects can be averted by restarting the EVCMS or EVCS. As an extra precaution, when applicable, we leveraged de-facto remote monitoring tools with minimal polling on remote hosts to continuously validate the performance, integrity and availability of the remote systems and running services, and we acted upon the outcome from these procedures, which did not reveal any issues. Moreover, we coordinated with network administrators and IT leadership at our institution, as well as with our upstream ISP, to ensure that our scans do not adversely impact network operations.

Responsible Disclosure: We executed a systematic Coordinated Vulnerability Disclosure (CVD) [36] effort and promptly communicated with the impacted vendors, reporting the vulnerabilities to them at least 6 months prior to writing this paper, to give them plenty of time to notify their respective users/operators while preparing for the security fix.

Data Privacy: We obtained an approved institutional review board (IRB) based on data retention/management policy in regards to the gathered data and IP addresses. Specifically, we retained data collected from the EVCMS host instances for the duration of the analysis, after which, to preserve the privacy of data and reliability, we removed from our machines, all data gathered during the study of the affected host instances.

IV. Experimental Results

We implemented ChargePrint, and we present here the results of its discovery and security analysis campaigns.

A. Discovering Internet-Connected EVCS in the Wild

1) Initial Search: The initial search queries using the selected engines resulted in identifying 1,800 hosts that are running 9 distinct EVCMS products, and as illustrated in Figure 4, the initial search produced a significantly larger number of verified hosts using Zoomeye and Fofa (about 1,700) as compared to Shodan and Censys (about 200 hosts). These significant differences are attributed to two main factors. First, each engine employs distinct scanning tools and techniques for device discovery and probing, which yield differences in the identified hosts and banner data, for instance, Shodan and Censys rely on ZMap and ZTag [37] while ZoomEye and Fofa rely on their own proprietary scanners. Second, each engine implements different customized lookup queries that associate the search constructs with the stored host information, for instance, to optimize the lookup and avoid searching the entire data, the search engines may associate a given host with device or system-specific tags (e.g., device type, OS) that are extracted from their banners. We note that none of the selected search engines have defined EVCS-related tags, and therefore, significantly hampering the outcomes of the initial search.

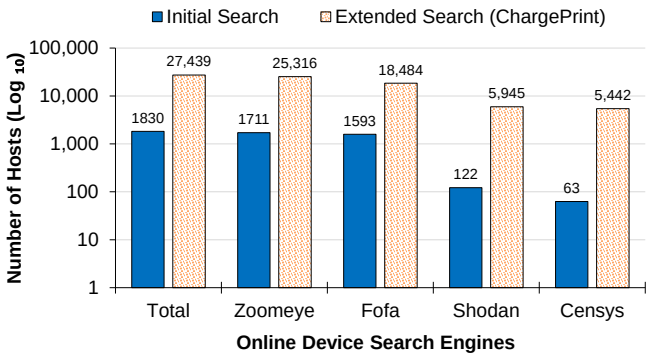


Fig. 4: Number of discovered hosts in initial & extended search with ChargePrint.

2) Extended Search Using ChargePrint: Motivated by the limited number of identified EVCS hosts from the initial search, we leverage ChargePrint to perform an extended and iterative lookup/query for EVCS hosts that are managed by EVCMS. As illustrated in Figure 4, our extended queries using ChargePrint produced a significantly larger number of hosts, as compared to the initial lookup, specifically, we utilized the candidates for the initial 9 EVCMS products to identify a total of 27,439 unique host instances instrumented by various EVCMS. By leveraging ChargePrint, we improved the EVCMS lookup outcome using all search engines, demonstrating its effectiveness, with a significant increase in the number of identified host instances that reached almost double the tenfold with Zoomeye (25,316), the tenfold with Fofa (18,484), and quintuple the tenfold with Shodan (5,945) and Censys (5,442), as presented in Figure 4.

We note that the total 27,439 EVCMS host instances were discovered through 5 iterations of the framework's discovery and fingerprinting campaigns. In addition, our analysis showed that these host instances are instrumented by 44 unique EVCMS products, which represent the number of groups obtained, and manually verified as valid EVCMS, during the host clustering phase, as discussed in Section III-A4. This highlights the importance of ChargePrint's iterative fingerprinting approach, which not only extends knowledge of the total number of Internet-connected EVCS hosts, but also discovers a wider range of deployed EVCMS products in the wild. We present the full list of 44 discovered and analyzed EVCMS

TABLE IV: List of analyzed vulnerable EVCMS.

EVCMS	Manufacturer	Vulnerable Version(s)
Ensto CSI	Ensto	4.63-6011
Emonscms	Open Energy Monitor	9.8.10
xChargeIn	Eaton Corporation	M_SW, A-X-S (3.10.7)
ICEMS	Bluesky Energy	1.8.0
EVlink	Schneider Electric	R7 V3.3.0.17
Heliox	Heliox Energy	Diag-1
Lancelot	Unicorn Systems	v4.9
Turnkey EVCI	ChargeLab	v2.7
Smartfox	Smartfox	12.3.4p
Mein	–	v2020
CSWI Etrel	Etrel	2.2.0.5
OpenEVSE	OpenEVSE	7.1.3
SuperChargeI	–	v2019
Suncountry	Sun Country Highway	sc1
OASIS Portal	OASIS	v2020
Syzygy EVLab	Syzygy	v1.5
Magtec	Magtec	v2020
Nuvve	Nuvve	v2020
KANDI EVS	Kandi Technologies	–
EVsmart	EnelX	Version 1.3.0.1
EVCSControl	EVCSControl	1.20.0
Whirlybird	WHIRLYBIRD Electronics	–
kosiarka	–	–
BaSE EVMS	Cornerstone Technologies	v2020
IoCharger CCMS	IoCharger	2.8
EV-Algorigo	Algorigo Inc.	v2020
MikEVSE	MikEVSE	–
Kia EV Portal	–	–
NCR	National Chargepoint	–
FCEIS	Fuzhou Comprehensive Energy	v2020
BetaSmart	Betasmart	v2020
ChargedHK	Hong Kong EV Association	–
GAP EVDAS	GAP	–
LFEV	LF Link	v1.12
Revitalize	Revitalize Charging solutions	
EVmob	ALTernative	–
SharedEnergy	DSM	6.2
DSmob	DataSpace Ecosystem	v2020
Irasus	Irasus Technology	v3.1.0
Greenwai	Greenwai	v1.0
Zonnigladen	Zonnigladen	–
USHybrid	US Hybrid Corporation	–
EMotorWerks	EMotorWerks	1.3
Fcevspat	–	v2021

products along with their manufacturers and up-to vulnerable versions in Table IV.

3) Geographical Distribution: The identified EVCMS hosts are distributed across 21 countries, with Hungary, Finland, U.S.A, France, and South Africa having a significantly larger number of hosts (about 78%), as compared to the remaining countries (Figure 5). However, this distribution does not fully comply with the number of deployed EV chargers worldwide [3], where countries such as the U.S.A and the U.K. are supposed to host the largest numbers of EVCS, respectively. This bias is due to ChargePrint relying on a number of initial EVCMS candidates, which might be commonly deployed in certain countries. Additionally, while we identified various EVCMS products in each country (Figure 5), the majority of the hosts found in these countries correspond to one or two unique products only. For instance, we identified over 10,000 hosts that deploy Ensto CSI, with 90% of them located in Hungary (4,900 hosts) and Finland (4,100 hosts).

