HoneyEVSE, a low-interaction honeypot for EVSE emulation

Alessio Trevisan
Department of Mathematics
University of Padova
Padova, Italy
alessio.trevisan.2@studenti.unipd.it

Massimiliano Baldo
Department of Mathematics
University of Padova
Padova, Italy
massimiliano.baldo@studenti.unipd.it

Federico Turrin
Department of Mathematics
University of Padova
Padova, Italy
turrin@math.unipd.it

Tommaso Bianchi
Department of Mathematics
University of Padova
Padova, Italy
tommaso.bianchi@phd.unipd.it

***Abstract*—Honeypots [1] are becoming one of the most widespread tools that might help to avert attacks to Industrial Control Systems (ICS) and analyze the possible strategies, means, techniques and targets used by the attackers.**
Although these emulators might be really effective for their purposes, they might be exposed whether they fail simulate a proper ICS in a trustworthy manner. In fact, for example, an honeypot that does not provide a reliable dataset or does not expose an adequate number of services might be easily discovered by an attacker.
However, achieving the emulation of a real ICS is a tough job, due to many factors carried by the requirements related to this objective. One of the most important and complicated demands that is usually associated to honeypots is the simulation of real physical processes that take place in an interactive ICS.
In this paper we present HoneyEVSE, a low-interaction honeypot that emulates the behaviour of an electric vehicle supply equipment (EVSE) by implementing a physical process simulation.
We developed this tool by using a set of pre-existing applications which allowed us to achieve many features such as a real recharging simulation and the exposure of a set of services which are expected by the users of a public EVSE.

***Index Terms*—HoneyEVSE, honeypot, EVSE, charging station**

I. Introduction

ICS [1] is a collective term that identifies a used to describe different types of control systems and associated instrumentation, which include the devices, systems, networks and controls used to operate and/or automate industrial processes. In order to properly provide their services, ICS need a special kind of programmable controllers (called PLC [2]) to monitor delicate and risky operations with high reliability, ease of programming and process fault diagnosis.
Nowadays, these structures find application in a very wide variety of environments, such as power plants, factories and all the kind of infrastructure devote to various service provisioning (e.g. gas, water and electricity supply).

In particular, thanks to the crescent development of the market of electric vehicles, a sector related to ICS that has grown rapidly in recent times is represented by EVSE[3]. This acronym identifies a set of tools and infrastructures (such as groups of charging stations) through which electric vehicle owners can recharge the batteries of their cars.
Indeed, as required by the increasing number of electric vehicles in circulation, service providers usually install their charging stations in spaces where users can take a break or park their cars (e.g. parking spots or petrol stations).
These systems are usually accessible through the Internet, as they might provide a set of functionalities that enhance the user experience of a user, such as the monitoring of the status of the recharge (using a web application), payment gateways or management functionalities for system administrators. This inevitably exposes the system to a series of cybersecurity threats which might be explored by some attackers, depending on the level of interaction and protection provided. In fact, even though these systems are usually protected by a firewall, many of them are pinpointed by Shodan [2][4] and their connections might represent an entry vector into the networks of EVSE operator systems. This may let an attacker to potentially control large fleets of EVSE devices which could impact power grid operations, transportation systems or other critical infrastructure.
Among the various solutions proposed in order to fix this kind of issues (which are generally shared among the various kinds of ICS), honeypots constitute one of the most widespread defensive countermeasures. These tools can be described as emulators of a proper industrial control system, whose aim is to deceive possible attacks by making the attacker believe to be targeting a real ICS. Moreover they might provide additional information to developers about the vulnerabilities that the

