A Large-Scale Empirical Study on the Occurrence of Improperly Secured Application Programming Interfaces*

Craig Opie[1] and Hangbo Zhang[2]

Index Terms— **Cybersecurity, Application Programming Interfaces, Vulnerability, Bug Bounty Enumeration, Severity ranking**

Abstract— **The intent of this paper is to conduct a large-scale empirical study on bug bounty reports related to API vulnerabilities and weaknesses. We aim to identify common critical and important API vulnerabilities discovered through the bug bounty process, using Microsoft's security update severity rating system. Additionally, we have defined a method to determine the severity ranking of a vulnerability. Our findings are cross-validated with OWASP Top 10 to determine the change in occurrence of each vulnerability. Our research can help developers determine real-world vulnerabilities and weaknesses, assign severity to common API vulnerabilities, and validate OWASP Top 10 regarding API. Furthermore, our study can provide a pathway for researchers to study and address common vulnerabilities.**

I. INTRODUCTION

Secure software is the foundation of a high-quality software system. Business models have shifted to include security in the software development life cycle due to the high cost of insecure applications and systems. Cybercrime is expected to cost the world $10.5 trillion annually by 2025 [1]. Despite the increase in cybercrime, on December 13th, 2016, Congress passed the 21st Century Cures Act [2], which requires healthcare providers to publish application programming interfaces (APIs) to provide complete access to all data formats and elements (known as "resources") of a patient's electronic health record (EHR) within one year of the date of enactment. Health Level Seven International (HL7), a healthcare standards organization, created the Fast Healthcare Interoperability and Resources (FHIR, pronounced "fire") API, which built upon earlier versions of HL7 data format standards. However, the FHIR API offered easy-to-use implementation through HTTP-based RESTful API protocol with resources represented in either RDF, XML, or JSON formatting, ultimately leading to the FHIR API protocol being widely accepted across the healthcare industry. Research published by cybersecurity analyst Alissa Knight [3] has revealed that five separate insecure implementations of the FHIR API protocol exposed over four million patient and clinician EHR resources across forty-eight mobile web clients and 25,000 healthcare providers and payers.

*This work was not supported by any organization

[1]C. Opie is a student of Computer Science, College of Natural Science, University of Hawai'i at Mānoa, POST 318, 1680 East-West Road, Honolulu, HI 96822 opieca at hawaii.edu

[2]H. Zhang is a student of Computer Science, College of Natural Science, University of Hawai'i at Mānoa, POST 318, 1680 East-West Road, Honolulu, HI 96822 hangbo at hawaii.edu

In September 2018, Facebook announced a vulnerability in its API code that resulted in the exposure of data from fifty million users and the theft of API access tokens [4]. This event followed Cambridge Analytica's abuse of Facebook's API security infrastructure, which allowed them to acquire data on eighty million Facebook users [5]. Despite the availability of patches to secure API frameworks, organizations often fail to implement them in a timely manner. For example, Equifax failed to patch a vulnerability in the Apache Struts framework used for their API, resulting in the exposure of personal information and social security numbers of over 143 million US citizens, with damages totaling more than 68 billion dollars [6].

The examples provided above are of large, known security events that have been reported. However, according to a survey conducted in 2017 on US companies with at least 250 employees or $1 million in revenue, the average-sized business manages as many as 363 public-facing APIs [7]. Furthermore, the Postman API Platform - a platform used by more than 17 million developers worldwide to build and use APIs - reports that the number of grouped API requests collected by their software has increased from less than half a million in 2016 to over 46 million as of January 2021, indicating that API usage is rapidly increasing [8].

The increasing use of cloud computing and remote work has led to a greater reliance on third-party APIs, which were originally designed for back-end applications that are not visible to the user [9]. However, the generic reuse of these APIs can lead to vulnerabilities, as identified by cybersecurity analyst Alissa Knight [3]. Average-sized businesses are more likely to use a readily available third-party solution without researching it properly. Unfortunately, common security measures such as web application firewalls and vulnerability identification software may not detect or prevent misuse of APIs as the attack vector is through the feature that makes APIs desirable. To secure APIs and protect existing networks, it is necessary to compare research publications on APIs with common vulnerabilities and weaknesses identified through bug bounty programs.

