Security Analysis of Cyber-Physical Systems

Nate Olsen
University of Virginia
Charlottesville, USA
nso4wg@virginia.edu

Matthew Jordan
University of Virginia
Charlottesville, USA
mrj3dd@virginia.edu

Soneya Hossain
University of Virginia
Charlottesville, USA
sh7hv@virginia.edu

Daniel Perez Lazarte
University of Virginia
Charlottesville, USA
dep7sc@virginia.edu

***Abstract*—Today's society is more interconnected than ever. Every year, the amount of Internet of Things (IoT) devices that users possess continues to increase exponentially. Within this population, a fair amount of these devices are classified as Cyber-physical systems (CPS). These CPS systems can have numerous uses and are utilized across various difference industries, including healthcare, automobiles, agriculture and urban development. While these devices have allowed for users to live their lives in a more efficient manner, they also bring their own set of limitations. With CPS having the ability to process raw user information and the ability to function autonomously, how can we ensure that these devices are secure? In this paper we preformed a security analysis on 21 cyber physical systems and found that CPS are in need of rigorous testing due to our analysis of system vulnerabilities.**

***Index Terms*—Security, Cyber-physical Systems, Python, Verification, Bandit**

I. Introduction

In the past decade, we have seen the widespread rise of the Internet of Things (IoT). The term "Internet of Things" refers to the concept that interconnected devices, devices that are connected to the Internet and other similar devices, are being embedded in everyday appliances. In 2019, a Gartner report estimated that over 14.2 billion connected IoT devices had already been launched and estimates that, by 2021, there will be over 25 billion IoT devices [1]. This report demonstrates that the growth of the IoT industry will continue to rise in the near future. This, however, is not the only industry that has grown in recent times. Network and sensor technologies have also continued to advance at a steady pace and are increasing used in the construction of new state-of-the-art devices. As a result, these advancements in sensor and communication networks have allowed for Cyber-Physical systems to be incorporated into a wide variety of settings. Some examples of settings include the healthcare sector, which introduced "smart-health" systems that aid in the data collection and therapeutics of patients with certain conditions, and the automobile sector, which is focusing its efforts on producing autonomously driving vehicles. While this rise of Cyber-physical systems is generally seen as a positive, there have been concerns relating to security and privacy that have been raised about the use of such devices.

In 2014, Tesla revolutionized the automobile industry with the introduction of autonomous driving vehicles. When the system was first launched, however, it raised some eyebrows due to the uncertainty of how safe the system was, both in terms of software and in terms of users' safety. Although Cyber-physical systems can open doors to technological advancements, they can also have their share of limitations. As these systems are processing raw user information and conducting actions based on this data, they can potentially be exploited by malicious third parties. As such, exploiting a vulnerability on a vital Cyber-physical system, such as an autonomous vehicle, could potentially injury or even kill a given user. Therefore, safeguards to identify and protect against these types of flaws are critical to develop.

In this paper, we dive into the issue of security in Cyber-physical systems and address several code vulnerabilities that were discovered within 21 chosen Cyber-physical systems. These CPS were sampled from a wide variety of locations and range from technologies such as autonomous steering models and to other IoT systems. During our research, we used the popular static analysis tool "Bandit" to isolate security vulnerabilities within a given Cyber-physical system's code. Bandit accomplishes this by building an Abstract Syntax Tree (AST) from the provided python input files. It then runs specific plugins on the AST to develop a report that highlights all the security vulnerabilities that were found in the code.

The contributions in this paper are as followed:

- To explore popular open source CPS systems from GitHub and to select projects across a range of different platforms.
- To discover security vulnerabilities by using Bandit to perform static analysis and to categorize the results.
- To analyze all of the reported issues and to raise conclusions based on our experimental results.

This paper is structured as follow: Section II will describe related works, Section III will discuss method, Section IV will discuss experimental results, Section V will discuss about challenge and limitations, and we conclude our paper in Section VI.

II. Related Work

There has been a lot of work done in the sphere of verification in cyber physical systems. In Yuchi et al, they developed and evaluated their own systematic testing tool called DeepTest [2]. DeepTest focused on automatically detecting unusual and erroneous behavior in autonomous vehicles that uses deep-neural-networks. These discovered erroneous behaviors were generally issues that would result in fatal car crashes. They were able to discover these behaviors by using automatically

generated tests that simulated real-world driving conditions like rain, fog, or lightning. These driving conditions were achieved by distorting the images fed into the DNN [2]. There has also been work done where they have explored the code of autonomous vehicles.