[1]Industrial Control System
[2]Programmable Logic Controller
[3]Electric Vehicles Supply Equipment
[4]https://www.shodan.io/ is a search engine for hardware devices connected to the Internet. It helps to retrieve the metadata that the server sends back to the client, such as the server software and what options the service supports

attacker tried to exploit, the data or actions that he/she tried to acquire/perform and the means and techniques used.
These systems are divided in two main classes: low-interaction honeypots and high-interaction ones. Low-interaction honeypots are easily identifiable by the attackers as they emulate only a small subset of functionalities that are usually offered by the platform itself and they do not provide reliable information to the attacker. On the other hand, high-interaction honeypots provide a wider set if functionalities and interfaces (camouflaging the ones of a real ICS). Furthermore, they usually emulate physical processes with high reliability. Achieving the implementation of a high-interaction honeypot is far more complex than the development of a low-interaction one, as it adds a consistent amount of work and design overheads.
In this paper we present HoneyEVSE [3], a low-interaction honeypot that simulates the behaviour of a charging station for electric vehicles by leveraging ACNPortal[5](more specifically ACNSim [4]), a suite of tools that collects data and simulates the recharge of a set of electric vehicles. This honeypot is also designed to be extended and turned into a high-interaction type, as it natively embeds a set of tools and services that have been previously used to develop more complex and complete emulators. HoneyEVSE poses itself as a pioneer honeypot in the EVSE sector, as the current literature is not yet presenting an honeypot that is specifically designed to emulate the behaviour of a charging station.

II. EVSE and related usecases

EVSE may vary a lot from each other depending on the manufacturers' implementation choices and service provisioning.
Nevertheless, their users will still expect a reasonable level of interaction with the system, so we proceeded to interview some electric vehicle owners in order to find out which are the most attended and common functionalities presented by the various EVSE that they came in touch in the past.
Furthermore, we analyzed some of the actual charging stations deployed in public places.
By combining the two sets of collected information we reconstructed a possible common list of user stories that are usually implemented by this kind of ICS, both from a service user and an administrator standpoint.

A. User-side requirements analysis

Electric vehicles owners represent the main category of stakeholders (in terms of number of users) that needs to interact with the platform. In a certain sense, this may represent a peculiar scenario in the ICS environment, as usually common users not working for the owner of the ICS do not interact with it or have a really limited access to it.
In particular, they expect to be able to:

- authenticate themselves and start/stop the charging session
- monitor the charge progress in an almost real-time way via some graphs or counters
- monitor the current intensity delivered by the charging station
- monitor the total amount of current delivered by the charging station
- monitor the duration of the charging session
- visualize an estimate that indicates the end of the charging time
- monitor the costs of the charging session in an almost real-time way
- pay for the service usage
- view and manage their own profile
- consult the history of charging sessions related to that specific brand of EVSE
- receive emails or push notifications sent by the administrator of the platform

B. Administrative-side requirements analysis

On the other hand, the other main class that needs to interface with the ICS is represented by the system administrators (or maintenance staff). Even though we did not have direct access to a real admin interface for an EVSE platform, we tried to conceive a plausible set of functionalities that a web application should provide to its administrators. In particular we expect that this HMI[6] should allow administrators to:

- see in real-time the number of cars that are currently connected to the charging stations
- see in real-time the quantity of current delivered by every charging station
- see in real-time the current intensity delivered by every charging station
- see in real-time the usage time of every charging station
- see in real-time the status of every charging station (e.g active/inactive/malfunctioning)
- visualize a log of actions and errors performed on every charging station
- consult the history of charging sessions related to a specific charging station and the corresponding vehicle
- perform research on the history of the usage of the charging stations
- visualize and manage personal information related to the users of the service
- manage different kinds of accounts with different permissions that can interact with the HMI (e.g. maintenance personnel accounts)
- consult the status of the payment related to a charging session
- update the firmware of the EVSE when a new release is available
- remotely access to the PLCs and servers
- download reports related to the use of the service
- create and send messages to the service users through emails or push notifications

[5]https://github.com/zach401/acnportal

[6]Human Machine Interface

III. Taxonomy and analysis of the state of the art for EVSE

The variety and modality of the offered services may vary a lot throughout the various implementations, depending also on whether the charging station is designed for a specific vehicle or for general usage. Considering such factors, we can classify EVSE in three macro groups, based on the ways and kinds of authentication and the modalities and timing of payment offered to their users.