In this paper, we investigate publicly available bug bounty reports from HackerOne [10], BugCrowd [11], and Pentester.land [12] to identify common vulnerabilities and weaknesses regarding APIs. We also discuss the severity of our findings when compared to existing publications and known issues captured through common vulnerability and weakness enumeration using mitre.org. This approach is different from previous works, which focus on conducting surveys to identify demographics about API use, classifying

previous API vulnerabilities, and balancing API security with API implementation in terms of functionality, speed, and performance. We concur with recent findings by Knight [3] that existing vulnerabilities can be classified into multiple categories aligned with the Open Web Application Security Project (OWASP[1]) API Security Top 10[2] [13]. We expand existing work by providing the following contributions:

1) We perform a large scale empirical study on bug bounty reports regarding API vulnerabilities and weaknesses.
2) We identify common critical and important API vulnerabilities discovered in the bug bounty process using Microsoft's security update severity rating system.
3) We determine if mapped research publications, that use security bug reports, capture the API vulnerabilities and weaknesses identified in our large scale empirical research of bug bounty reports regarding API vulnerabilities and weaknesses.

Our research addresses the challenges faced by developers and security researchers by determining real-world vulnerabilities and weaknesses identified through relevant bug bounty reports. This helps developers assign severity to common API vulnerabilities and validates existing research and security practices related to APIs. As a result, developers and security researchers will have a deeper understanding of API security from both academia and industry, enabling them to enhance their own API security practices.

This paper is organized as follows: In Section II, we describe the goals of the study and the research questions we aim to address. In Section III, we provide background information and review related publications.

II. GOAL AND RESEARCH QUESTIONS

A. Research Questions

The objective of this research paper is to help software engineers identify common security gaps related to APIs by conducting a systematic comparison of mapped research publications that focus on security, common weakness and vulnerability enumeration, and publicly available bug bounty reports.

1) **RQ_1: What are the common vulnerability and weakness types associated with APIs identified through publicly available bug bounty reports?**
   Vulnerabilities and weaknesses are often identified and exploited in trending behavior. Researchers or analysts identify a vulnerability and then search for similar vulnerabilities and weaknesses. This information allows researchers and developers to identify when a vulnerability or weakness is discovered to be widely effective, enabling them to prioritize updates for quality.
2) **RQ_2: Which common vulnerability and weakness types associated with APIs are rated critical or important using Microsoft's security update severity rating system?**
   Common vulnerabilities and weaknesses are not typically published with severity ratings. It is usually up to developers to determine the severity of each vulnerability for each project. However, scanning common vulnerability and weakness data sets and applying severity ratings is a time-consuming task that increases overhead that must be allocated by each project. Consolidating common vulnerabilities and weaknesses associated with APIs with severity ratings will reduce the overhead allocation necessary for existing and future projects.
3) **RQ_3: Which mapped research publications that use security bug reports capture the publicly available bug bounty reports for APIs?**
   Mapped research publications that use security bug reports are used by developers and researchers to identify applicable vulnerabilities and weaknesses. However, comparing publicly available bug bounty reports for APIs with mapped research publications can validate the applicability of prior research to APIs.

III. LITERATURE REVIEW

A. Background

1) Background on Systematic Mapping Studies (SMS): Systematic mapping is a technique that is widely used in medical research and more recently in software engineering [14] [15] [16]. An SMS provides a "map" of the research field by classifying papers on the basis of relevant categories and counting the work in each of those categories. An SMS offers a summary of the research domain to support researchers in identifying topics that are well-studied and identifying gaps that need further analysis [17]. For this research, we used a peer-reviewed SMS titled "Security Bug Report Usage for Software Vulnerability Research: A Systematic Mapping Study [16]."

2) API Security: An API is an interface that defines how different software interacts. It controls the types of requests that occur between programs, how these requests are made, and the kinds of data formats that are used. It acts as the back-end framework for applications. By nature, APIs expose application logic and sensitive data, such as Personally Identifiable Information (PII), and therefore have increasingly become a target for attackers. API security refers to the practice of preventing or mitigating attacks on APIs. It is a specific security focus that addresses the unique security risks of APIs.

3) Bug Bounty Program: A bug bounty program is a deal offered by many websites, organizations, and software developers, where individuals can receive recognition and compensation for reporting bugs, particularly those related to security exploits and vulnerabilities. These programs allow developers to discover and resolve bugs before the general public is aware of them, preventing incidents of widespread

[1]OWASP is an online community that produces freely-available articles, methodologies, documentation, tools, and technologies in the field of web application security. It provides free and open resources.