In Nie et al., researchers were able to hack into the Tesla Model S and get direct access to a lot of its internal features [3]. As such, they were able to exploit vulnerabilities in the network system, Linux kernel and the Javascript code. With their findings, they sent a report to Tesla, who quickly patched the raised vulnerabilities. As this Tesla model was already in the possession of many customers, a malicious attack on the car could have caused serious harm to the user. Verification has not only been limited to smart cars, however, as it has also been heavily surveyed within the scope of smart cities.

In AlDairi, the authors surveyed the potential risks and dangers in the development of smart cities by addressing the potential issues that arise in terms of security and privacy. [4] Since smart cities involve complex interconnected systems, it is important that all the information processed is secure and private. They highlight that vulnerabilities must be taken into account in the development of the smart city and should continue to be a focus after the city has been developed.

III. METHODS

A. Collecting Open Source CPS Projects from GitHub

For our project, we wanted to select a wide variety of platforms in order to observe which security vulnerabilities are common across different platforms. In order to do so, we decided to select a series of popular Cyber-physical systems projects from GitHub. Specifically, we looked for high-starred python projects that came from diverse platforms. For example, our chosen GitHub projects involve projects that relate to self-driving cars, self-driving car simulation, home assistant systems, autonomous drone shipping, drone simulators, and autonomous swarm missions. However, we didn't want to focus solely on projects, as we also wanted to include testing on python libraries that are widely utilized in major Cyber-Physical systems and other CPS frameworks. As such, we selected the following libraries to be tested: "PythonRobotics" - a library for robotics algorithms, "PyAdvanceControl" - a library for autonomous control, and "pyvit" - a tool kit for interfacing with cars written in Python. For our experiment, we selected frameworks relating to Cyber-Physical system real time simulation, such as Mini-CPS, frameworks for custom personal assistant system creation, such as Kalliope, and frameworks for healthcare imaging. Our complete list of projects', libraries' and frameworks' names, short descriptions and GitHub links can be found in Table I.

B. Running Static Analysis Using Bandit Tool

In the beginning, we intended to use "Dynamic Symbolic Testing" for our research purposes. However, due to tool limitations and the hardware requirements that were required in order to run the code, using this method was deemed infeasible. Therefore, we switched to performing static analysis

TABLE I
CPS PLATFORMS DESCRIPTION

Platform: Self-driving-car steering model **Description:** Steering models are written in python with deep learning models to predict steering angle. We tested four steering models: Autumn, cg23, chauffeur, rambo **GitHub Link:** https://udacity.com/self-driving-car
Platform: Home-assistant **Description:** This is an open source home automation system designed to provide local control, and ensure privacy; modular approach makes other devices or actions to be implemented easily. **GitHub Link:** https://github.com/home-assistant/core
Platform: MONAI **Description:** This is a Python-based, open-source framework for deep learning in healthcare imaging. **GitHub Link:** https://github.com/Project-MONAI/MONAI
Platform: Powerwheels Racing Autonomous Vehicle Model **Description:** Autonomous Vehicle Project at Fubar Labs for the Autonomous Powerwheels Racing competition. **GitHub Link:** https://github.com/dvbuntu/autonomous
Platform: MiniCPS **Description:** MiniCPS is a framework for Cyber-Physical Systems real-time simulation. It includes support for physical process and control devices simulation, and network emulation. **GitHub Link:** https://github.com/scy-phy/minicps
Platform: pyvit **Description:** pyvit is a toolkit for interfacing with cars from Python. It aims to implement common hardware interfaces and protocols used in the automotive systems. **GitHub Link:** https://github.com/linklayer/pyvit
Platform: Kalliope **Description:** Kalliope is a framework that offers to create custom personal assistant system; expending the existing modules and codes, it is possible to develop custom Kalliope bot. **GitHub Link:** https://github.com/kalliope-project/kalliope
Platform: Smart-IoT-Planting-System **Description:** Smart-IoT-Planting-System is an intelligence/smart/automatic planting system in the agricultural greenhouse which is base on IoT technology. It consists of sensors, terminal device(STM32 MCU), gateway(RPi), Web server. **GitHub Link:** https://github.com/Python-IoT/Smart-IoT-Planting-System
Platform: Azure-iot-sdk-python **Description:** A Python SDK for connecting devices to Microsoft Azure IoT services. **GitHub Link:** https://github.com/Azure/azure-iot-sdk-python
Platform: Self-driving car simulation **Description:** This project is a simulation of a self driving car using machine learning techniques and tkinter. **GitHub Link:** https://github.com/hccoffin/Self-Driving-Car-Simulation
Platform: Drone-shipping **Description:** This is an open source project that intended to help people to create autonomous drone missions that operate with a pixhawk controller. **GitHub Link:** https://github.com/galprz/Drone-shipping
Platform: AirSim **Description:** AirSim is a simulator for drones, cars and more, built on Unreal Engine (we now also have an experimental Unity release). **GitHub Link:** https://github.com/microsoft/AirSim
Platform: Autonomous_UAVs_Swarm_Mission **Description:** This is an autonomous quadcopter swarm project. The purpose of this project is to achieve a designated mission by multiple drone cooperation. **GitHub Link:** https://github.com/AlexJinlei/Autonomous_UAVs_Swarm
Platform: DynamicAttentionNetworks **Description:** Dynamic Attention Networks For Time Series State Forecasting in Cyber Physical Systems. **GitHub Link:** https://github.com/nmuralid1/DynamicAttentionNetworks
Platform: SMT-Based CPS Parameter Synthesis and Repair **Description:**ParSyn is a flexible software framework for parameter synthesis and repair of cyber-physical systems. **GitHub Link:** https://github.com/hriener/parsyn-cegis
Platform: PythonRobotics **Description:** This is a Python code collection of robotics algorithms, especially for autonomous navigation. **GitHub Link:** https://github.com/AtsushiSakai/PythonRobotics
Platform: PyAdvancedControl **Description:** Python codes for advanced control **GitHub Link:** https://github.com/AtsushiSakai/PyAdvancedControl