A. Direct authentication during the recharge

The first kind of EVSE minimizes the overhead for the user by directly receiving the ID of the car under recharge, which is associated to the account of the user.
This is the common case for Tesla EVSE, with which only Tesla vehicles can be recharged. The power suppliers usually have no displays, as the status of the charging session is consultable in real-time from the display of the car or from the web and mobile applications released by the producer.
In order to start a charging session the user just needs to plug-in the socket. Once the recharge is completed or the user terminates it (by removing the plug or stopping it from the app), the final costs are directly charged on his/her account, which is associated to a payment method (for example bound to a credit card or bank account).
These charging stations can only operate with Tesla vehicles but, at the time being, the company is starting to integrate solutions that will allow other electric vehicles to use this service.

Fig. 1. An example of a common Tesla charging station

B. Standard authentication through credentials insertion

The second possible scenario involves the creation of a user account that allows the car owner to authenticate himself/herself by inserting his/her credentials on a web or mobile application. In alternative, an user might be in possession of a badge, wield with the same purpose.
This is the common case for Enel X EVSE, in which the user possesses his/her own account and manually needs to authenticate itself to access to the various functionalities offered by the platform.
This kind of scenario is open to every kind of electric vehicle as it does not require any specific requirement related to the car and it is still the most common and familiar way available to the users to authenticate themselves.
This kind of EVSE may also integrate a display to allow the user to interact directly with the system.
The recharge can be automatically started by plugging the socket once the user authenticates himself/herself. The user can also interrupt the recharge at any time from the application. The costs associated to the service usage are directly charged to the payment method inserted by the user.

Fig. 2. An example of a common Enel X charging station

C. Temporary authentication with QR-Code scanning

The last scenario involves the scanning of a QR-Code for temporary authentication. The QR-Code can be static (just a link to the payment service page of the provider) or dynamically refreshed on a display mounted on the charging station (in this way it can be effectively associated to a charging session).
In order to start charging the battery, an user needs to scan the QR-Code, pre-authorize the payment (which is carried out at

the end of the charging session) through the available payment systems offered by the platform and wait for the charging station to be activated (once the payment intent has been received, the charging station will allow power to be supplied to the vehicle).
In this way, the user does not need to own an account, but yet this methodology carries a lot of overheads about the payment management and the eventual cashback. Also this methodology is open to every kind of electric vehicle as it does not need any specific feature related to the car.

Fig. 3. An example of a common QR-Code type of charging station

IV. Cyberthreats related to the EVSE

ICS might be exposed to a considerable set of threats whether the adopted security measures are not adequate to protect it.
The objectives of an attacker can be multiple, ranging from the desire to cause temporary or permanent service interruptions, misappropriation of service usage, physical damage to the structures or personnel or the theft of personal data. The set of attacks that can be performed in a EVSE can be divided into three main classes: side-channel attacks, software injection (like SQL injection and XSS[7])/malwares and DoS [8] attacks.

A. Side-channel attacks

Side channel attacks include all the threats carried out by observing all extra information that can be gathered from outside the system. For example, two possible side-channel attacks led on EVSE are EVExchange [5] and EVScout2.0 [6]. The former targets the charging stations in order to make the user pay for misappropriation of service usage carried by the attacker, while the latter analyzes the current and pilot signal exchanges between the electric vehicle and the charging station during the charging session in order to profile a car.

B. Software injection and malware attacks

The second set includes all the possible attacks that can be addressed to web and mobile applications that contribute to the set of features made available by the system itself. There are plenty of attacks that can be commonly associated to this kind of entrypoints.
First of all, SQL injection attacks (that is to say all the possible attacks that involve the insertion of SQL queries in forms and other entrypoints available in an application to the extent of causing harm to the system, such as database deletion or personal information theft) can be a relevant threat whether not properly addressed.
The second major threat for web applications is represented by XSS attacks. This class includes all the possible attacks that try to modify the behaviour of a web page by injecting malicious client-side programming language code (such as Javascript) to the extent of stealing information or deploying malwares into the user browser.
The last subclass includes the deployment of malwares into the user vehicle, PLCs available in the ICS or device used to interact with the online services offered by the platform (targeting both users and providers of the service). This kind of attacks can include ransomwares, trojans, spywares, keyloggers and rootkits, which can be used to cause disservices, personal and financial information theft, physical harm to people and other unwanted side effects on the service operations.