4) Ports and Services: We leverage the identified EVCMS host banners, and find the open ports that are used for running the EVCMS web interface service, as well as find ports associated with known services such as SSH. As illustrated in Figure 6, the majority of the identified EVCMS products were running HTTP(S) services on common ports (e.g., 80, 8080, and 443), however, we also found alternative ports that are configured for HTTP(S) services by various EVCMS products (e.g., 81, 82, and 8888). While open ports provide information about supported services on the identified EVCMS hosts, some combination of these ports can be used in future work as advanced vendor-specific features for targeted discovery and accelerated fingerprinting of some EVCMS products.

TABLE V: Overview of identified EVCMS vulnerabilities.

Severity	CWE	Vulnerability	# Issues	# EVCMS	# Hosts
Critical	89	SQLi	4	4	1,684
	611	XXE	5	5	1,290
	798	Hard-Coded Cred.	6	6	900
	918	SSRF	7	3	1,457
	1236	CSVi	1	1	1,203
High	79	XSS	29	19	7,754
	352	CSRF	12	9	7,789
	942	CORS Misconfig.	2	2	3,731
	942	FCDP Misconfig.	2	2	1,205
Medium	200	Info. Exposure	17	17	13,787
	306	Missing Auth.	3	3	1,005
	425	Forced Browsing	2	2	1,402
	799	No Rate Limit	30	30	17,500

B. Quantifying the Security Posture of EVCMS

The in-depth security analysis on the EVCMS product/host instances unearthed 120 vulnerabilities that belong to 13 Common Weakness Enumeration (CWE) [38] classes with critical, high, and medium severity (Table V), discovered across 25,300 host instances (about 92% of all hosts), and enabling remote exploitation of the EVCMS and control over the underlying EVCS. Additionally, as illustrated in Figure 7, 29 products, deployed on 13,989 hosts, were associated with high and/or critical vulnerabilities, and almost all the remaining EVCMS products that had medium-severity vulnerabilities, were deployed on a small number of hosts ($\leq$8), except Ensto CSI which was deployed on over 10,000 hosts. We note that about 8% of the verified EVCMS hosts were not associated with any vulnerabilities, which is due to the inability to examine their corresponding firmware and portal endpoints to perform in-depth security and vulnerability analysis. In what follows, we provide details and examples of the identified vulnerabilities across the analyzed EVCMS products and host instances:

1) Critical-Severity Vulnerabilities: As listed in Table V, we uncovered 5 critical-severity server-side vulnerabilities, which affect 7 unique EVCMS products (Figure 7) that instrument 4,431 EVCMS host instances (about 16%). We uncovered 4 occurrences of SQLi on 1,684 EVCMS host instances, which can allow full exploitation of the host by extracting the stored databases that contain tables with sensitive information such as the user account details and payment information. Furthermore, we discovered XXE and SSRF vulnerabilities, which allow adversaries to force the EVCMS into sending arbitrary requests to internal/external networks as well as exfiltrate data from the EVCMS. We also discovered that 900 and 1,203 EVCMS host instances suffer from hard-coded credentials and CSVi respectively, which would allow attackers to compromise the EVCMS by gaining direct access to the resources/configurations and uploading persistent payloads.

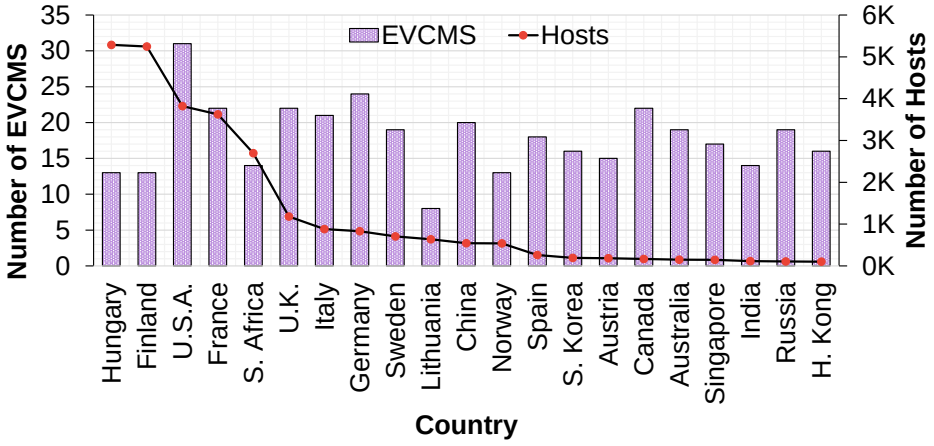


Fig. 5: Geographic distribution of discovered EVCMS hosts/instances.

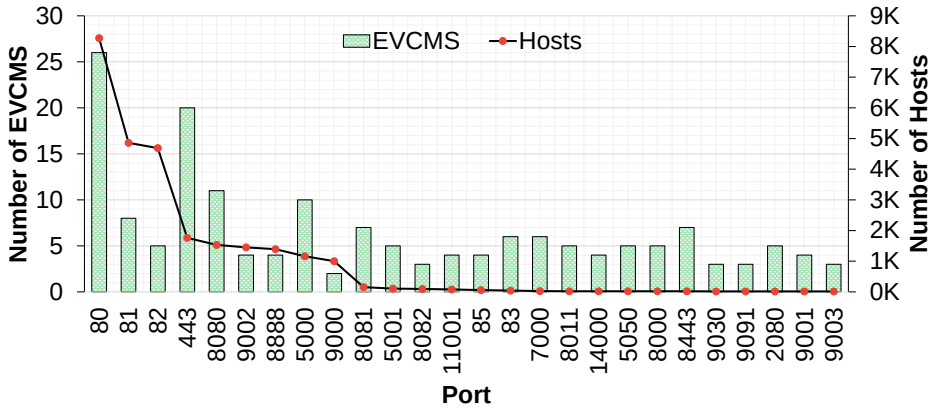


Fig. 6: Distribution of EVCMS ports across the instances.

2) High-Severity Vulnerabilities: We found 4 high-severity client-side vulnerabilities affecting 22 EVCMS products that are installed on 9,750 host instances (about 35% of all). The XSS vulnerabilities would allow an adversary to execute arbitrary JavaScript code within the context of the vulnerable EVCMS, hijacking user accounts and enabling the adversary to operate the EVCS by gaining control over all available functionalities. Moreover, XSS weaknesses can be leveraged for privilege escalation on the corresponding EVCMS by embedding persistent payloads or creating backdoors via injected web shells, that execute in the context of the system users, allowing an adversary to obtain administrator-level access to the EVCS by exposing session cookies. Moreover, we find CSRF vulnerabilities, which would allow adversaries to force a target user into performing unintended actions like changing the EVCS settings and configurations (e.g., restart the EVCS). Furthermore, we identify CORS and PCDP misconfiguration vulnerabilities, that enable adversaries to attack the EVCMS by exfiltrating account data and session cookies.