[2]OWASP Top 10 is a list of the most critical web application security risks identified by the OWASP community. The list is updated every three years. The list is determined by a panel of experts from the OWASP community, who select the top 10 risks that are most prevalent, easy to exploit, and have a significant impact on the organization.

abuse and data breaches. Many organizations have implemented bug bounty programs, including Facebook, Google, Microsoft, and the Internet Bug Bounty [18] [19].

4) Exploit and Vulnerability: An exploit is a piece of software, a chunk of data, or a sequence of commands that takes advantage of a bug or vulnerability to cause unintended or unanticipated behavior to occur on computer software, hardware, or something electronic (usually computerized). It is written either by security researchers as a proof-of-concept threat or by malicious actors for use in their operations. When used, exploits allow an intruder to remotely access a network and gain elevated privileges, or move deeper into the network [20].

Vulnerabilities are flaws in a computer system that weaken the overall security of the device/system. Vulnerabilities can be weaknesses in either the hardware itself or the software that runs on the hardware. Vulnerabilities can be exploited by a threat actor, such as an attacker, to cross privilege boundaries and perform unauthorized actions within a computer system. To exploit a vulnerability, an attacker must have at least one applicable tool or technique that can connect to a system weakness. In this frame, vulnerabilities are known as the attack surface [21].

5) Microsoft Security and Severity Rating System: Microsoft Security is a built-in software for Windows OS that includes virus and threat protection, account protection, firewall and network protection, app and browser control, device security, device performance and health, family options, and automatic security updates. It continually scans for malware, viruses, and security threats [22]. Microsoft publishes a severity rating system that rates each vulnerability according to the worst theoretical outcome if the vulnerability is exploited, to help customers understand the risk associated with each vulnerability they patch. The rating system is intended to provide a broadly objective assessment of each issue, distinct from the likelihood of a vulnerability being exploited. There are four levels of the rating: critical, important, moderate, and low. Each rating has different identifiers. Critical vulnerabilities can be exploited to allow code execution without user interaction, important vulnerabilities can result in compromise of the confidentiality, integrity, or availability of data or processing resources with warnings or prompts, moderate vulnerabilities have impacts that are mitigated to a significant degree by factors such as authentication requirements or non-default configuration, and low vulnerabilities have impacts that are comprehensively mitigated by the characteristics of the affected component [23].

B. Related Work

To avoid duplicating a study that has already been conducted, we first identified related studies and reviews to determine the need for and timeliness of our study.

Farzana et al. [16] conducted a systematic review of 46 publications that used security bug reports and identified three main topics: vulnerability classification, vulnerability summarization, and vulnerability dataset construction. Their study found potential research opportunities for further development in vulnerability analysis, but did not provide a detailed description of API security threats and aspects, focusing more on software security in general.

Davis et al. [9] investigated why API insecurity is often overlooked in a zero-security environment, while Bigelow et al. [24] discussed the impact of data vulnerability from using insecure APIs in a secure environment. Farhan et al. [25] focused on API security, exploring how user and developer culture and behavior might have affected API security awareness. These studies provided insight into the reasons behind API insecurity, its effects, and advice for improvement, but did not provide empirical evidence of the cause of API insecurity or the consequences of API insecurity, or detailed identification of API vulnerabilities.

Diaz-Rojas et al. [26] reviewed literature and identified 66 threats to web APIs, 21 techniques, 11 API design patterns and 34 methods that can be applied at the design level to detect, resist, react to, or recover from the discovered threats. They also noted the relationship between the discovered threats and techniques in regards to the reported effectiveness of certain techniques and difficulty of defending against certain threats. While this study provided detailed information about web API vulnerabilities and weaknesses, it was limited to web APIs only.

Based on these findings, we conclude that to identify common security gaps related to APIs, we need to conduct our own study, using a large-scale empirical study of bug bounty reports to identify critical and important API vulnerabilities. We will also verify our findings by comparing them to current bug bounties to validate continuing vulnerabilities with APIs.