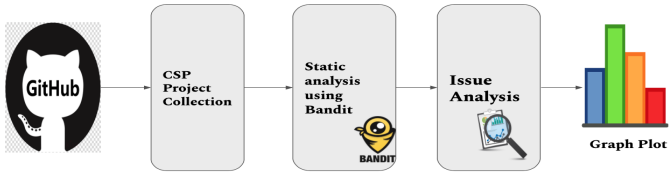

Fig. 1. Work Flow of Our Experiment

on the systems instead. Since static analysis performs analyses without running the actual program itself, this allowed for our experiments to be feasibly accomplished with respect to the Cyber-physical systems.

C. Bandit Tool

The Bandit tool [5] is designed to find common security issues in Python code. Bandit does not run dynamic execution, rather it only performs static analysis on the Python code base. In order to do so, Bandit processes each file, builds an AST from it, and then runs appropriate plugins against the produced AST nodes. Once Bandit has finished scanning all the files, it generates a readable report. Bandit supports many different tests to detect various security issues in python code. These tests are created as plugins and new ones can be created to extend the functionality offered by bandit. Initially, Bandit has 7 plugins groups: B1xx-misc tests, B2xx-application/framework misconfiguration, B3xx-blacklists (calls), B4xx-blacklists (imports), B5xx-cryptography, B6xx-injection, and B7xx-XSS. Under these plugin ids, Bandit has 34 test plugins. By running these 34 test plugins on our given AST nodes, Bandit can detect a wide range of common security issues. For example, Bandit is capable of detecting the following security vulnerabilities: SQL-injection, Hard-coded SQL expressions, Weak cryptographic keys, SSL without host key verification, hard-coded passwords, Hard-coded temp directories, Linux commands wildcard injection and more. All these issues can cause severe security risks if exploited. Furthermore, since Cyber-physical Systems can be closely connected to a given user's health, the safety and security of these systems are critical. Therefore, the crux of our work is to accumulate and demonstrate the different types of security issues found within CPS platforms.

D. Issue Analysis

Once we had run the Bandit static analysis on our projects, the next step was to manually analyse the reported issues. We did this in order to distinguish our real data from real issues and false positives. From this, we categorized our results in terms of the issue's severity and Bandit's confidence in the results. Under severity, the issues were categorized into low, medium and high severity, based on their potential security risk. Similarly, under confidence, the issues were categorized into low, medium and high confidence issues, based on Bandit's level of confidence in the results.