3) Medium-Severity Vulnerabilities: We discovered 4 medium-severity vulnerabilities, that affect 30 EVCMS products installed on 17,831 EVCMS host instances (65% of all), and which can open door to access partial privileged functionalities such as exposing maintenance endpoints through forced browsing, or allow the adversary to view confidential EVCS-related states and settings through information exposure vulnerabilities. In addition, the missing authentication and rate limit vulnerabilities would allow accessing specific resources/-functionalities on the EVCMS without affirming privilege.

C. Attack Implications on EV Stakeholders

Within a complex ecosystem of inter-playing entities on top of the EVCS, the exploitation of the discovered vulnerabilities can lead an adversary to compromise the EVCMS and its inter-communications to conduct attacks against the stakeholders namely the EVCS, the users/operators, and the power grid.

EV Ecosystem and Operations. The discovery of vulnerabilities within the context of EVCMS products, which orchestrate the EVCS landscape constitutes a major security problem since this endangers the overall EV charging ecosystem as illustrated in Figure 8. Under normal conditions, the users/operators are able to manage their EVCS by relaying operations through the EVCMS, to control the underlying EVCS by sending commands to it. On a simplified diagram (Figure 9), we present details of the intercommunication protocols that tie the main entities: operator, EVCMS, and EVCS. As shown in Figure 9,

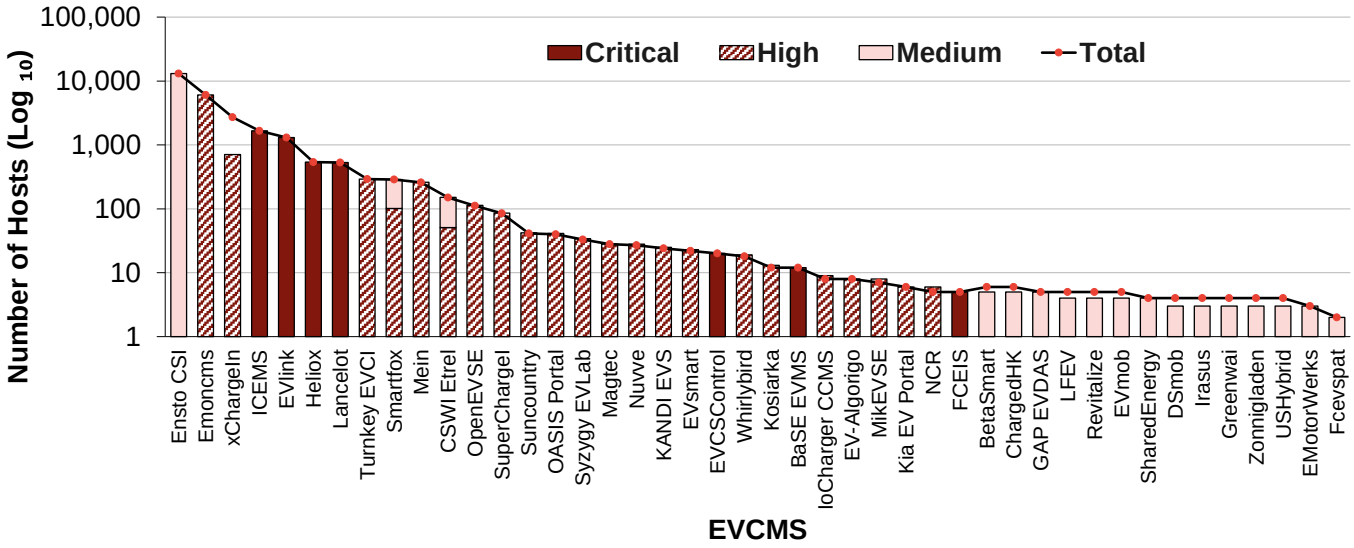


Fig. 7: Distribution of vulnerabilities across EVCMS products/hosts.

the intercommunication protocols can be broken down to 4 phases:

1) In authentication phase, the operator requests access to the EVCMS by submitting their credentials through an HTTP POST request to the corresponding login endpoint, whenif these credentials are valid, the EVCMS will grant the operator access and redirect them past the login wall.
2) In the management phase, the operator will request available functionalities by sending an HTTP GET request to which the EVCMS will provide the appropriate dashboard components based on their privilege level, after which the operator can perform various operations that are demanded through endpoint-specific HTTP GET/POST requests.
3) During the control phase, EVCMS establishes a communication channel, to send and execute commands (e.g., start/stop charging, etc) for the specific operation requested, after which the EVCS will return status to update the respective EVCMS configuration.
4) In monitoring phase, the operator will request EVCS status overview by issuing an HTTP GET request to the EVCMS that provides back the demanded information, and in turn, the EVCS, which is connected to the power grid through power links (Figure 8), can extract power from the grid to feed into plugged-in EVs to charge them, or inject power back into the grid to discharge plugged-in EVs. This ability for EVCS to discharge EVs, relies on the EVCS bidirectional power flow feature enabled by Vehicle-to-Grid (V2G) technology [39].

Attacks Against the EVCS. An attacker who exploits the discussed vulnerabilities can manipulate the EVCS charging operations and schedules (i.e., initiate, delay, or stop charging). Many of the analyzed EVCMS suffered from XSS that could allow an attacker to inject malicious JavaScript code into the context of the EVCMS and hijack the operator's account, allowing the modification of the operator's account and EVCS settings/configurations by manipulating and/or disrupting ongoing charging operations by overriding commands through the controlled EVCS. An attacker can potentially damage the battery of the connected EV by modifying their charging levels and ignoring critical battery conditions through the toleration of high voltages/currents [40].

In addition, by gaining high-privileged access to a compromised EVCMS through SQLi exploitation, an adversary can downgrade the EVCS firmware and potentially upload a maliciously crafted firmware, allowing to maintain covert low-level access on the EVCS that is difficult to detect, thus creating persistence over the EVCS. An adversary can also exploit a group of EVCS via SSRF or XXE and leverage them as network proxies to perform coordinated local and/or Internet scanning activities as a part of a botnet. While these botnets can be used to flood other devices on the Internet through distributed denial of service (DDoS) attacks, they can also be utilized to discover additional vulnerable networked devices, which can be compromised to expand the attacker's foothold and increase the attack surface. Furthermore, an adversary can leverage CSRF to lock the EVCS, disable specific features and deny physical/virtual access to the legitimate users, thus performing a physical DoS, and such attacks could be weaponized with crafted ransomware to score financial gain by abusing the power over these compromised EVCS and locking them until a ransom is fulfilled.