C. Dos attacks

At last, DoS class includes all the attacks that are willing to temporarily or permanently dismiss a service by flooding its servers with an excessive number of requests.
In particular, plausible DoS attacks may target the servers that host the web applications associated to the ICS. In these cases, the service is likely to be shut down whether the system is not designed to be highly scalable and attacker disposes of a large number of nodes or computational power and bandwidth.

D. Honeypot and cyberthreats

High-interaction reliable honeypots can be used to deceive all the attacks belonging to the second and third main classes. In industrial configurations, these emulators can be deployed in a dedicated segment of the network by exposing it directly on the Internet (to divert attention from the real target of interest) or camouflaging it in between the existent devices to act like a sentinel.
Furthermore, they can be designed in order to collect all the data inserted by the attackers, which can be used for further analysis to understand which vulnerabilities were exploited and what were the intentions of the attacker.

[7]Cross Scripting Attacks
[8]Denial of Services

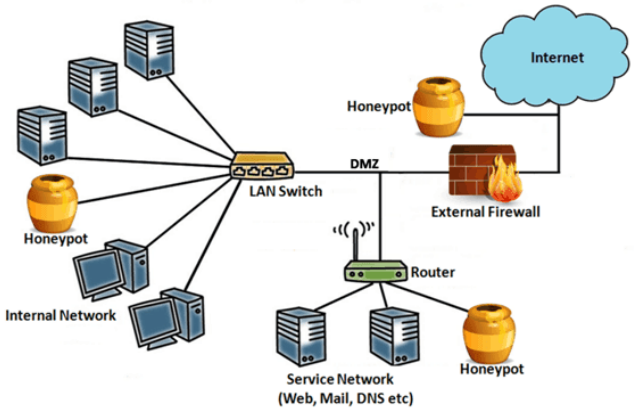

Fig. 4. Possible honeypot routing schema

V. HoneyEVSE

HoneyEVSE is designed to deceive possible attacks addressed to actual EVSE and to retrieve information about the attempts carried out by the attacker to damage the system. The honeypot itself is still in an embryonic stage but it has been conceived to be extensible. HoneyEVSE has been particularly inspired by two previous projects, HoneyPLC [7] and ICSPot [8]. The former introduces the concepts of protocol emulation based on Honeyd [9][9] while the latter embeds a physical process emulation of a water tower.

A. Architecture

HoneyEVSE avails itself of a series of third-party tools to implement a physical process simulation and some communication protocols that are usually present in ICS environments. Specifically, the former requirement was achieved by using ACNPortal (which will be analyzed afterwards in the section V-C), while the latter objective was completed by integrating Honeyd. The aim of Honeyd is to create virtual hosts on a network, run arbitrary services and modify their behaviors to the extent of simulating the execution of certain operating systems. We developed protocol emulations for TCP/IP, UDP, Telnet (and SSH) and FTP, as we expect that a real EVSE platform would implement all of them. More specifically, TCP/IP is likely to be used to manage APIs available to the web app and the HMI, while Telnet (and SSH) can provide remote management of the server and FTP might be used to handle file uploads such as profile pictures or firmware updates.
The honeypot also exposes two web services, respectively a Web application that manages the interaction of the users of the charging stations and an HMI that responds to the needs of the administrators of the ICS in terms of monitoring and possible maintenance.
The mapping of the ports follows the one of a typical production server environment (for reasons related to the deception of network reconnaissance tools, such as Nmap[10]) and is drawn as follows:

- port 80: exposes a simple Siemens web interface of the PLC;
- port 21: exposes the FTP connection;
- port 23: exposes for the Telnet connection;
- port 5000: exposes the HMI and the web app.

