An Empirical Study of Vulnerabilities in Robotics

Kaitlyn Cottrell*, Dibyendu Brinto Bose†, Hossain Shahriar‡, and Akond Rahman§
*§Department of Computer Science, Tennessee Technological University
†Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology
‡College of Computing and Software Engineering, Kennesaw State University
*kmcottrell42@tntech.edu, †brintodibyendu@gmail.com, ‡hshahria@kennesaw.edu, §arahman@tntech.edu
*† Kaitlyn Cottrell and Dibyendu Brinto Bose are joint first authors of the paper

***Abstract*—The ubiquitous usage of robots in modern society necessitates secure development of robotics systems. Practitioners who engage in robot development can benefit from a systematic study that investigates the categories of vulnerabilities that appear in robotics systems. *The goal of this paper is to help practitioners mitigate vulnerabilities in robotics systems by conducting an empirical study of vulnerabilities in robotics systems.* We conduct an empirical study where we analyze 176 robotics-related vulnerabilities collected from the Robot Vulnerability Database (RVD). Our findings show that: (i) robotics-related vulnerabilities can be classified into nine categories; (ii) memory-related vulnerabilities are the most frequent category, (iii) 92.6% of the reported vulnerabilities are software-related, and (iv) software components in robotics systems include more critical vulnerabilities compared to that of hardware components. Based on our findings, we provide a list of development activities that can be used to mitigate vulnerabilities for robotics systems.**

***Index Terms*—empirical study, mining software repositories, qualitative analysis, robotics, vulnerabilities**

I. Introduction

A malfunctioning robot was responsible for injuring an employee at Omnipure Filter Company, Inc. [22]. The malfunctioning robot pinned the employee against a 400-degree Fahrenheit mold that caused burns to the employee's arm, face, and upper torso [22]. If robots have security vulnerabilities, then malicious users can conduct attacks to deter robots from their expected behaviors and cause serious consequences, similar to the burn injury mentioned above.

Now-a-days robots are used in diverse set of domains, for example, eldercare robots, such as CareO-Bots [1] are used to perform household tasks and provide mobility assistance [6]. As another example, companies, such as Amazon uses manufacturing robots to accomplish manufacturing labor activities, such as welding and assembling equipment [6]. According to the International Data Corporation, spending on robotics is expected to reach USD 241.4 billion by the end of 2023 [26].

Despite widespread usage of robots in multiple sectors of our lives, security vulnerabilities remain a concern. Malicious users can exploit security vulnerabilities in hardware and software components of robotics systems to conduct security attacks and cause malfunction, i.e., deviate robots from their expected behaviors [6]. Security attacks on robots can have serious consequences in multiple sectors of our daily lives, which include but are not limited to (i) cause bottlenecks and shutdowns in the assembly line, (ii) cause disruption in the food supply chain, (iii) provide incorrect treatment for patients, and (iv) conduct unwanted military attacks that can injure or kill thousands of civilians and military personnel [6].

Researchers [13] have observed a lack of awareness amongst practitioners related to security issues that can exist in robotics systems. Researchers [13] conducted a survey with 50 practitioners, and reported that 76% of the surveyed practitioners never applied security testing activities on their robotics systems. Researchers [13] further reported that 50% of the survey respondents did not believe that security attacks on robotics systems are realistically possible.

The above-mentioned evidence highlights lack of security awareness amongst practitioners who are engaged in robotics systems development. Unaware practitioners can be a weak link in the entire robotics development process, who can potentially leave security vulnerabilities in robotics systems. One strategy to inform practitioners is to systematically study the reported vulnerabilities for robotics systems. A categorization of vulnerabilities related to robotics can be helpful in this regard, as vulnerability categorization can reveal the vulnerability categories that occur for robotics systems, and the actions that practitioners need to take in order to mitigate different categories of vulnerabilities.

In other domains, such as Android, researchers [17] have documented the importance of analyzing and categorizing reported vulnerabilities. Vasquez et al. [17] stated categorizing vulnerabilities can help Android practitioners "*in focusing their verification and validation activities*". Our hypothesis is that through systematic categorization of robotics-related vulnerabilities, we can identify activities to mitigate vulnerabilities in robotics systems.

The goal of this paper is to help practitioners mitigate vulnerabilities in robotics systems by conducting an empirical study of vulnerabilities in robotics systems.

We answer the following research questions:

- **RQ1 [*Frequency*]:** How frequent are vulnerabilities in robotics systems?
- **RQ2 [*Component*]:** In what types of robotics components are vulnerabilities located?
- **RQ3 [*Categorization*]:** What categories of vulnerabilities exist for robotics systems?

[1]https://www.care-o-bot.de/en/care-o-bot-4.html

- **RQ4 [*Severity*]:** What is the reported severity for vulnerabilities in robotics systems?

We conduct an empirical study using vulnerability reports included in the Robotics Vulnerability Database (RVD) [28] to characterize vulnerabilities in robotics. We apply a qualitative analysis called open coding [23] to determine the software and hardware types in which vulnerabilities appear. Next, we apply open coding to categorize robotics vulnerabilities. An overview of our methodology is presented in Figure 1.

Our contributions are listed as follows:

- A list of vulnerability categories that appear for robotics systems;
- An empirical analysis of vulnerabilities that occur in hardware and software components of robotics systems; and
- A curated dataset of vulnerabilities that map to identified categories [3].

We organize rest of the paper as follows: we provide the methodology of our empirical study in Section II. We report research results in Section III. We discuss the findings of our paper along with the limitations of our paper in Section IV. We discuss related work in Section V. Finally, we conclude the paper in Section VI.