Severity	Number of Issues
Undefined	0
Low	34150
Medium	166
High	58

Confidence	Number of Issues
Undefined	0
Low	0
Medium	401
High	33973

Fig. 2. Number of issues by severity and confidence

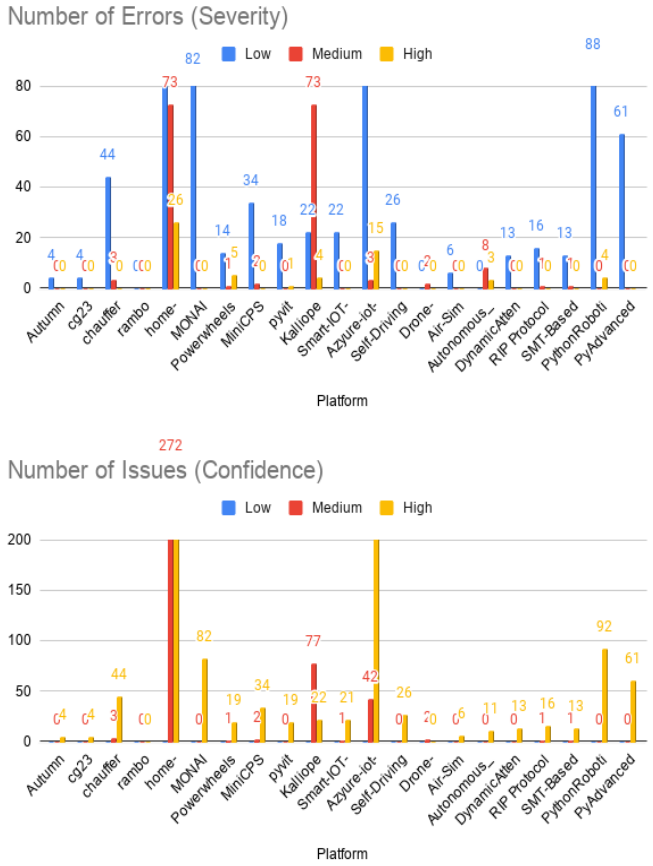

Fig. 3. Number of issues of each platform

IV. Results

With our approach, we found that there were a number of different errors reported by Bandit. Figure 2 shows the total number of issues reported by both severity and confidence of all twenty-one platforms we tested. Figure 3 graphs the number of issues for each platform. From this we can see that the majority of issues came from the home-assistant platform. We found this platform had a total of 30,340 issues found by bandit so it is an outlier in terms of number of issues found.

To highlight some of the issues that were discovered by Bandit, we will discuss some of the common errors found. A

full list of issues can be found on our project GitHub page [6].

A. Issue B605

This issue is raised when Bandit finds that a platform tries to start a process with a shell. This is commonly done in python using `os.popen` or `os.system` functions to run shell commands. This is often done in cyber physical systems to get some form of data from the machine on its current status and can use use variables defined in the program as input to the process. This has the potential for attackers to inject code to be run as shell which can give them more access to the system and it's underlying processes. This is a high severity issue due to the fact that injection attacks are widely used and have the potential to leak data or have a malicious program to be run.

This was the most common high severity issue reported by Bandit with a total of twenty-one reports across five different platforms.

```
>> Issue: [B605:start_process_with_a_shell]
     Starting a process with a shell, possible
     injection detected, security issue.
   Severity: High   Confidence: High
   Location: /Users/soneyabintahossain/
       cps_project_test/home-assistant-core/
       homeassistant/scripts/macos/__init__.py
       :33
   More Info: https://bandit.readthedocs.io/en
       /latest/plugins/
       b605_start_process_with_a_shell.html
33 os.popen(f"launchctl load -w -F {path}")
```

B. Issue B322

The next most common issue found by Bandit is issue B322. This issue is raised when Bandit finds that the program uses the `input` function to read standard input from a user. This is problematic for platforms running Python 2 because it will run the resulting string as python source code, which can lead to an injection attack. However, `input` is safe to use when using Python 3. This was our first encounter with a false positive generated with Bandit. Because Bandit is a static analysis tool, it is unable to differentiate between programs being run with Python 2 or Python 3. We tried running this with both the Python 2 version and the Python 3 version of Bandit and it was still unable to correct this. For applications like Home-Assistant and Power-Wheels Racing, both of these programs are run using Python 3 so this is not actually a significant error. However, given that Bandit cannot differentiate between the two version it still reported this issue as having high confidence, meaning that the tool was confident this is an error with the program.

```
>> Issue: [B322:blacklist] The input method in
     Python 2 will read from standard input,
     evaluate and run the resulting string as
     python source code. This is similar,
     though in many ways worse, then using eval
     . On Python 2, use raw_input instead,
     input is safe in Python 3.
   Severity: High   Confidence: High
   Location: /Users/soneyabintahossain/
       cps_project_test/power-wheel/attic/
       PiCamCollect.py:42
   More Info: https://bandit.readthedocs.io/en
       /latest/blacklists/blacklist_calls.html
       #b322-input
41 camera.start_recording(collector, format='
    rgb')
42 input('Press enter to stop recording') #
    will cause hang waiting for user input
43 camera.stop_recording()
```