Attacks Against the Users/Operators. The vulnerable EVCMS instances can be exploited to gain control over the EVCS and dictate its relation with the EVs, and such exploits may expose the stored data within the systems, risking the security and privacy of its users/operators. For instance, by hijacking the EVCMS user session through XSS or CORS/FCDP misconfigurations, an adversary can obtain access to the operators/users' personally identifiable information (PII) such as name, address, and telephone number, and this information can be leaked or sold to cyber criminals and then used for blackmailing, harassment, and identity theft. In extension to PII leakage attacks, an adversary can also obtain access to

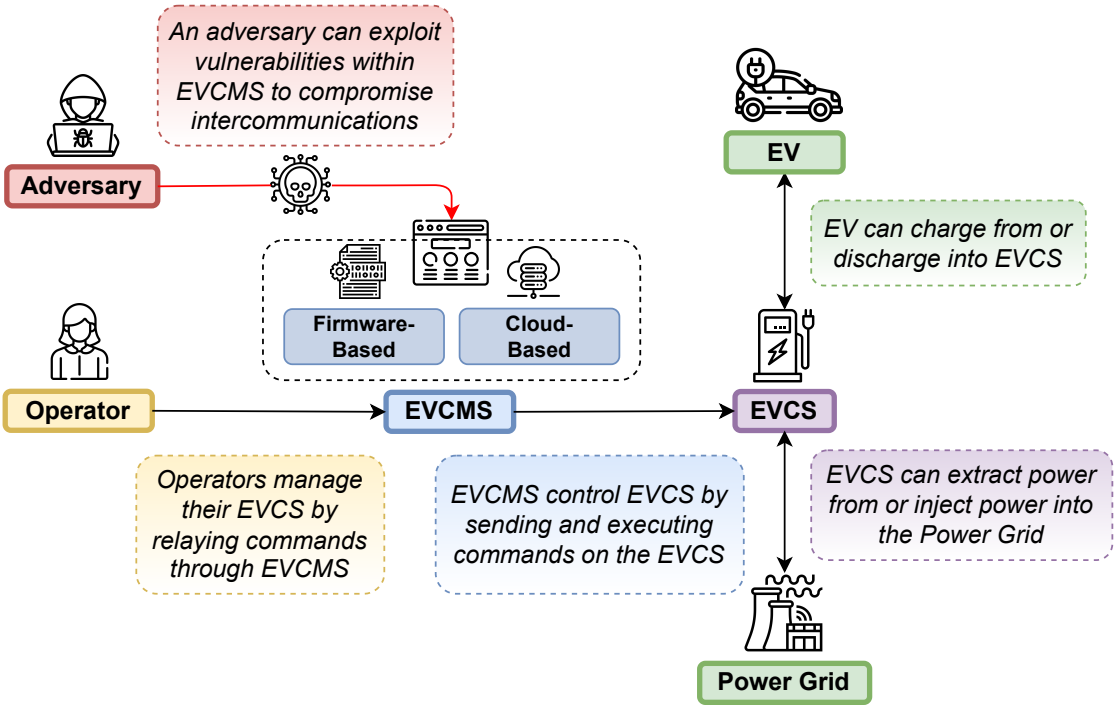


Fig. 8: Interplaying entities surrounding EVCMS, with interfering adversary.

other resources stored on the EVCMS such as charging records and EV-specific log data, from which they can infer charging schedules and use to perform surveillance/espionage on the corresponding user/operator. On another level, many EVCS permit electronic billing and payments through their respective EVCMS, and consequently, this data could also be leaked by an adversary from compromised EVCMS host instances, and abused by cyber criminals to perform payment frauds, through the exploitation of vulnerabilities such as SQLi which are utilized to dump the information stored within the EVCMS.

Attacks Against the Power Grid. The vulnerable EV charging ecosystem introduces a new and rapidly growing attack surface against critical infrastructure, mainly due to the direct connection and integration of EVCS within the power grid, and given the important role it plays in serving electricity to millions of customers, any attacks against such infrastructure would lead to significant implications. In fact, performing large-scale cyber attacks against the power grid attracts various adversaries including large organizations and state-sponsored malicious actors, who may seek economical and/or reputation damage to their opposition [41].

In essence, to conduct these attacks, the adversary requires to control the charging process of a large number of compromised EVCMS host instances to control their underlying EVCS and initiate concurrent EV charging sessions as well as stop/delay ongoing charging operations to conduct several dangerous frequency instability attacks against the power grid [42]–[44]. The literature provides information and simulation results that demonstrate several ways to perform frequency instability attacks against the power grid. For instance, adversaries can create a switching attack by commanding the vulnerable EVCS to charge and discharge the connected EVs within a short time period, causing frequency disturbances and cascading failures in the power grid [45], [46]. Moreover, such attacks can be performed by force-discharging 8,300 EVs, as demonstrated by the simulation analysis results in [47]. Considering the number of vulnerable EVCMS that were discovered using our analysis (i.e., 27,439), and the fact that a single EVCMS is typically managing several EVCS, we conclude that it is very feasible to conduct such frequency instability attacks against the power grid. While in these studies authors assume EVCS compromise and do not discuss the exploitation details, in our work, we present the technicalities of creating a botnet of compromised EVCS.

D. Recommended Countermeasures

While in this work we highlight major security flaws in EVCMS, we also recommend a number of mitigating countermeasures to address the current security issues, which have been overlooked by the respective system developers, and strengthen the deployed EVCMS against future attacks. First, mitigating the discussed attacks requires patching all the identified classes of vulnerabilities within the EVCMS, and for that we refer to the documentation on CWE MITRE [38] and the Open Web Application Security Project [22] for detailed information about known/recommended countermeasures for the CWE-ID of each vulnerability. For instance, to prevent SQLi (CWE-89), developers must not use dynamic queries nor include user-supplied input in query execution, and to prevent XSS (CWE-79), must properly encode and sanitize user supplied data on entry points and parameters before it is returned in the response. Second, to mitigate cyber attacks that target the power grid, the operators can monitor the EVCS charging schedules to detect anomalies in the charging