IV. METHODOLOGY

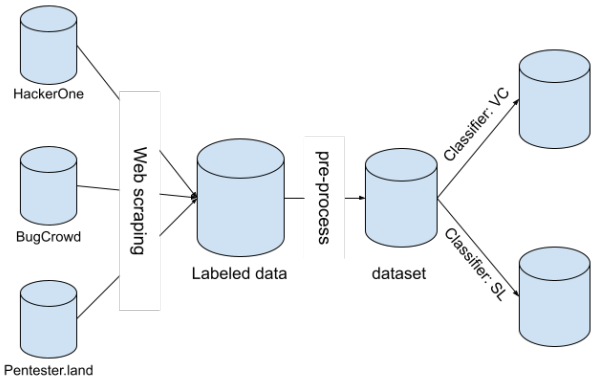

Fig. 1. Methodology Flowchart

We investigated and determined common vulnerability and weakness types associated with APIs using vulnerability databases and bug bounty platforms. Some industry leaders and government organizations publish patched security vulnerabilities which can be found in the National Vulnerability Database [27]. However, others choose to ignore reporting guidelines or fail to take action to correct the

vulnerability at all. To address this, a cybersecurity community effort to publish vulnerabilities to a public database, known as the Common Vulnerabilities and Exposures (CVE) database [28], has emerged as a means to hold companies accountable for their cybersecurity and privacy responsibilities. Bug bounty platforms offer a unique perspective on the frequency, severity, and longevity of vulnerabilities after they have been published in the NVD and CVE databases. Our process consisted of the following five steps: (i) collecting bug reports for API vulnerabilities; (ii) performing bug report quality assessment, sanitization, and classification using the OWASP guidelines [13] for API security; (iii) identifying vulnerabilities with critical or important severity ratings using the Microsoft Security Update Severity Rating System; (iv) quantifying and ranking the bugs reported after they have been published in public vulnerability databases; and (v) validating prior research SMS applicability to APIs and OWASP API security top 10.

A. Collect Bug Reports for API Vulnerabilities

We collected bug reports for API vulnerabilities from HackerOne, BugCrowd, and Pentester.land using their built-in search filters for the keyword 'API' and a self-built web scraping tool [29]. A total of 1028 reports were extracted from HackerOne, 15 reports from BugCrowd, and 85 reports from Pentester.land, resulting in a dataset of 1128 reports. The collected data was cataloged by title, report number, URL, severity rating, bounty, upvotes, CVE, and report details.

We chose to use HackerOne, BugCrowd, and Pentester.land for this report because each site offered consolidated disclosed reports that were not repeated among each other. Additionally, the reports were able to be collected via web scraping and allowed for convenient data processing within a python application [29]. HackerOne and Pentester.land also offered built-in filtering for keywords. To determine which keywords were most effective at finding API-related bug reports, we performed a search using HackerOne's built-in bug report filter using keywords associated with APIs in general: API, Application Programming Interface, REST, RESTful, and SOAP. Then, we randomly sampled 25 bug reports for each keyword to determine the ratio of True Positives. Our sample results led us to use 'API' as our keyword when searching for API-related bug bounties.

- 'API' returned more than 1,000 results with 92% of the samples being related to APIs.
- 'Application Programming Interface' returned 15 results with 20% of results being related to APIs.
- 'REST' returned more than 1,000 results with 20% of the samples being related to APIs.
- 'RESTful' returned more than 1,000 results with 25% of the samples being related to APIs.
- 'SOAP' returned 21 results with 81% of the results being related to APIs.

We collected our data using an ASUS PN51-E1 computer, equipped with an AMD Ryzen 7 5700u processor, 32 GiB of DDR4-3200 SODIMM RAM, and a 1.0TB M.2 SSD. The computer was running Ubuntu 20.04.5 LTS (64-bit) via a 1Gbps internet connection provided by Spectrum Charter Communications. The internet connection was tested using Google's Measurement Lab, and it showed that it had a download speed of 717 Mbps and an upload speed of 40 Mbps, immediately after the completion of data collection.

We collected bug reports from HackerOne using a Python application [29] that scraped the HackerOne hacktivity webpage by providing a keyword of 'API', specifying a sort based on 'Popularity', and specifying a type of 'Disclosed'. The Python application [29] used a combination of Selenium and Beautiful Soup to control the web browser and parse the information received. We used Selenium to continuously scroll down on HackerOne's hacktivity page for a duration of 300 seconds to populate our cursory dataset. This step collected the bug report number, title, URL, severity rating, associated CVE ID, bounty amount, and number of upvotes. Then, each report's associated URL was visited to collect the bug report's date, weakness description, summary, and details exchanged between the security researcher and the host organization. The database was saved to an SQLite3 database for storage and exported to a CSV file for post-processing.