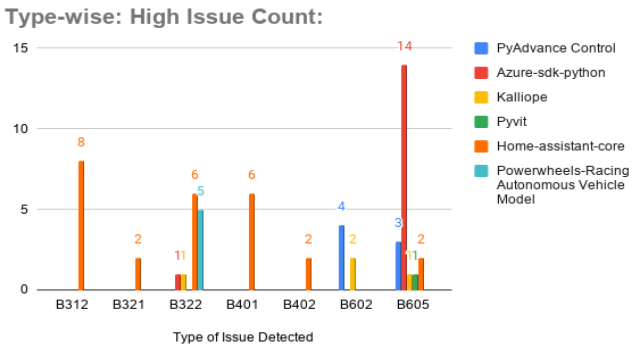

Fig. 4. Type-wise High Severe Issue Count

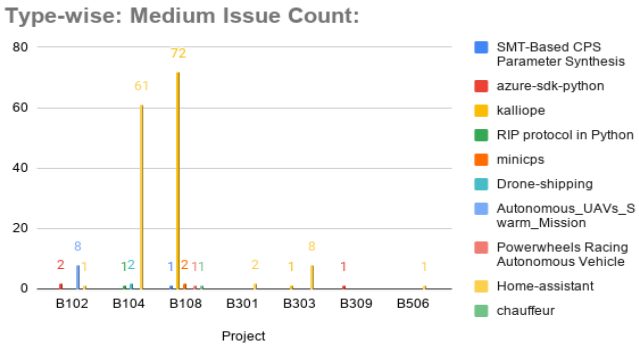

Fig. 5. Type-wise Medium Severe Issue Count

C. Issue B108

Another issue that we found that was both interesting and common across projects was Issue B108. This issue refers to when Bandit detects that a program insecurely uses a temporary file or directory to store or read from files. Typically this happens in a program when a file needs to be saved and then converted to another file type in the program or the data is too large to be stored and therefore needs to be saved. From Figure 5 we can see that this is the most common Medium error that we encountered in our testing of cyber physical systems with a total of 76 errors across 5 different projects. This error is problematic because an attacker can have access to a temp directory on most file systems as it does not require

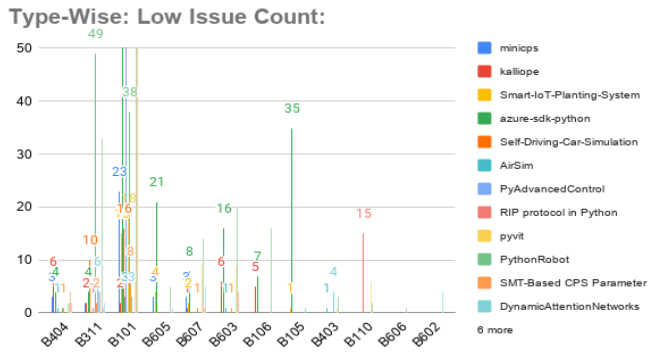

Fig. 6. Type-wise Low Severe Issue Count

any extra permission to access. If an attacker realizes that a program is reading from a temporary file, they have the ability to overwrite that file during run-time and cause the program to read incorrect data.

For example, the largest offender of this type was the platform Kalliope. This is a personal assistant app similar to Apple's Siri or Amazon Alexa. The program will read a audio .wav file from the temp/ directory of the file system and use that as input to be parsed. An attacker could modify the wav file and make it so that file is corrupted so it no longer works or have it say something completely different. Because many elderly people use personal assistants, if an elderly person is injured and cannot reach their phone to call for 911, they will try to use their personal assistant to call. If an attacker corrupts the audio file then that could be impossible for them to do. They also have the ability to read from this file an attacker could listen in to every conversation that a person has with their personal assistant, resulting in a large invasion of privacy.

```
>> Issue: [B108:hardcoded_tmp_directory]
   Probable insecure usage of temp file/
   directory.
   Severity: Medium   Confidence: Medium
   Location: /Users/soneyabintahossain/
       cps_project_test/kalliope/Tests/
       test_api/test_synapse_view.py:253
   More Info: https://bandit.readthedocs.io/en
       /latest/plugins/
       b108_hardcoded_tmp_directory.html
252 # Scenario 1 : input wav file
253       temp_file = "/tmp/kalliope/tempfile.
   wav"  # tempfile.NamedTemporaryFile(suffix
   =".wav")
254       result_file = self.flask_api.
   synapses_blueprint._convert_to_wav(
   temp_file)
```