II. Methodology

We use RVD [28] to identify vulnerabilities that appear in robots. We use RVD as it only focuses on robotics, and provides a list of vulnerabilities that occur for robotics systems. In RVD, each vulnerability report is available as a GitHub issue. The issue report includes information for a vulnerability in JSON format, where fields and sub-fields are explicitly labeled. Figure 2 provides an example of a vulnerability report indexed in RVD.

Vulnerability datasets may suffer from quality issues that can impact empirical analysis. We mitigate this limitation by conducting the following filtering criteria:

- *Criterion-1*: we filter vulnerabilities included in RVD that do not include CVEs; and
- *Criterion-2*: we remove duplicate vulnerability reports by inspecting the Common Vulnerability Enumeration (CVE) index.

We apply the above-mentioned criteria to include vulnerabilities that are confirmed by a credible source, such as the National Vulnerability Database (NVD). NVD indexes confirmed vulnerabilities using the CVE entry. Practitioners tend to mislabel non-security bugs as vulnerabilities and report duplicate vulnerabilities, which motivated us to apply CVE-based filtering similar to prior work [5]. After applying the aforementioned criteria we end up with 176 vulnerabilities. The dataset used to answer the four research questions is available online [3].

A. Methodology to Answer RQ1 (Frequency)

We answer RQ1 by reporting (i) vulnerabilities per component using Equation 1, and (ii) vulnerabilities per year using Equation 2.

$$\text{Vulnerability Per Component } (p)\% = \frac{\text{\# of vulnerabilities reported for component } p}{\text{total vulnerabilities in the dataset}} * 100\% \quad (1)$$

$$\text{Vulnerability Per Year } (x)\% = \frac{\text{\# of vulnerabilities reported in year } x}{\text{total vulnerabilities in the dataset}} * 100\% \quad (2)$$

B. Methodology to Answer RQ2 (Component)

We answer RQ2 by first identifying the types of hardware devices and software projects that exist in our vulnerability dataset. Next, we apply metrics to quantify the frequency of vulnerabilities that occur for each identified hardware device type and software project type.

To identify software project types and hardware device types by applying a qualitative analysis technique called open coding [8]. We allocate a rater, the first author of the paper, to conduct open coding. The first author is a graduate student with one year experience in software engineering and cybersecurity.

The rater derives the type of hardware devices and software projects in the following manner: *first*, the rater inspects all available fields for each vulnerability entry. *Second*, the rater identifies the name and description of the software project or hardware device, along with the title and description of the associated vulnerabilities. *Third*, the rater applies open coding on the text obtained from the previous step. The text includes description of a hardware device or a software project.

Figure 3 provides an example of applying open coding: we first extract raw text 'ABB IDAL HTTP server is a HTTP-based web server' and 'ABB WebWare Server is a web-based server' from the two software project descriptions listed under 'Description'. Next, we identify two initial categories 'HTTP-based web server' and 'Web-based server'. We merge the two initial categories into a category called 'Web Servers' as the two initial categories correspond to software projects related to web servers. We repeat the above-mentioned procedure to determine hardware device types and software project types.

Rater Verification: The process of deriving hardware device types and software project types is susceptible to bias. We mitigate this bias by allocating another rater, who is the last author of the paper. The last author separately applies closed coding [8] on the description of 50 randomly-selected vulnerability reports. The last author is an academic researcher in secure software development with 7 years of academic experience in cybersecurity. For each of the 50 vulnerability reports, the rater individually examines to which the vulnerability category the listed device in the vulnerability report maps to. We calculate the agreement between the first and the last author using Cohen's Kappa [7]. For the 50 vulnerability reports the Cohen's Kappa is 0.77 between the first and last author, which suggests 'substantial' agreement.

Frequency of Vulnerabilities: We quantify the frequency of vulnerabilities for hardware device types and software project types respectively, using Equation 3 and 4. Equations 3 and 4