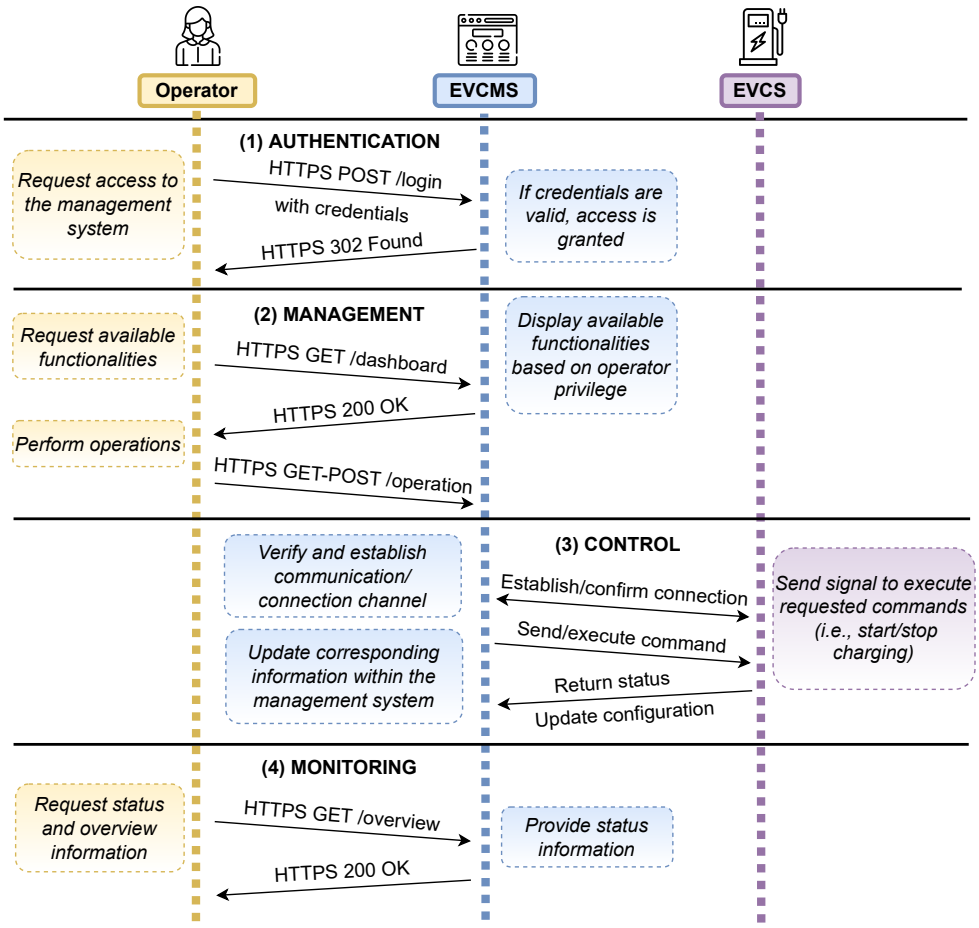


Fig. 9: Diagram for EVCMS intercommunication protocols.

behavior by leveraging machine learning models to learn normal patterns and identify abnormal activities.

In addition, to prevent EVCS configuration and EV charging schedule tampering, the EVCMS and the EVs can implement a mutual consensus to validate system modifications on both ends, and as a result, an adversary who compromises the EVCMS will not be able to enforce custom charging schedule configurations without the approval of the participating entities (e.g., EV operator/user). Third, to diminish the EVCMS attack surface, it is essential for system developers to adopt the secure-by-design process by continuously assessing the security of their products during systems development life cycle (SDLC), and for the operators to properly and securely setup their EVCS in order to prevent certain attacks. For instance, users should always replace the default credentials that are shipped on the EVCS firmware with strong account credentials, as well as set up resilient authentication methods.

Furthermore, private EVCS operators can disable public device discovery on their EVCMS portals to hide them from remote Internet attackers, and configure a firewall that only allows connections from trusted parties. Finally, given the wide-scale impact/implications of the discovered vulnerabilities, there should be strict EVCMS security measures and design standards made and mandated by the law, to motivate system developers/vendors to employ more efforts into making sure that their systems undergo the necessary levels of scrutiny to meet client security expectations and fulfill the requirements imposed by the corresponding laws.

V. Disclosure of Zero-days and Patch Follow-Up

With the established CVD process, we communicated the findings to the EVCMS system developers through the appropriate channels, via encrypted emails sent to the dedicated Product Security Incident Response Team (PSIRT) addresses and via website security incident forms, prior to disclosing results to allow them to take the necessary actions. While for various EVCMS products, there was not enough information about the corresponding vendors/developers in order to communicate the findings to (e.g., absence of dedicated email addresses, inability to find vendor website, lack of response), several manufacturers like Schneider Electric, Cornerstone Technologies, Bluesky Energy, Eaton, and Etrel have received and acknowledged the discovered zero-days, and assigned in coordination with the National Institute of Standards and Technology (NIST) [48], more than 20 CVE-IDs such as: *CVE-2021-22706* (CVSS score: 8.8/high), *CVE-2021-22722* (CVSS score: 8.9/high), *CVE-2021-22729* (CVSS score: 9.4/critical), *CVE-2021-22730* (CVSS score: 9.4/critical). Furthermore, these vendors took steps to address the vulnerabilities, especially the assigned CVEs, by deploying the

TABLE VI: EVCMS software patch version releases.

EVCMS	Manufacturer	Patch Version(s)	Date mm/yy
Ensto CSI	Ensto	5.13.1-11316	01/22
Emonscms	Open Energy Monitor	11.0.4	10/21
xChargeIn	Eaton Corporation	Green Motion	03/22
ICEMS	Bluesky Energy	2.0.0	11/21
EVlink	Schneider Electric	R8 V3.4.0.4	05/22
Lancelot	Unicorn Systems	v5.5	09/21
Smartfox	Smartfox	EM2 00.01.03.19	07/22
CSWI Etrel	Etrel	INCH 5.0	06/22
OpenEVSE	OpenEVSE	8.2.0	04/22
Suncountry	Sun Country Highway	sc2	10/21
OASIS Portal	OASIS	v2022	02/22
Syzygy EVLab	Syzygy	v2.0	11/21
EVsmart	EnelX	Version 1.4.0.28	05/21
EVCSControl	EVCSControl	1.31.1	06/22
BaSE EVMS	Cornerstone Technologies	v22	01/22
IoCharger CCMS	IoCharger	3.2	11/21
FCEIS	Fuzhou Comprehensive Energy	v2022	07/21
Greenwai	Greenwai	v2.0	04/22

corresponding patches in the new software releases.

To assess the status of the planned and deployed software patches, we performed a follow-up on the progress that these manufacturers have made within 3-6-9 months from the initial date of reporting the vulnerabilities to them throughout the year 2021. In Table VI, we present the results of this follow-up, listing the EVCMS with their corresponding software patch versions in which they addressed the vulnerabilities. Out of the 44 analyzed EVCMS, 18 (40%) had software update releases to patch security weaknesses, among which we listed the latest releases along with the corresponding dates. For instances, the vulnerabilities within EVlink version R7 (Table IV) were patched on several phases that were completed with the final release of the new version R8 on May 2022, as shown in Table VI. For several of these EVCMS products, we worked closely with the manufacturers/developers, like Schneider Electric and Cornerstone Technologies, to perform successive tests and validate the deployed fixes. In most cases, the patches accurately resolved the issues, however, there were certain scenarios where new variants of the previous vulnerabilities were uncovered. For instance, when XSS issues on particular parameters got patched, new issues were uncovered on new/other endpoints due to introducing new software features.

All in all, based on the static analysis performed on these EVCMS software updates, better defense mechanisms have been implemented, meeting up-to-date security standards deployed in current IT systems. As for the remaining systems, no updates have been affirmed or observed due to various reasons: (1) lack of sufficient contact information thus the inability to communicate with the corresponding vendors/developers concerning the discovered vulnerabilities, (2) the patches not being released yet, (3) the absence of software update for logistical reasons such as some products being discontinued, or (4) the manufacturer shifting efforts from patching current system vulnerabilities to investing in new product lines.

Furthermore, to survey the state of updates for the original set of EVCMS host instances, we performed a global scan by probing the systems through version string extraction and comparing those to the current list of patched EVCMS versions. We determined that out of 27,439 hosts, 6,980 (25.5%) still represent the EVCMS instances that were originally detected, while the remaining 20,459 (74.5%) are either dynamic IP addresses that are no longer associated with the EVCMS instances or failed to resolve successfully. We note that out of those 6,980 EVCMS instances, only 1,112 (15.9%) have been updated accordingly to a new software release version that patches the discovered vulnerabilities. In addition, we perform a new round of scans on a fresh set of 14,900 EVCMS host instances, out of which the version strings indicate that only 1,760 (11.8%) instances are updated, thus properly secured.