We chose to represent HoneyEVSE as a Siemens Simatic 300 profile[11] as it was the only model that was able to perfectly fool the most common reconnaissance tools.

```
No exact OS matches for host.
TCP/IP fingerprint:
OS:SCAN(V=7.80%E=4%D=2/19%OT=22%CT=1%CU=1%PV=Y%DS=2%DC=T%G=Y%TM=63F254AD%P=
OS:x86_64-pc-linux-gnu)SEQ(SP=105%GCD=1%ISR=106%TI=Z%CI=Z%II=I%TS=A)OPS(O1=
OS:M5B4ST11NW7%O2=M5B4ST11NW7%O3=M5B4NNT11NW7%O4=M5B4ST11NW7%O5=M5B4ST11NW7
OS:%O6=M5B4ST11)WIN(W1=FE88%W2=FE88%W3=FE88%W4=FE88%W5=FE88%W6=FE88)ECN(R=Y
OS:%DF=Y%T=40%W=FAF0%O=M5B4NNSNW7%CC=Y%Q=)T1(R=Y%DF=Y%T=40%S=O%A=S+%F=AS%RD
OS:=0%Q=)T2(R=N)T3(R=N)T4(R=Y%DF=Y%T=40%W=0%S=A%A=Z%F=R%O=%RD=0%Q=)T5(R=Y%D
OS:F=Y%T=40%W=0%S=Z%A=S+%F=AR%O=%RD=0%Q=)T6(R=Y%DF=Y%T=40%W=0%S=A%A=Z%F=R%O
OS:=%RD=0%Q=)T7(R=Y%DF=Y%T=40%W=0%S=Z%A=S+%F=AR%O=%RD=0%Q=)U1(R=Y%DF=N%T=40
OS:%IPL=164%UN=0%RIPL=G%RID=G%RIPCK=G%RUCK=8D5F%RUD=G)IE(R=Y%DFI=N%T=40%CD=
OS:S)
```

Fig. 5. Result of a Nmap scan

At last, HoneyEVSE provides custom logs for every interaction that users perform on the platform (limited to logins and sign-ups) in order to capture information about the data inserted by eventual attackers that can be used for further analysis of the explored vulnerabilities.

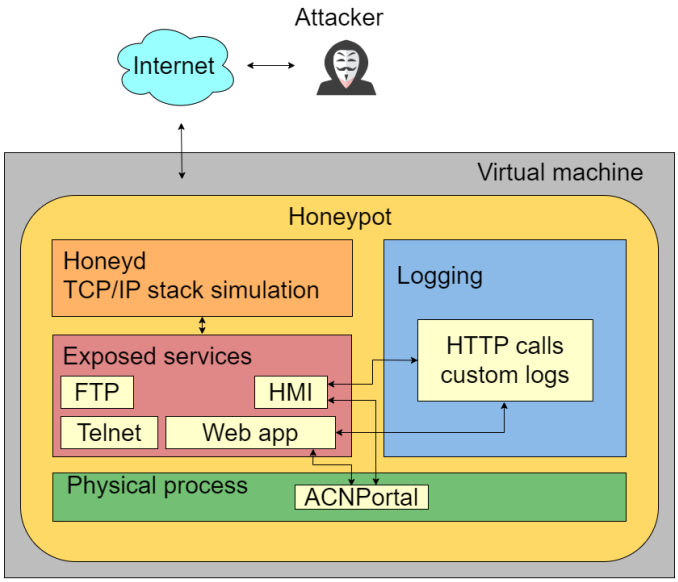

Fig. 6. Architectural scheme of HoneyEVSE

B. How to execute the honeypot

HoneyEVSE has a makefile that handles its setup and start. To launch the honeypot the user needs to:

[9]https://www.honeyd.org/

[10]https://nmap.org/

[11]a set of information in order to identify a specific model of controller to be emulated