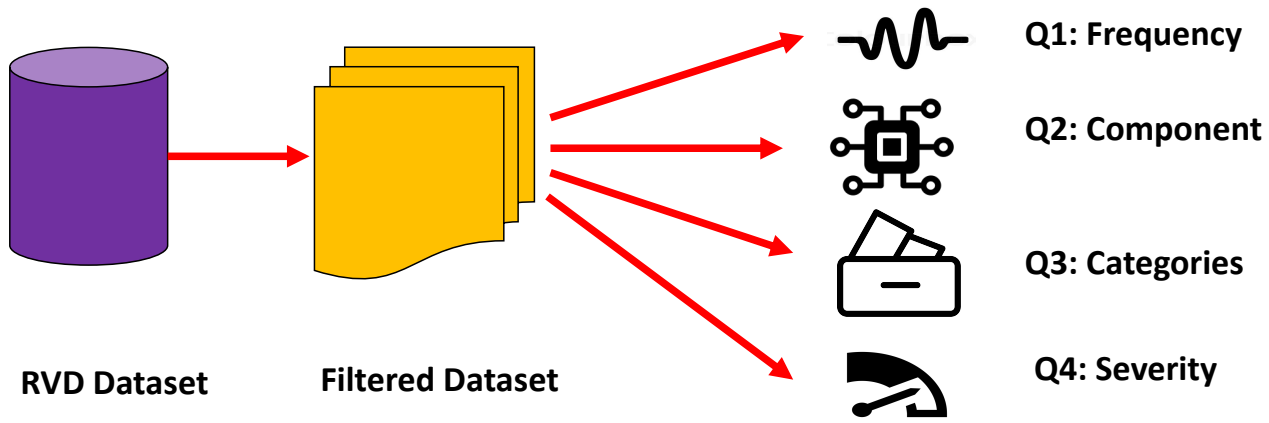

Fig. 1: An overview of the methodology used to answer the four research questions.


```
"id": 3328,
"title": "RVD#3328: Privilege Escalation and DoS on
    ↪ several Mitsubishi products.",
"type": "Vulnerability",
"description": "A permissions issue in GX Works 2 &
    ↪ 3 and MELSOFT could allow an attacker to
    ↪ escalate privilege and execute malicious
    ↪ programs, which could cause a
    ↪ denial-of-service condition, and allow
    ↪ information to be disclosed, tampered with,
    ↪ and/or destroyed.",
"cve": "CVE-2020-14496",
"keywords": [
    "Mitsubishi, DoS, Privilege escalation"
],
"system": [
    "GX Works2, GX Works3, MELSOFT"
],
"vendor": "Mitsubishi Electric Corporation",
"severity": {
    "severity-description": "high",
},
```


Fig. 2: An example vulnerability report indexed in RVD.

respectively, calculates two metrics 'Vulnerability Proportion per Hardware Type' and 'Vulnerability Proportion per Software Type'.

$$\text{Vulnerability Proportion per Hardware Type } (h)\% = \frac{\text{\# of vulnerabilities recorded for hardware type } h}{\text{total hardware-related vulnerabilities in dataset}} * 100\% \quad (3)$$

$$\text{Vulnerability Proportion per Software Type } (s)\% = \frac{\text{\# of vulnerabilities recorded for software type } s}{\text{total software-related vulnerabilities in dataset}} * 100\% \quad (4)$$

C. Methodology to Answer RQ3 (Categorization)

We identify vulnerability categories using open coding [8], similar to that of RQ2. The process of deriving categories is susceptible to bias. We mitigate this bias by allocating another rater, who is the second author of the paper. The second author separately applies closed coding [8] on the description of 50 randomly-selected vulnerability reports. The second author is a fourth year undergraduate student with 2 years of academic experience in cybersecurity. For each of the 50 vulnerability reports, the rater individually examines if the vulnerability maps to any of the categories identified by the first author. For the 50 vulnerabilities the Cohen's Kappa is 0.63 between the first and second author, which suggests 'moderate' agreement, according to Landis and Koch [16]. Reasons for disagreements are attributed to the second author's lack of familiarity with the topic. In our categorization one vulnerability can belong to multiple categories.

Category Frequency: We report the frequency of the identified vulnerability categories using two metrics: (i) Vulnerability per Category using Equation 5, and (ii) Vulnerability Category Per Component using Equation 6. In Equation 6 q corresponds to two components: hardware and software.

$$\text{Vulnerability Proportion per Category } (p)\ \% = \frac{\text{\# of vulnerabilities marked as category } p}{\text{total vulnerabilities in the dataset}} * 100\% \quad (5)$$

$$\text{Vulnerability Category per Component } (p, q)\% = \frac{\text{\# of vulnerabilities in component } q \text{ labeled as category } p}{\text{total vulnerabilities in component } q} * 100\% \quad (6)$$

D. Methodology to Answer RQ4 (Severity)

In our empirical study we also investigate the severity of the vulnerabilities as reported in the RVD. Such investigation can give researchers and practitioners an understanding about which platform, hardware or software is likely to contain critical or high severity vulnerabilities. We use the 'severity description' field that is available as part of each RVD entry.

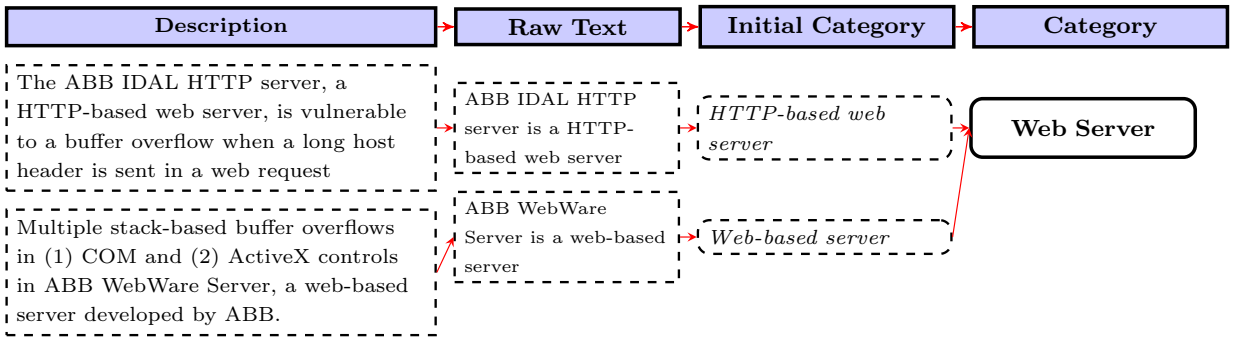

Fig. 3: An example to demonstrate the process of deriving software project types using open coding.

RVD reports four levels of severity for each vulnerability: 'critical', 'high', 'medium', and 'low'. We answer RQ4 using two metrics, 'Severity/Platform' and 'Severity/Category', which we compute respectively, using Equations 7 and 8. In Equations 7 and 8, s corresponds to four severity levels, namely, 'critical', 'high', 'medium', and 'low'. In Equation 7 p corresponds to two platforms: hardware and software. In Equation 8 c corresponds to the category levels identified in Section 5.

$$\text{Severity/Platform}(s,p)\ \% = \frac{\text{\# of vulnerabilities for platform } p\text{, marked as severity } s}{\text{total vulnerabilities in platform } p} * 100\% \tag{7}$$

$$\text{Severity/Category}(s,c)\ \% = \frac{\text{\# of vulnerabilities marked as severity } s \text{ that belong to category } c}{\text{total vulnerabilities that belong to category } c} * 100\% \tag{8}$$

III. Results

On December 25, 2020 we download 385 vulnerability entries from the RVD. Of the 385 vulnerability entries, 58 entries had no CVEs, and 151 entries were duplicates of others. After applying our filtering criteria, we obtain 176 vulnerabilities that we use in our empirical study.