In both these scans, we observe that the updated EVCMS instances deploy Ensto CSI and EVlink, and we highlight that the overall low number of updates can be attributed to the general EVCMS design architecture which has low update frequency and lacks auto-update features hence needing manual/technical patching, and enforces requirements to restart the EVCS when applying updates, which are operational burdens that most users avoid. Nevertheless, it is best for these EVCMS instances not to be exposed to the public Internet and rather only be connected to isolated or virtual internal networks to reduce attack surface, and still maintain the expected remote functionality.

VI. Lessons Learned and Future Work

While we identified and analyzed 44 EVCMS, we note that obtaining information about all available EVCMS in the wild is a challenging task due to the proprietary nature of some EVCMS products which are only provided to Charging Point Operators (CPO) and enterprise-level customers with a prepaid subscription. Additionally, the security mechanisms of some EVCMS made it unfeasible to inspect content located behind authentication portals. In those cases, partial EVCMS assets were examined, thus the remaining assets could still potentially conceal vulnerabilities. In terms of future work, we note that we leveraged classification methods along with extracted features, which can be updated or modified to enhance the overall fingerprinting outcome, by implementing and evaluating further classification models to find the best methods/parameters for each stage. Finally, we could also leverage ChargePrint and the knowledge of this study to build and deploy an online platform for conducting real-time discovery and vulnerability analysis of EVCMS products that are submitted for vetting by the respective system developers.

VII. Related Work

In this study, we examine both discovery and security aspects of a niche attack surface that has never been tackled before in the literature, by designing a novel framework that utilizes custom techniques to circumvent challenges that are unique to EVCMS analysis. As summarized in Table VII, we systematize previous research work by categorizing the literature into three classes: device fingerprinting, EVCS security, and firmware analysis, while comparing them in terms of fingerprinting and security analysis.

Device Fingerprinting. Several studies have proposed approaches to discover and fingerprint IoT and ICS devices. Feng et al. [8] proposed an engine that generates association rules for discovering and annotating IoT devices by extracting relevant terms in their application-layer response data then using them

TABLE VII: Literature systematization. The ✓, ◐ , and empty cell imply complete, partial and no fulfillment, respectively.

		Fingerprinting					Security Analysis										
Category	**Reference**	Firmware Available	Hardware Available	Banner-Based	Rule-Based	Feature-Based	Knowledge Rehashing	Black-Box	White-Box	Manual	Automated	Custom	Third-Party	Static	Dynamic	Known Vuln. (N-days)	Unknown Vuln. (0-days)
Our Work	ChargePrint	◐		✓		✓		◐	◐	◐	◐	◐	◐	✓			✓
Device Fingerprinting	Feng et al. [8]			✓	✓												
	Costin et al. [49]	✓		✓		✓											
	Wang et al. [9]			✓		✓											
	Yu et al. [50]			✓		✓											
EVCS Security	Alcaraz et al. [6]								✓	✓				✓			✓
	Alcaraz et al. [51]								✓	✓				✓			✓
	Boe et al. [52]															✓	
	Baker et al. [7]							✓		✓		✓					✓
	Pratt et al. [53]						✓									✓	
	Antoun et al. [54]						✓									✓	
	Gottumukkala et al. [55]						✓									✓	
	Fraiji et al. [56]						✓									✓	
	Mousavian et al. [57]						✓					✓					
System Analysis	Costin et al. [20]	✓						✓			✓		✓	✓		✓	
	Zheng et al. [21]	✓						✓			✓	✓			✓	✓	
	Srivastava et al. [58]	✓						✓			✓	✓		✓	✓	✓	
	Wright et al. [59]						✓										
	Chen et al. [60]	✓						✓			✓	◐	◐		✓	◐	◐
	Fasano et al. [61]	✓					✓	✓			✓	✓			✓		
	Sasaki et al. [14]			✓		✓		✓		◐	◐		✓	✓		◐	◐

as web search engine queries to find product descriptions. Costin et al. [49] utilized supervised machine learning to classify a database of embedded device firmware then fingerprint online web interfaces to link them to the corresponding images from the database. Sasaki et al. [14] presented a fingerprinting methodology tailored for discovering ICS remote management devices, by selecting networks with probable presence of ICS and collecting WebUIs from them, then identifying and creating signatures from remote management systems detected based on WebUIs with customized fields containing name and location of monitored facilities. The authors compare and note that their results surpass those of industry source that identify ICS such as Shodan, nevertheless, a large percentage of these devices were already tagged as ICS by the search engine. Despite that, we note that unlike ICS, which already have well-documented and built-in tags provided by device search engines such as Shodan and Censys, there is a lack of knowledge about EVCMS and an absence of built-in tags for identifying them. This indeed hampers the results of previous work when it comes to fingerprinting EVCMS in the wild.

From a different perspective, Wang et al. [9] proposed an engine for identifying IoT devices by leveraging the highest similarity of response data between IoT devices of the same vendor or product by extracting structure and style features from response data. Yu et al. [50] proposed a firmware identification method by analyzing web pages content to extract information while using classification and page segmentation to identify device model and firmware version. In contrast to other device types, EVCS have limited and non-trivial banners due to the difficulty of locating information and specifications related to them especially since most EVCMS products are cloud-based and closed-sourced, in addition to the absence of banner rules for identifying them. Furthermore, EVCMS's diversity and lack of standardization among developers results in a wide range of banner representations that are harder to analyze and keep track of, making it unfeasible to use existing approaches to fingerprint EVCS, therefore, we design our approach which we bootstrap with EVCMS seeds from preliminary search.

EVCS Security. Previous studies have looked at different aspects of EVCS security, however, EVCS firmware and EVCMS security received little academic attention. Alcaraz et al. [6] conducted a security analysis of open charge point protocol (OCPP) and presented weaknesses that allow man-in-the-middle attacks to interfere with EV resource reservation, and presented countermeasures to protect against their proposed attacks [51]. Boe et al. [52] performed a security analysis of vehicle-to-grid protocol, presenting attacks to target the charging process. Baker et al. [7] implemented a wireless eavesdropping tool to conduct electromagnetic side-channel attacks for recovering messages from the EVCS power-line communication networks. Pratt et al. [53] devised security principles to prevent cyber attacks against the EVCS and the power grid. Antoun et al. [54] and Gottumukkala et al. [55] presented a theoretical overview of cyber threats associated with the EV charging ecosystem components. Fraiji et al. [56] surveyed the security of the Internet-of-Electric-Vehicles (IoEV) pointing out cyber attacks that can be used to disrupt its operations. Mousavian et al. [57] proposed a model that optimizes security risk within the EV infrastructure to handle a malware propagation attack through the EVCS communication networks.

We note that these works rely on theoretical attack scenarios where the EVCS are assumed to be infected by malware without discussing the exploitation process. To the best of our

knowledge, our work is the first to conduct an in-depth security analysis on a body of EVCMS products while presenting a range of vulnerabilities that would allow practical remote exploitation and manipulation of EVCS, shedding the light on the large-scale insecurity of the EV ecosystem, while alarming vendors to implement immediate mitigations.