1) Sign up to ACNportal at the following link https://ev.caltech.edu/login
2) Obtain an API key for ACNPortal
3) paste it in the makefile by replacing the placeholder "YOUR API KEY"
4) open the terminal inside the folder /HoneyEVSE/evse
5) run the command `make evse`
6) open a browser on localhost:5000 to see the root page

The makefile allows to execute the following operations

1) creation of the .env file that stores the API key for the production environment
2) generation of the charges by running the simulation (managed by the module /HoneyEVSE/evse/utils/generate_charges.py, started by the wrapper init.py
3) terminal instance creation that runs the command `flask run` in order to deploy the web application

C. Physical process emulation

As anticipated before, HoneyEVSE avail itself of ACNPortal to simulate an ongoing physical process. ACNPortal is a collection of tools to help researchers and other stakeholders understand the challenges of large-scale EV charging and develop practical solutions to those challenges. Its dataset is based on the measurements collected on a real set of smart charging stations (which are managed by a scheduling algorithm to optimize the charging time of multiple vehicles linked to it) installed at the Caltech Institute of technology (and later on in other sites).
The tool itself takes a determined period of time and collects the following data about a charging session:

- `_id`: unique identifier for the session record
- `chargingCurrent`: time series of the measured current draw of the EVSE during the session
- `clusterID`: unique identifier for a subset of EVSE at a site, such as a single garage
- `connectionTime`: timestamp that identifies the moment when the electric vehicle is plugged in
- `disconnectTime`: timestamp that identifies the moment when the electric vehicle is unplugged
- `doneChargingTime`: timestamp that identifies when the last non-zero current draw is recorded
- `kWhDelivered`: amount of energy that was delivered during the charging session
- `pilotSignal`: time series of the pilot signals passed to the EVSE during the session
- `sessionID`: unique identifier for the session
- `spaceID`: unique identifier for the parking space
- `stationID`: unique identifier for the EVSE
- `timezone`: timezone of the site, based on pytz format
- `userID`: unique identifier for the user
- `userInputs`: list of inputs provided by the user

With this dataset, ACNSim [4] (a simulator included in the ACNPortal framework) is able to emulate a charging session that uses a scheduling algorithm (passed as a parameter) to the extent of delivering electricity to the various instances of charging stations for the different vehicles.
Unfortunately, the main concern of ACNPortal involves the representation of the final status of a charging session, but at the current state of the art, this framework was the best choice available to us, even though it fails to provide us essential information about the intermediate state of the recharge (e.g. snapshot of the delivered quantity of charge in a determined moment of the charging session) and about the requirements of the vehicle at its arrival to the charging station (e.g. the requirements of the battery for its recharge, as they depend on the manufacturers' implementation).
Furthermore, the data set presents non significant data about the initial status of the battery, as all the batteries that were inserted in the data set have initial charge equal to zero. This is indeed a non-plausible scenario as we do not expect users to recharge the batteries of their electric vehicles only when they are fully discharged.
These limitations required to properly deal with some aspects that were essential to elaborate a proper simulation of a single charging session of an electric vehicle.
Both the concerns described above were addressed in HoneyEVSE. In particular, the snapshots of the intermediate statuses of the charging session were simulated by re-running the whole charging emulation and monitoring the following parameters during every minute of the process. In particular we were interested into saving:

- the required energy
- the current charging ratio delivered by the charging system
- the remaining energy required to fill the battery at its maximum capacity
- the percentage of completion of the recharge (calculated by us)
- an estimate of missing time (in minutes) needed to complete the recharge
- the real-time costs (expressed in €) associated to the recharge

Instead, in order to simulate a plausible initial status of the battery (expressed as its percentage of charge), we implemented the following formula (that is also used to calculate the percentage of completion of the recharge during the monitoring): $100 - self.discharge - self.remaining\_demand/self.requested\_energy$
where the `discharge` factor is a random value included between 40 and 80 that indicates a percentage to be subtracted to obtain a plausible percentage of the charge of the battery, `remaining_demand` indicates the quantity of charge that is required in that instant to complete the charging session and`requested_energy` is the parameter (taken by the data set) that indicates the quantity of charge required to complete the recharge at the beginning of the simulation.