A. Answer to RQ1 (Frequency)

We report the vulnerability per component values in Table I. We observe 92.6% of studied 176 vulnerabilities to occur in software components.

TABLE I: Answer to RQ1: Vulnerabilities per Component

Type	Count	Vuln. Per Component (%)
Hardware	13	7.4
Software	163	92.6

The vulnerability per year values is presented in Table II. We observe the highest count of vulnerabilities in 2020, followed by 2016 and 2017. Results from Table II show that after vulnerabilities per year is highest for 2020. We also observe a striking increase from year 2014 to 2015. One possible explanation can be related to the reporting of vulnerabilities: perhaps since 2014, more vulnerabilities were reported and indexed in the RVD.

TABLE II: Answer to RQ1: Vulnerability per Year

Type	Vuln. Per Year (%)
2009	2.3
2010	0.5
2011	0.5
2012	1.1
2013	0.5
2014	1.1
2015	10.7
2016	18.6
2017	18.6
2018	13.4
2019	16.3
2020	22.2

B. Answer to RQ2 (Component)

We report the type of hardware devices and software projects in which vulnerabilities appear as follows:

Hardware device types: Altogether 13 vulnerabilities appeared for the following hardware device types:

Monitoring Systems: A monitoring system is a device that logs and analyzes data packets coming from devices used to implement a robot. Vulnerabilities are documented for monitoring devices, such as 'ABB VSN300 WiFi Logger'.

Ethernet Devices: An Ethernet device is used to connect with Ethernet-based local and remote networks. According to a recent report [14], Ethernet is becoming popular to connect robots in industry settings. However, we observe security vulnerabilities to exist for Ethernet devices, such as in an Ethernet device called 'M2M ETHERNET'.

Multiplexer Devices: A multiplexer is used to select between multiple analog or digital input signals. In robotics systems multiplexers are used to process signals from a variety of sensors. A multiplexer device called 'ABB FOX515T' included a vulnerability indexed in RVD.

Next Unit Computing Devices: A next unit computing (NUC) device is a hardware device that is partially assembled compared to that of a regular retail computer. Unlike regular retail computers, NUC devices allow more customization and lower costs. NUC devices are used in robots that uses cameras, e.g., Intel's retail robot, which uses multiple cameras to interpret information in its environment [18]. Intel's NUC is an example NUC device that included a vulnerability.

Human Machine Interfaces: A human-machine interface (HMI) is a hardware device that provides a graphic terminal to enable a robot operator to control, monitor, and collect data from the robotics system [2]. HMIs can be useful to track, troubleshoot, and correct errors in the robot's movements. The 'ABB CP635 HMI' is an example HMI device that included a vulnerability.

Actuators: An actuator is a hardware device that is used to move a robot's joint [3]. Actuators are typical in robots that have multiple joints that are used in horizontal and vertical movements. An example robot actuator is 'Dynamixel' [4] .

We report the vulnerability proportion per hardware device type values for each device type in Table III. One vulnerability can appear for more than one hardware type.

TABLE III: Vulnerability Proportion for Six Hardware Device Types

Type	Prop. per Hardware Type (%)
Monitoring Systems	53.8
Ethernet Devices	23.1
Multiplexer	15.4
Human Machine Interface	15.4
NUC Devices	7.7
Actuators	7.7

Software projects: Altogether 163 vulnerabilities appeared for the following software projects:

Web Server: This category includes software projects that serve as web servers. Web servers are used to store data collected from robots, as well as use to control robots over the HTTP protocol. Web servers are also used in cloud robotics [27] where cloud resources are used to control robots.

Management Software: This category includes software projects used to manage, train, and simulate robots. Furthermore, this category of software projects can be used to collect data in order to gain further insights. Examples of management software includes 'RobotStudio' provided by ABB [5].

Operating System Software: This category includes software projects that are used as operating systems dedicated for robotics, such as the Robot Operating System (ROS) [20]. ROS is perceived as a medium for diverse groups to collaborate to build upon each other's work in the robotics domain. Operating system software for robotics includes utilities and libraries to construct accurate robot behavior.

Universal Robots Controller: Universal Robots Controller is a software framework the enables practitioners to interact with robot arms developed by a Denmark-based company called Universal Robots. The purpose of this framework is to provide programming utilities so that the robot arms provided by Universal Robots are easily programmable. The framework provides a Python library called 'urx' [25] to program robots built by Universal Robots and other command line utilities using which practitioners can run BASH-like commands. Universal Robots Controller is different from the other categories, as this particular category is only applicable for robot arms developed by Universal Robots.

We report vulnerability proportion per software type values for each of the four software projects in Table IV. Amongst the four software project types, Universal Robots Controller include the highest number of reported vulnerabilities that appear for software components.

TABLE IV: Vulnerability Proportion for Four Software Project Types

Type	Prop. per Software Type (%)
Universal Robots Controller	62.4
Management Software	25.9
Web Server	6.9
Operating System Software	4.8

C. Answer to RQ3 (Categorization)

We identify nine vulnerability categories that appear in robotics systems. We provide descriptions and examples for each category below:

Authorization/Authentication: This category includes vulnerabilities that are related with incorrect authentication and unauthorized privileges. If a malicious user exploits vulnerabilities that belong to this category, then the exploitation will result in unauthorized access.