System Analysis. A number of prior research examined the security of IoT and ICS device firmware. Costin et al. [20] performed a large-scale IoT firmware analysis by leveraging security tools and static analysis techniques. Sasaki et al. [14] performed a security analysis of ICS remote management systems by examining insecure configurations, surveying unpatched known vulnerabilities, and performing penetration tests to find zero-days by leveraging existing scanners (e.g., Nmap, OpenVAS). While the topic of ICS has been previously examined and has more extensive literature that discussed its security, it is difficult to leverage off-the-self tools to perform security analysis on products with deep code paths like EVCMS. Thus, we rely on tailored and specific security analysis procedures that leverage static analysis/dynamic indicators to detect vulnerabilities within EVCMS. Zheng et al. [21] proposed a fuzzer for IoT firmware to uncover 1-day vulnerabilities through user and system mode emulation. Srivastava et al. [58] built an emulation and dynamic analysis framework for Linux-based firmware that employs fuzzing and static analysis techniques to uncover software bugs. Wright et al. [59] categorized popular works in the field of firmware re-hosting, and presented common challenges faced during system emulation and analysis. Chen et al. [60] presented an automated system (Firmadyne) that performs dynamic analysis of embedded device firmware through full system emulation.

We tested Firmadyne on the collected EVCMS firmware images in an attempt to observe analysis results, however, it failed to run many of the supplied EVCMS firmware, due to its inability to analyze proprietary file compressions like EPK. Nevertheless, we incorporated into ChargePrint's design some of the analysis techniques introduced by Firmadyne, such as those related to filesystem extraction methodologies. Fasano et al. [61] presented rehosting as an alternative to firmware emulation for dynamic analysis of hardware systems. While these studies are effective at analyzing IoT firmware, most of these approaches rely on known CVE vulnerabilities and security scanners, thus do not discover new vulnerabilities, in addition to suffering from high false positives. Most these studies also solely perform analysis on firmware packages, while we perform analysis on both available firmware and online endpoints without having access to the underlying firmware. Furthermore, most these studies rely on dynamic analysis techniques to examine the embedded device images, which is practically unfeasible when analyzing EVCMS images due to hardware restrictions and proprietary firmware which are incompatible with emulation software like QEMU [62].

VIII. Conclusion

We provide the first attempt to explore threats associated with EVCS by evaluating the security posture of their EVCMS as a new attack surface. We present a novel discovery and security analysis framework (ChargePrint) that fingerprints EVCMS instances in the wild while analyzing their vulnerabilities. Moreover, we leveraged ChargePrint to extend the device discovery/fingerprinting capabilities of existing search engines by identifying 27,439 hosts that implement 44 different EVCMS. Furthermore, our in-depth analysis raises serious concerns regarding the insecurity of the implemented EVCMS at scale by uncovering 120 0-day vulnerabilities that can lead to remote exploitation across the majority (92%) of the EVCMS hosts. Finally, while we note attack implications against various stakeholders in the EV charging ecosystem, we communicate our findings to the respective system developers to motivate them towards improving the security of EVCMS and the overall EV charging ecosystem.

Acknowledgments

We thank the reviewers for their time and valuable feedback. This work has been supported by Natural Sciences and Engineering Research Council of Canada (NSERC) and Concordia University. This work was also partially supported by a grant from the U.S. National Science Foundation (NSF), Office of Advanced Cyberinfrastructure (OAC), #1907821 and #2104273.

References

[1] Hydro-Quebec, "Electric Vehicle Charging Stations: Technical Installation Guide," https://www.hydroquebec.com/data/electrification-transport/pdf/technical-guide.pdf, 2015.

[2] Gouvernement du Québec, "New Electric Vehicle Rebate," https://vehiculeselectriques.gouv.qc.ca/english/rabais/ve-neuf/programme-rabais-vehicule-neuf.asp, 2020.

[3] International Energy Agency, "Global EV Outlook 2021," https://www.iea.org/reports/global-ev-outlook-2021, Jun. 2020.

[4] F. Lambert, "Electric car charge points soar to 7.3 million chargers, 60% growth in public chargers," https://electrek.co/2020/06/15/electric-car-charge-points-data, Electrek, Jun 2020.

[5] D. Shelar *et al.*, "Compromising Security of Economic Dispatch in Power System Operations," in *Proc. of the 47th Annual IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN)*, 2017, pp. 531–542.

[6] C. Alcaraz, J. Lopez, and S. Wolthusen, "OCPP Protocol: Security Threats and Challenges," *IEEE Transactions on Smart Grid*, vol. 8, no. 5, pp. 2452–2459, 2017.

[7] R. Baker and I. Martinovic, "Losing the car keys: Wireless PHY-Layer insecurity in EV charging," in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 407–424.

[8] X. Feng *et al.*, "Acquisitional rule-based engine for discovering Internet-of-Things devices," in *27th USENIX Security Symposium (USENIX Security 18)*. Baltimore, MD: USENIX Association, Aug. 2018, pp. 327–341.

[9] X. Wang *et al.*, "Iottracker: An enhanced engine for discovering internet-of-thing devices," in *2019 IEEE 20th International Symposium on" A World of Wireless, Mobile and Multimedia Networks"(WoWMoM)*. IEEE, 2019, pp. 1–9.

[10] National Vulnerability Database, "Common Vulnerabilities and Exposures (CVE)," https://cve.mitre.org, 2021.

[11] Office of Energy Efficiency & Renewable Energy, "Vehicle Charging," https://www.energy.gov/eere/electricvehicles/vehicle-charging, 2020.

[12] J. Tournier *et al.*, "A survey of iot protocols and their security issues through the lens of a generic iot stack," *Internet of Things*, vol. 16, p. 100264, 2021.

[13] OCPP, "OCPP 2.0 Part 2: Specification. Technical Report," 2018.

[14] T. Sasaki *et al.*, "Exposed infrastructures: Discovery, attacks and remediation of insecure ics remote management devices," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 2379–2396.

[15] B. Zhao *et al.*, "A large-scale empirical study on thevulnerability of deployed iot devices," *IEEE Transactions on Dependable and Secure Computing*, 2020.

[16] Shodan, "The search engine for Internet-connected devices." https://www.shodan.io, Shodan, 2021.

[17] Censys, "A search engine based on Internet-wide scanning for the devices and networks." https://censys.io, Censys, 2021.

[18] Zoomeye, "ZoomEye - Cyberspace Search Engine," https://www.zoomeye.org, Zoomeye, 2021.

[19] Fofa, "Fofa," https://fofa.so, Fofa, 2021.

[20] A. Costin *et al.*, "A Large-Scale analysis of the security of embedded firmwares," in *23rd USENIX Security Symposium (USENIX Security 14)*. San Diego, CA: USENIX Association, Aug. 2014, pp. 95–110.

[21] Y. Zheng *et al.*, "FIRM-AFL: High-Throughput greybox fuzzing of IoT firmware via augmented process emulation," in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 1099–1114.

[22] OWASP, "OWASP Foundation — Open Source Foundation for Application Security," https://owasp.org, 2021.

[23] MITRE, "2021 CWE Top 25 Most Dangerous Software Weaknesses," https://cwe.mitre.org/top25/archive/2021/2021_cwe_top25.html, 2021.

[24] Zyte, "Scrapy — A Fast and Powerful Scraping and Web Crawling Framework," https://scrapy.org, Scrapy, 2021.