D. Main features and available interactions

Due to the large number of expected functionalities and their possible variation related to their implementation among the

three different classes, we decided to restrict our focus to the development of a honeypot for EVSE belonging to the second classIII-B described in the taxonomy.
In particular, we decided to develop a user-side web application which implements the basic flow that allows the interaction with an almost real-time graph that tracks the state of a recharge of an electric vehicle.
HoneyEVSE includes a couple of functionalities that allow users to actively interact with the web application. These features were conceived in order to see whether attackers may try to perform attacks such as SQL injection or XSS (Cross Site Scripting).
At its current state, HoneyEVSE presents four main web pages shown to its users, respectively accessible from the following routes: /login, /register /home (which is actually the root of the web application) and /admin. As suggested by the names, in the first two paths HoneyEVSE allows to simulate logins and sign-ups. Through the provided forms, an attacker may try to inject malicious SQL queries (e.g. with the intent to erase the user list of the application) or code (e.g. Javascript or other client side programming languages code that alters the behaviour of a page with the purpose of stealing personal information or installing malwares on the users' browsers).
The signup flow allows to reach the /home route automatically once the user fulfills the form, while every login attempt is rejected. This flow simulates with the lowest possible effort the ban of a user that tried to perform some malicious operations on the platform. Indeed, this is not the expected outcome of a sign-in functionality, but it is acceptable as the only goal of the two forms is to try and see whether any code injection attempt was carried out.
The /home route shows an almost real-time interface available to the service users that shows the following data:

- Estimated time for recharge completion
- Total quantity of energy delivered during the charging session (in KwH)
- Vehicle status (connected, disconnected, under recharge)
- Total cost associated to the charging session
- Intensity of the current delivered (in Ampère)
- Percentage of completion of the charging session

The total cost of the recharge is calculated as the total amount of energy delivered * 0.60€/kwh, which is the medium cost per khw for the service supply in real EVSE.
In conclusion, the /admin route embeds all the data shown in the user interface plus the id and the time of arrival of the vehicle and a graph. In particular, the graph is updated every second and shows the quantity of charge delivered by a charging station, which is actually constant until the end of the recharge as no scheduling algorithm is running on the observed charging station.

E. Logs

As previously stated, the necessity of retrieving information about the operations performed by the attacker requires by definition a way to log all the interactions carried out on the web application, in order to collect data samples that might be relevant for further analysis.
HoneyEVSE stores every kind of HTTP request triggered by the buttons inserted in the pages of the web app. In particular, the honeypot logs the register and login attempts by including the following information:

- a timestamp (based on the location of the host server)
- the kind (restricted to 'GET' and 'POST' methods) of the HTTP request
- the parameters of the request (the data taken from the two forms)
- the HTTP response status code emitted from the server
- IP of the device

Logs are both displayed on the terminal of the hosting server and saved into a .log file.

```
2023-02-18 16:25:11,160 INFO app :
Trying to login
with email: mb@studenti.unipd.it
and password: TestForLog123 from 127.0.0.1
2023-02-18 16:25:11,161 INFO werkzeug :
127.0.0.1 - - [18/Feb/2023 16:25:11]
"POST /login HTTP/1.1" 200 -

2023-02-18 16:25:49,501 INFO app :
Trying to register
with name: Massimiliano, surname: Baldo,
email: mb@studenti.unipd.it,
password: TestForLog123,
confirm_password: TestForLog123
from 127.0.0.1
2023-02-18 16:25:49,501 INFO werkzeug :
127.0.0.1 - - [18/Feb/2023 16:25:49]
"[32mPOST /register HTTP/1.1[0m" 302 -
```

Fig. 7. Example of log stored by HoneyEVSE

VI. Conclusions

HoneyEVSE has not still been deployed yet, as we are planning to enhance the number of functionalities and their correct implementation according to the expectations of real users. Even though the honeypot is not currently exposed on the Internet, its current status can be considered as a first milestone that was achieved during the development of this project.