Example: `CVE-2017-7916` is an example of an authorization/authentication vulnerability. The vulnerability was discovered in a web application called 'ABB VSN300'. According to the description, the application does not properly restrict

[2]https://www.robots.com/faq/what-is-a-hmi

[3]https://robotacademy.net.au/lesson/actuators/

[4]https://robots.ros.org/dynamixel/

[5]https://new.abb.com/products/robotics/robotstudio

privileges of the guest account [1]. As a result, a malicious user might be able to access configuration information, which should be restricted.

Cryptography-related Vulnerabilities: This category includes vulnerabilities that occur due to inadequate encryption of sensitive information used within robotics systems. Existence of cryptography-related vulnerabilities can enable malicious users gain unwanted access.

Example: `CVE-2020-10267` [1] is an example vulnerability related to cryptography. The vulnerability was discovered in multiple versions of software used in Universal Robots Control. According to the vulnerability description, the software does not encrypt artifacts that were installed from a third-party software project called 'URCaps'. The consequence of this vulnerability is that upon exploitation malicious users with access to the robot or the robot network will be able to access all available intellectual property [1].

Denial of Service (DOS): This category includes vulnerabilities that can result into denial of service attacks. Denial of service is an attack where a malicious user seeks to make a computing machine unavailable to its intended users by temporarily or indefinitely disrupting services of a host connected to the Internet. Vulnerabilities that cause DOS attacks can degrade the service quality, cause response delays, excessive losses, and service interruptions [6].

Example: One example DOS-related vulnerability is `CVE-2015-7697` [1]. The vulnerability occurs for 'UnZip' that is used in 'urx', a Python library to interact with robots created by Universal Robots [25]. The vulnerability was discovered by creating an empty 'bzip2' data file. A malicious user can exploit the vulnerability to cause a DOS attack by creating an infinite loop with an empty bzip2 data in a ZIP file.

Dependency-related Vulnerabilities: This category includes vulnerabilities that occur when software packages with vulnerabilities are used as dependencies. Outdated software versions can have known vulnerabilities, and usage of outdated software can propagate known vulnerabilities into robotics systems.

Example: `CVE-2016-5699` is an example of a dependency-related vulnerability, where a vulnerable version of Python was used in URx [1]. Any Python library older than 3.4.4 includes a Carriage Return Line Feed (CRLF) vulnerability, which enables malicious users to inject arbitrary HTTP headers. The vulnerability allows malicious users to inject arbitrary HTTP headers via CRLF sequences in a URL.

Directory Traversal: This category includes vulnerabilities that occur for allowing incorrect directory traversals. Incorrect directory traversal focuses on gaining access to the file system by injecting malicious directory paths so that sensitive directories and files, such as `/etc/passwd` can be accessed.

Example: `CVE-2016-6321` [1] is an example of a vulnerability related with incorrect directory traversal. The vulnerability was located in the 'GNU tar' application that affected urx. GNU tar does not properly handle member names containing '..', which can allow a malicious user to bypass a protection mechanism, and write to arbitrary files.

Hard-coded Secret: This category includes vulnerabilities that occur when sensitive information, such as hard-coded usernames, and passwords are included in robotics-related software artifacts. Hard-coded secrets is detrimental to the security of a system. According to Common Weakness Enumeration (CWE), "*Username and password information should not be included in a configuration file or a properties file in plain text as this will allow anyone who can read the file access to the resource*" [9]. Similar to infrastructure as code [21], we notice hard-coded secrets being exposed in robotics software artifacts.

Example: `CVE-2020-10270` [1] is an example of a vulnerability where sensitive information were hard-coded into a software artifact. This vulnerability can allow malicious users to take control of the robot remotely and use the default user interfaces provided by Mobile Industrial Robots (MiR).

Input Sanitization: This category includes vulnerabilities that occur when input is not adequately sanitized. Upon exploitation this category of vulnerabilities can result in execution of arbitrary code if a malicious file is created.

Example: `CVE-2016-1248` [1] is an example vulnerability related with input sanitization. The vulnerability could lead to execution of arbitrary code when a file with a crafted modeline is opened.

Insecure Default Settings: This category includes vulnerabilities that occur when default configuration settings of the robotic system is predictable and can be used to conduct attacks against the robotics system. Cybersecurity experts advocate for modifications of default configurations of the system so that attack surface of a system reduced.

Example: `CVE-2020-10274` [1] is an example of a vulnerability where default settings are used to configure a robotics control dashboard. The control dashboard uses access tokens that are included in a publicly available database of default credentials. This vulnerability can be leveraged by malicious users to exfiltrate sensitive data, such as indoor map images.

Memory-related Vulnerabilities: This category includes vulnerabilities that occur when performing operations related to memory. Exploitation of vulnerabilities that belong to this category can lead to overflow errors, such as buffer overflow and heap overflow.

Example: An example memory-related vulnerability is `CVE-2017-1000409` [1], which upon exploitation can create a buffer overflow in 'glibc 2.5'. 'glibc' is the abbreviated version of GNU C Library that is extensively used in operating system kernels, such as FreeBSD, and in drivers for hardware, such as MIPS and PowerPC.

Category Frequency: In our categorization, one vulnerability can belong to multiple categories. We report the vulnerability per category values in Table V. We report the vulnerability category per component values in Table VI. From Table V we observe memory-related vulnerabilities to be the most frequently occurring vulnerability category in our set of 176

[6]https://owasp.org/www-community/attacks/Denial_of_Service

vulnerabilities. According to Table VI, input sanitization and memory-related vulnerabilities to be the most frequent categories respectively, for hardware components and software components.