[25] R. Labs, "Binwalk: Nb 1 Firmware extraction tool in the world." https://www.refirmlabs.com/binwalk, Refirm Labs, 2021.

[26] J. W. Ratcliff and D. E. Metzener, "Pattern-matching-the gestalt approach," *Dr Dobbs Journal*, vol. 13, no. 7, p. 46, 1988.

[27] rizin.re, "Cutter," https://cutter.re, rizin.re, 2021.

[28] PortSwigger, "Burp Suite - Application Security Testing Software - PortSwigger)," https://portswigger.net/burp, 2021.

[29] B. Damele and M. Stampar, "sqlmap: automatic SQL injection and database takeover tool," http://sqlmap.org, 2021.

[30] O. Alrawi *et al.*, "The betrayal at cloud city: An empirical analysis of Cloud-Based mobile backends," in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 551–566.

[31] S. Schrittwieser, M. Mulazzani, and E. Weippl, "Ethics in security research which lines should not be crossed?" in *2013 IEEE Security and Privacy Workshops*. IEEE, 2013, pp. 1–4.

[32] J. Dittmer, "Minimizing Legal Risk When Using Cybersecurity Scanning Tools." https://www.sans.org/white-papers/37522/, SANS, Jan. 2017.

[33] Z. Durumeric, E. Wustrow, and J. A. Halderman, "ZMap: Fast internet-wide scanning and its security applications," in *22nd USENIX Security Symposium (USENIX Security 13)*. Washington, D.C.: USENIX Association, Aug. 2013, pp. 605–620.

[34] F. Li *et al.*, "You've got vulnerability: Exploring effective vulnerability notifications," in *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, Aug. 2016, pp. 1033–1050.

[35] T. Ristenpart *et al.*, "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds," in *Proceedings of the 16th ACM conference on Computer and communications security*, 2009, pp. 199–212.

[36] Organisation for Economic Co-operation and Development (OECD), "ENCOURAGING VULNERABILITY TREATMENT," Online at https://one.oecd.org/document/DSTI/CDEP/SDE(2020)3/FINAL/en/pdf, 2021.

[37] The ZMap Project, "The ZMap Project," https://zmap.io, The ZMap Project, 2021.

[38] MITRE, "Common Weakness Enumeration (CWE)," https://cwe.mitre.org, 2021.

[39] Z. Zhou *et al.*, "Secure and efficient vehicle-to-grid energy trading in cyber physical systems: Integration of blockchain and edge computing," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 50, no. 1, pp. 43–57, 2020.

[40] F. Sagstetter *et al.*, "Security challenges in automotive hardware/software architecture design," in *2013 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2013, pp. 458–463.

[41] G. Liang *et al.*, "The 2015 ukraine blackout: Implications for false data injection attacks," *IEEE Transactions on Power Systems*, vol. 32, no. 4, pp. 3317–3318, 2016.

[42] S. Soltan, P. Mittal, and H. V. Poor, "BlackIoT: IoT botnet of high wattage devices can disrupt the power grid," in *27th USENIX Security Symposium (USENIX Security 18)*. Baltimore, MD: USENIX Association, Aug. 2018, pp. 15–32.

[43] S. Acharya, Y. Dvorkin, and R. Karri, "Public Plug-in Electric Vehicles + Grid Data: Is a New Cyberattack Vector Viable?" *IEEE Transactions on Smart Grid*, pp. 1–1, 2020.

[44] O. Erdinc *et al.*, "Smart household operation considering bi-directional ev and ess utilization by real-time pricing-based dr," *IEEE Transactions on Smart Grid*, vol. 6, no. 3, pp. 1281–1291, 2014.

[45] A. K. Farraj and D. Kundur, "On Using Energy Storage Systems in Switching Attacks that Destabilize Smart Grid Systems," in *Proc. of the IEEE Power & Energy Society Innovative Smart Grid Technologies Conference (ISGT)*, 2015, pp. 1–5.

[46] L. Shan *et al.*, "A Coordinated Multi-Switch Attack for Cascading Failures in Smart Grid," *IEEE Transactions on Smart Grid*, vol. 5, no. 3, pp. 1183–1195, 2014.

[47] M. A. Sayed *et al.*, "Electric vehicle attack impact on power grid operation," *International Journal of Electrical Power & Energy Systems*, p. 107784, 2021.

[48] National Institute of Standards and Technology (NIST), "National Institute of Standards and Technology — NIST," https://www.nist.gov, 2021.

[49] A. Costin, A. Zarras, and A. Francillon, "Towards automated classification of firmware images and identification of embedded devices," in *IFIP International Conference on ICT Systems Security and Privacy Protection*. Springer, 2017, pp. 233–247.

[50] D. Yu *et al.*, "Large-scale iot devices firmware identification based on weak password," *IEEE Access*, vol. 8, pp. 7981–7992, 2020.

[51] J. E. Rubio, C. Alcaraz, and J. Lopez, "Addressing Security in OCPP: Protection Against Man-in-the-Middle Attacks," in *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, Paris, France, 2018, pp. 1–5.

[52] K. Bao *et al.*, "A Threat Analysis of the Vehicle-to-Grid Charging Protocol ISO 15118," *Computer Science-Research and Development*, vol. 33, no. 1-2, pp. 3–12, 2018.

[53] R. M. Pratt and T. E. Carroll, "Vehicle Charging Infrastructure Security," in *2019 IEEE International Conference on Consumer Electronics (ICCE)*, Las Vegas, NV, USA, 2019, pp. 1–5.

[54] J. Antoun *et al.*, "A Detailed Security Assessment of the EV Charging Ecosystem," *IEEE Network*, vol. 34, no. 3, pp. 200–207, 2020.

[55] R. Gottumukkala *et al.*, "Cyber-physical System Security of Vehicle Charging Stations," in *2019 IEEE Green Technologies Conference (GreenTech)*, Lafayette, LA, USA, 2019, pp. 1–5.

[56] Y. Fraiji *et al.*, "Cyber Security Issues of Internet of Electric Vehicles," in *2018 IEEE Wireless Communications and Networking Conference (WCNC)*, Barcelona, Spain, 2018, pp. 1–6.

[57] S. Mousavian *et al.*, "A risk-based optimization model for electric vehicle infrastructure response to cyber attacks," *IEEE Transactions on Smart Grid*, vol. 9, no. 6, pp. 6160–6169, 2018.

[58] P. Srivastava *et al.*, "Firmfuzz: Automated iot firmware introspection and analysis," in *Proceedings of the 2nd International ACM Workshop on Security and Privacy for the Internet-of-Things*, 2019, pp. 15–21.

[59] C. Wright *et al.*, "Challenges in firmware re-hosting, emulation, and analysis," *ACM Computing Surveys (CSUR)*, vol. 54, no. 1, pp. 1–36, 2021.

[60] D. D. Chen *et al.*, "Towards automated dynamic analysis for linux-based embedded firmware." in *NDSS*, vol. 1, 2016, pp. 1–1.

[61] A. Fasano *et al.*, "Sok: Enabling security analyses of embedded systems via rehosting," in *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, 2021, pp. 687–701.

[62] Software Freedom Conservancy, "QEMU: A generic and open source machine emulator and virtualizer." https://www.qemu.org/, QEMU, 2021.