A. Current limitations

In its current status, HoneyEVSE still presents many limitations related to the restricted set of functionalities and protocols that is currently available on the platform.
Furthermore, as described in the section Main features and available interactions V-D, the login flow is not implemented according to the expectation of a common user, the reset

password is missing and an hacker can easily expose the honeypot as there is no error returned whether the fields password and confirm password do not coincide.
Moreover, the UI lacks of reliability in resembling the one of a real web application and the honeypot itself has not been deployed yet.
In conclusion, a real user of an electric vehicle (or a well informed attacker) can also deceive the honeypot as there is no splitting in costs associated to the parking of the vehicle and the costs associated to the effective delivery of electric current. In fact, while the former are lower during the recharge period but consistently higher once the battery is fully charged, the latter are higher during the recharge period but turn to zero once the battery is charged and also may vary among the different kind of service delivered by the EVSE (for example whether a vehicle uses AC or DC current).
All these limitations are related to the amount of time needed to effectively develop, test and deploy a high-interaction honeypot, which is indeed a project that demands considerable resources both in terms of costs and time.

B. Extensibility

Further development to bring the product in a deployable and highly-interactive state is certainly needed, as for example a user is expecting to visualize most of the functionalities described in the section EVSE and related usecases II.
The most challenging but captivating perspective for HoneyEVSE enhancement involves the implementation of file pickers, in order to allow any kind of malware update on the platform. File uploading can in fact create more complex and heavy attacks (e.g. ransomwares, trojans, ecc...) and was considered for further development on the user-side by implementing a profile picture upload while, on the admin side, a picker for the uploading of new versions of the firmware could represent a valuable solution.
Moreover, generating more reliable data during the simulation can be a step over to obtain a better emulation of the EVSE behaviour. This can be achieved using a performing schedule algorithm for charging EVSE (which is already embedded in the ACNPortal framework or can be manually implemented), to the extent of emulating the functioning of more charging stations managed by a single interface (which is a more plausible scenario for the administrative side and for the ICS itself).
In conclusion, even though the payment transaction logic is usually held by third-party services, it would be interesting to emulate a payment gateway internal to the web application in order to see whether an attacker would be interested into attempting any kind of information theft on it.

REFERENCES

[1] J. Franco, A. Aris, B. Canberk, and A. S. Uluagac, "A survey of honeypots and honeynets for internet of things, industrial internet of things, and cyber-physical systems," *IEEE Communications Surveys Tutorials*, vol. 23, no. 4, pp. 2351–2383, 2021.

[2] J. Matherly, "Complete guide to shodan," 10 2016. [Online]. Available: https://ia800705.us.archive.org/17/items/shodan-book-extras/shodan/shodan.pdf

[3] M. Baldo, A. Trevisan, F. Turrin, and T. Bianchi, "Honeyevse," 02 2023. [Online]. Available: https://github.com/massimilianobaldo/HoneyEVSE

[4] Z. Lee, S. Sharma, D. Johansson, and S. Low, "Acn-sim: An open-source simulator for data-driven electric vehicle charging research," *IEEE Transactions on Smart Grid*, vol. PP, 12 2020.

[5] R. P. . F. T. Mauro Conti, Denis Donadel, "Evexchange: A relay attack on electric vehicle charging system," *link.springer.com*, 2022.

[6] A. Brighente, M. Conti, D. Donadel, and F. Turrin, "Evscout2.0 : Electric vehicle profiling through charging profile," *ACM Transactions on Cyber-Physical Systems*, 09 2022.

[7] E. López-Morales, C. Rubio, A. Doupé, Y. Shoshitaishvili, T. Bao, and G.-J. Ahn, "Honeyplc: A next-generation honeypot for industrial control systems," 10 2020, pp. 279–291.

[8] M. Conti, F. Trolese, and F. Turrin, "Icspot: A high-interaction honeypot for industrial control systems," 07 2022, pp. 1–4.

[9] N. Provos, "Honeyd: A virtual honeypot daemon (extended abstract)," 01 2003.