TABLE V: Vulnerability Proportion per Category

Type	Vuln. Proportion per Category (%)
Memory	30.7
Input Sanitization	25.5
Authorization/Authentication	23.2
Denial of Service	18.1
Cryptography	6.8
Dependency	6.8
Insecure Default Settings	6.8
Directory Traversal	3.5
Hard-coded Secret	2.3

TABLE VI: Vulnerability Category per Component Values for Hardware and Software

Type	Hardware (%)	Software (%)
Authorization/Authentication	33.3	22.5
Cryptography	5.6	7.3
Denial of Service	5.6	19.6
Dependency	0.0	6.0
Directory Traversal	3.4	2.5
Hard-coded Secret	0.0	2.4
Input Sanitization	38.9	24.1
Insecure Default Settings	0.0	7.3
Memory	22.2	32.4

D. Answer to RQ4 (Severity)

We answer RQ4 by reporting the values for Severity/Platform and Severity/Category metric. We report the values for the Severity/Platform metric in Table VII. For example, we observe 12.5% of vulnerabilities reported for hardware to be critical vulnerabilities. In the case of hardware, 12.5% and 6.2% of vulnerabilities are respectively, labeled as critical and high. For software components, 21.2% and 36.1% of vulnerabilities are respectively, labeled as critical and high. We observe critical vulnerabilities to occur more frequently in software components compared to that of hardware components.

TABLE VII: Vulnerability Severity for Hardware and Software

Type	Hardware (%)	Software (%)
Critical	12.5	21.2
High	6.2	36.1
Medium	0.0	18.7
Low	81.3	24.0

We report the values for the Severity/Category metric in Table VIII. For example, we observe 12.0% of vulnerabilities that are categorized as Authorization/Authentication to be critical vulnerabilities. In the case of dependency, directory traversal, hard-coded secret, and memory, 25% or more of the vulnerabilities are categorized as critical vulnerabilities. For hard-coded secret, 100% of all vulnerabilities are labeled as critical.

IV. Discussion

In this section, we *first* synthesize our findings as lessons. *Second*, we discuss the implications of our findings:

A. Lessons Learned

We summarize the lessons learned from our empirical study as follows:

1) Lesson#1: New Ecosystem, Similar Vulnerabilities: Compared to other domains, robotics is a relatively new ecosystem that is getting increasing attention. Despite becoming a relatively new domain, we observe the vulnerability categories to be similar to that of other existing domains, such as web software applications. Vulnerability categories, such as dependency-related vulnerabilities, input sanitization, and memory-related vulnerabilities also appear for web application software. Our hypothesis is that the commonality of vulnerability categories between robotics and other domains can provide researchers clues on how to bring in cybersecurity knowledge, vulnerability detection and repair techniques from other domains, for example, from the domain of web application software into robotics.

2) Lesson#2: Not Only Software or Hardware: Answer to RQ2 provide evidence that vulnerabilities are not limited to only software or only hardware. We observe that vulnerabilities can appear in multiple types of hardware devices, such as actuators, multiplexers, and NUCs. Furthermore, we observe that robotics vulnerabilities can appear for multiple software project types.

Our findings shed light on the complexity of robotics system development. We observe that the entire development toolchain for a robot requires a diverse set of devices. Practitioners can take these observations into account in order to devise cybersecurity best practices for these devices. The need of such guidelines is becoming increasingly important, as robots become more ubiquitous and play crucial roles in diverse domains, such as health care and supply chain.

B. Implications

We discuss the implications of our paper below:

1) Implications Related to Vulnerability Mitigation: Based on our research we advocate practitioners to apply the following development activities to mitigate the nine categories of vulnerabilities:

Mitigating Authorization/Authentication Vulnerabilities via Threat Modeling: Threat modeling is the technique of identifying and prioritizing security threats for any system [24]. Table V shows the presence of vulnerabilities related to authentication/authorization for robotics systems. This category of vulnerabilities can be mitigated through adequate threat modeling, as threat modeling can enable practitioners to gain understanding on how authorization or authentication issues might occur in robotics systems.

Mitigating Memory and Input Sanitization Vulnerabilities via Vulnerability Scanning: We advocate practitioners to regularly scan their robotics source code repository to identify

TABLE VIII: Severity of Vulnerability Categories

Type	Critical (%)	High (%)	Medium (%)	Low (%)
Authorization/Authentication	12.0	17.0	63.6	7.4
Cryptography	20.1	33.4	13.3	33.2
Denial of Service	19.3	32.2	38.8	9.7
Dependency	25.0	41.7	25.0	8.3
Directory Traversal	40.0	0.0	0.0	60.0
Hard-coded Secret	100.0	0.0	0.0	0.0
Input Sanitization	13.6	36.3	11.3	38.8
Insecure Default Settings	0.0	100.0	0.0	0.0
Memory	25.9	42.4	16.6	15.1

vulnerabilities. Existing tools, such as `CPPCheck` [7] and `PySA` [8] exists respectively, for C/C++ and Python, which can identify certain vulnerabilities in source code. If practitioners are using third party libraries as dependencies for their robotics system implementation, then practitioners should scan their dependencies. Dependency scanning is becoming mainstream, for example, GitHub provides an utility called `Dependabot` [9] that automatically notifies repository maintainers if vulnerable dependencies exist in their source code. Commercial tools, such as `Snyk` [10] can also be used to identify vulnerable software dependencies.