V. Discussion

In this paper, we discovered that most, if not all, of current Cyber-Physical Systems are prone to some level of security vulnerabilities. While most of these discovered flaws are seemingly insignificant, we were able to find many cases of issues where the severity was quite large. If these issues continue to go unchanged, these given systems are leaving themselves vulnerable to nefarious adversaries. Our research lends itself into the fact that rigorous testing of software is necessary in order to limit the number of issues that the model produces. It also raises the issue of being wary of software that you can find online. All of the systems detailed in this paper are open-source, meaning that anyone can search for these online and use them in their own unique ways. We can oftentimes assume that a widely popular piece of software/system must be safe based solely on its popularity with other developers. However, by using these systems in your own individual projects, you are setting yourself up to have the same number of security issues. Therefore, we have demonstrated the importance of testing in Cyber-physical Systems and demonstrated the levels of issues found in several popular Cyber-physical Systems sourced from Git-Hub.

A. Challenges and Limitations

Initially, we intended to test Udacity:Self-driving-car steering models using dynamic symbolic execution. As all the models are python implementation with deep leaning based model to predict steering angle, we needed to find a suitable Python Symbolic testing Engine for that. There are a very few symbolic testing tool for Python. One of the main reasons for the limited symbolic execution tool is that Python is not a typed language, therefore, it is quite challenging to perform symbolic execution on Python code base. However, we found one Python symbolic engine called PyExZ3 [7]. We found that the installation guidelines and documentations are very poor and this tool does not support complex functionalities and data structures. Furthermore, due to the deep learning based models we failed to perform symbolic execution. It is also an open and challenging problem to verify programs with deep models as we all know that.

Moreover, another challenge is that many open source Cyber-Physical Systems require actual hardware to run. Our initial aim was to run dynamic symbolic execution, which is a powerful verification technique, therefore, we failed to test those CPS systems due to the lack of hardware. Finally, we decided to test those CPS systems using static analysis that does not require to run the program.

Furthermore, for 21 CPS projects, we get a total of 35,000 issues reported by Bandit. One of the biggest challenges was to investigate those issues manually. Therefore, another limitation in our project is the production of false-negatives and false-positives produced by Bandit. While our general level of confidence for each system was high, we cannot determine for certain if all of the issues found in the course of our research are truly issues. Therefore, while this gives a good benchmark for how many issues are in the given system, it cannot definitely say how many true issues actually exist. We were also only able to fully test 21 different systems, which is only a small sample size of the total number of Cyber-physical systems out in the market.

VI. Conclusion

As the field of Cyber-physical systems continues to evolve and become more ingrained in our day-to-day lives, the need to ensure that these systems are secure will also need to continue to rise. As a result of our research, we demonstrated that 21 different systems could be subjected to low, medium and high levels of security issues with a high level of confidence. In conclusion, we discovered that Cyber-Physical Systems are still in need of rigorous testing to best ensure the security and well-being of their users.

A. Contributions

Team members' individual contributions:
sh7hv: Tested 13 projects, analyze and categorize issues for graph representation, wrote method section and few limitations.
nso4wg: Tested 8 projects, wrote results section.
mrj3dd: Created graphs, wrote up the challenges/limitations/conclusion section, and proofread/edited the paper.
dep7sc: Analysis/Writing/Editing

References

[1] Gartner, "Gartner identifies top 10 strategic iot technologies and trends," 2018.Available: https://www.gartner.com/en/newsroom/press-releases/2018-11-07-gartner-identifies-top-10-strategic-iot-technologies-and-trends

[2] Tian, Yuchi, et al. "Deeptest: Automated testing of deep-neural-network-driven autonomous cars." Proceedings of the 40th international conference on software engineering. 2018

[3] Nie, Sen, Ling Liu, and Yuefeng Du. "Free-fall: Hacking tesla from wireless to can bus." Briefing, Black Hat USA (2017): 1-16.

[4] AlDairi, Anwaar. "Cyber security attacks on smart cities and associated mobile technologies." Procedia Computer Science 109 (2017): 1086-1091.

[5] Bandit. "https://bandit.readthedocs.io/en/latest/plugins/".

[6] Project-Link. "https://github.com/soneyahossain/CPS-testing-with-Bandit".

[7] PyExZ3. "https://github.com/thomasjball/PyExZ3".