Mitigating Cryptography Vulnerabilities: Security experts have suggested the following activities that practitioners in the robotics domain can follow: automated key rotation, robust user authentication, and logging to demonstrate compliance [11]. Reinventing the wheel, i.e., development of cryptography algorithms from scratch is also discouraged [12].

Mitigating Denial of Service Vulnerabilities: We echo views expressed by cybersecurity practitioners [13], and advocate practitioners to apply the following activities to mitigate denial of service vulnerabilities: monitoring and analysis of robot traffic patterns, and application of configuration-related best practices.

Mitigating Dependency Vulnerabilities: Following recommendations from cybersecurity experts [14] we advocate the following activities to mitigate dependency vulnerability-related dependencies: scanning and updating software dependencies, as well as testing of third-party applications and libraries that are used in the robotics system.

Mitigating Vulnerabilities Related to Directory Traversal: Similar to other domains [15], validating user inputs and verifying canonical paths can help practitioners mitigate vulnerabilities related to directory traversal for robotics systems.

Mitigating Hard-coded Secrets via CredScan: We advocate practitioners to apply tools, such as CredScan [16] so that practitioners can automatically identify hard-coded secrets in robotics-related software artifacts. Practitioners can also apply code review practices so that they can identify hard-coded secrets before deploying the software.

Mitigating Insecure Default Settings: We advocate practitioners to always change default configuration settings of software so that configuration settings of the software is not predictable and the attack surface for the software is reduced.

2) Fuzzing-related Implications: Practitioners can use fuzzing to detect and repair vulnerabilities pro-actively while developing robotics systems. Fuzzing is a testing technique, which generates erroneous and random input to a software so that the software of interest can be monitored for exceptions, such as crashes [2]. Fuzzing can reveal latent memory-related vulnerabilities by generating unexpected input to the programs of interest.

3) Implications for Toolsmiths: Existence techniques might be helpful to repair robotics vulnerabilities. Memory-related vulnerabilities, such as buffer overflow vulnerabilities can be fixed using techniques, such as `CodePhage`, `PASAN`, and `AutoPAG`, as all of these techniques repair buffer overflow vulnerabilities [12]. Vulnerabilities related to memory leaks can be repaired using `LeakFix`, whereas, vulnerabilities related to input sanitization can be repaired using `SemRep` [12]. If source code of robotics systems is hosted on version control systems, then dependency-related vulnerabilities can be detected using automated pull requests. Application of all above-mentioned techniques for robotics systems is challenging and under-explored. Toolsmiths can build upon existing work to tailor security tools that can easily integrated into the workflow of robotics systems development.

C. Threats to Validity

Conclusion Validity: We acknowledge that our empirical study is limited to the set of vulnerabilities that we mined from RVD. We may have missed vulnerabilities that are not available in RVD. Our vulnerability categorization is susceptible to rater bias, which we mitigate using two raters. Furthermore, as part of empirical study we analyze 176 vulnerabilities downloaded on December 2020, and therefore, we may have not included vulnerabilities that are published after December 2020 in RVD.

[7] http://cppcheck.sourceforge.net/

[8] https://engineering.fb.com/2020/08/07/security/pysa/

[9] https://dependabot.com/

[10] https://snyk.io/

[11] https://www.cryptomathic.com/news-events/blog/cryptographic-key-management-the-risks-and-mitigations

[12] https://securitytoday.com/Articles/2019/08/01/Dont-Reinvent-the-Wheel.aspx

[13] https://blog.eccouncil.org/types-of-ddos-attacks-and-their-prevention-and-mitigation-strategy/

[14] https://itnext.io/mitigate-vulnerabilities-in-your-open-source-dependencies-3a31c1fce9b9

[15] https://portswigger.net/web-security/file-path-traversal

[16] https://secdevtools.azurewebsites.net/helpcredscan.html

External Validity: Our paper suffers from external validity as our empirical results may not hold for another set of vulnerabilities found in robotics systems.

V. Related Work

Our paper is related with prior research that has investigated safety and reliability issues for robotics systems. Khadidos et al. [15] used simulation to propose a detection model that detects exogenous failures in robotics. Clark et al. [6] proposed a set of hypothetical cybersecurity attacks that are targeted for robotics systems. Beltrame et al. [4] applied pattern traversal flow analysis to automatically identify is a robot is violating safety policies. Quarta et al. [19] proposed a novel attack model for industry robots an attacker model, and showed how an attacker can compromise a robot controller. Dieber et al. [11] constructed and evaluated a novel technique to secure all communication channels for ROS. DeMarinis et al. [10] also studied ROS and provided a list of best practices to secure ROS at the network layer to provide a stopgap solution for critical security issues. We observe a lack of empirical research related to robotics vulnerabilities, which we address in our paper.

VI. Conclusion

Robotics systems are becoming common place in a diverse set of domains, such as households, manufacturing industry, and health care. Ubiquitous usage of robots necessitates them to be securely developed so that vulnerabilities are mitigated. We have systematically analyzed 176 vulnerabilities reported for robotics systems to provide understanding on what vulnerabilities appear. We have identified nine vulnerability categories amongst which memory-related vulnerabilities are most frequent. Furthermore, we observe 92.6% of the reported vulnerabilities are software-related, and software components to include more critical vulnerabilities compared to that of hardware components. We have provided guidelines on how our analysis can inform practitioners to take actions for vulnerability mitigation. We hope our paper will facilitate secure development of robotics systems.

VII. Acknowledgment

We thank the PASER group at Tennessee Technological University (TTU) for their valuable feedback. This research was partially funded by the National Science Foundation (NSF) award # 2026869 and the Cybersecurity Research, Education, and Outreach Center (CEROC) at TTU.

References

[1] aliasrobotics/RVD. Robot vulnerability database (rvd). https://github.com/aliasrobotics/RVD, 2020. [Online; accessed 04-Nov-2020].

[2] Paul Ammann and Jeff Offutt. *Introduction to software testing*. Cambridge University Press, 2016.

[3] Anonymous Authors. Dataset for paper. https://figshare.com/s/1289b19dff6054ca09c3, 2020. [Online; accessed 05-Nov-2020].

[4] Giovanni Beltrame, Ettore Merlo, Jacopo Panerati, and Carlo Pinciroli. Engineering safety in swarm robotics. In *Proceedings of the 1st International Workshop on Robotics Software Engineering*, RoSE '18, page 36–39, New York, NY, USA, 2018. Association for Computing Machinery.

[5] Farzana Ahamed Bhuiyan, Akond Rahman, and Patrick Morrison. Vulnerability discovery strategies used in software projects. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering Workshops*, ASE '20, page 13–18, New York, NY, USA, 2020. Association for Computing Machinery.

[6] G. W. Clark, M. V. Doran, and T. R. Andel. Cybersecurity issues in robotics. In *2017 IEEE Conference on Cognitive and Computational Aspects of Situation Management (CogSIMA)*, pages 1–5, 2017.

[7] Jacob Cohen. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20(1):37–46, 1960.

[8] Benjamin F Crabtree and William L Miller. *Doing qualitative research*. SAGE Publications, 1999.

[9] CWE. Cwe-798: Use of hard-coded credentials. https://cwe.mitre.org/data/definitions/798.html, 2020. [Online; accessed 19-August-2020].

[10] N. DeMarinis, S. Tellex, V. P. Kemerlis, G. Konidaris, and R. Fonseca. Scanning the internet for ros: A view of security in robotics research. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8514–8521, 2019.

[11] Bernhard Dieber, Benjamin Breiling, Sebastian Taurer, Severin Kacianka, Stefan Rass, and Peter Schartner. Security for the robot operating system. *Robotics and Autonomous Systems*, 98:192 – 203, 2017.

[12] Luca Gazzola, Daniela Micucci, and Leonardo Mariani. Automatic software repair: A survey. *IEEE Transactions on Software Engineering*, 45(1):34–67, 2017.

[13] Carlos Gonzalez. Fear the hacker! robot security is a growing threat. https://www.machinedesign.com/mechanical-motion-systems/article/21835782/fear-the-hacker-robot-security-is-a-growing-threat, 2017. [Online; accessed 02-Nov-2020].

[14] Hirschmann. Ethernet for machines and robots. https://www.digikey.com/Site/Global/Layouts/DownloadPdf.ashx?pdfUrl=BA9FDC4CCD394DAE87908341BC3EEB5E, 2020. [Online; accessed 04-Nov-2020].

[15] Adil Khadidos, Richard M. Crowder, and Paul H. Chappell. Exogenous fault detection and recovery for swarm robotics. *IFAC-PapersOnLine*, 48(3):2405 – 2410, 2015. 15th IFAC Symposium onInformation Control Problems inManufacturing.

[16] J. Richard Landis and Gary G. Koch. The measurement of observer agreement for categorical data. *Biometrics*, 33(1):159–174, 1977.

[17] Mario Linares-Vásquez, Gabriele Bavota, and Camilo Escobar-Velasquez. An empirical study on android-related vulnerabilities. In *Proceedings of the 14th International Conference on Mining Software Repositories*, MSR '17, pages 2–13, Piscataway, NJ, USA, 2017. IEEE Press.

[18] Martha DeGrasse. Intel demos retail robot. https://enterpriseiotinsights.com/20170119/channels/news/intel-retail-robot, 2017. [Online; accessed 10-Nov-2020].

[19] D. Quarta, M. Pogliani, M. Polino, F. Maggi, A. M. Zanchettin, and S. Zanero. An experimental security analysis of an industrial robot controller. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 268–286, 2017.

[20] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[21] Akond Rahman, Md. Rayhanur Rahman, Chris Parnin, and Laurie Williams. Security smells in ansible and chef scripts: A replication study. *ACM Trans. Softw. Eng. Methodol.*, 2020. To appear. pre-print: https://arxiv.org/pdf/1907.07159.pdf.

[22] Occupational Safety and Health Administration. Accident: 113594.015 - employee is pinned against heated mold by robotic arm and su. https://www.osha.gov/pls/imis/accidentsearch.accident_detail?id=113594.015, 2019. [Online; accessed 01-Nov-2020].

[23] Johnny Saldana. *The coding manual for qualitative researchers*. Sage, 2015.

[24] Adam Shostack. *Threat modeling: Designing for security*. John Wiley & Sons, 2014.

[25] SintefManufacturing/python-urx. urx. https://github.com/SintefManufacturing/python-urx, 2020. [Online; accessed 10-Nov-2020].

[26] Statista. Global spending on robotics and drones in 2020 and 2023. https://www.statista.com/statistics/441948/forecast-for-robotic-market-spending-worldwide/, 2021. [Online; accessed 11-Feb-2021].

[27] R. Toris, J. Kammerl, D. V. Lu, J. Lee, O. C. Jenkins, S. Osentoski, M. Wills, and S. Chernova. Robot web tools: Efficient messaging

for cloud robotics. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4530–4537, 2015.

[28] Victor Mayoral Vilches, Lander Usategui San Juan, Bernhard Dieber, Unai Ayucar Carbajo, and Endika Gil-Uriarte. Introducing the robot vulnerability database (rvd). *arXiv preprint arXiv:1912.11299*, 2019.