We collected bug reports from BugCrowd manually because BugCrowd does not offer a keyword filter, and only had 131 total disclosed bug reports to date. Our team manually viewed each disclosed bug report and entered bug reports associated with APIs into a Google spreadsheet [30].

We collected bug reports from Pentester.land by downloading the write-ups as a JSON file [3] and used a Python application [29] to parse the bug reports for our keyword 'API' in the Title field. The results were saved as a CSV file for post-processing.

B. Perform Quality Assessment, Sanitization, Classification

We removed 22 reports that did not contain sufficient details about the bug, 11 reports that were challenge event write-ups, and 29 reports where the company stated that the bug was known, previously disclosed, or previously identified but not yet corrected to ensure that duplicate reports were not included in our study. To further refine the data, we performed word sanitization to remove filler words and created a word cloud to easily identify key concepts. Additionally, vulnerabilities were grouped by common CVE names to assist in classification. This work was performed using an online tool written in Python [31]. We first ran the NLP on the dataset without stopwords. Then, we extracted the stopwords from the generated word cloud and re-ran the NLP on the dataset with the extracted stopwords. We repeated these steps until a valuable word cloud was generated, as shown in Table 1.

Vulnerability classification was catalogued into applicable terms and categories taken from OWASP and included: Broken Object Level Authorization (BOLA), Broken User Authentication, Excessive Data Exposure, Broken Function

[3]https://pentester.land/writeups.json

Level Authorization, and Mass Assignment, following the findings from Knight [3]. We manually renamed the weaknesses to match the categories from the OWASP, such as renaming Information Disclosure to Excessive Data Exposure, and Improper Access Control to Broken Access Control. Additionally, we combined some weaknesses into the same categories, such as Code/Command Injection and SQL Injection to Injection. The results can be found in Table 2.

C. Identify Vulnerabilities with Severity Ratings

After processing the data, we refined the dataset according to a severity taxonomy which includes critical, important, moderate, and low levels, following Microsoft's rating system [23]. Out of the total of 1056 reports, 125 were labeled as critical (11.84%), 232 were labeled as important (21.97%), 433 were labeled as moderate (41.00%), 178 were labeled as low (16.86%), and 88 were not labeled with any severity level (8.33%).

D. Quantify and Rank the Bugs

The bugs were categorized based on the number of occurrences and ranked according to their severity level. The severity level was determined by using the weighted values established by the Department of Defense Iron Bank in the Overall Risk Assessment Calculation [32]. The determination was heavily influenced by the severity rating and not by the number of occurrences, as a single injection attack can be more damaging to an organization than ten security misconfigurations or insufficient logging or monitoring events. The weight assigned to each severity level is as follows: Critical - 10, Important - 4, Moderate - 0.5, and Low - 0.25. We used the following equation to determine the rank:

$$Rank = NSf$$

$$N = NumberOfOccurrences$$

$$Sf = SeverityFactor$$

The vulnerabilities were grouped by OWASP vulnerability type, and each vulnerability was evaluated for its severity rating. We used the mode (the most commonly occurring value) of each severity rating within each vulnerability type to determine the most commonly occurring severity rating. If there were two modes, we selected the more severe rating as the mode for that vulnerability type. We then multiplied the number of occurrences by the most commonly occurring severity rating for each vulnerability type to determine the vulnerability type's rank. The results of this calculation are available in Table 2.

V. RESULTS

After processing the data and refining the dataset according to this taxonomy, we found that out of the total of 1056 reports, 158 reports were related to Information Disclosure, representing 14.96% of the total. 118 reports were related to Cross-Site Scripting (XSS) which is 11.17%, 76 reports were related to Improper Authentication which is 7.20%, 67 reports were related to Improper Access Control which is 6.34%, 57 reports were related to Cross-Site Request Forgery (CSRF) which is 5.40%, and the remaining 656 reports were related to other API vulnerabilities, accounting for 54.93%. We have provided the top 5 other API vulnerabilities to expand the category as follows: Code/Command Injection, Privilege Escalation, Insecure Direct Object Reference (IDOR), Server-Side Request Forgery (SSRF), and Violation of Secure Design Principles.

Vulnerability Category	Count
Excessive Data Exposure	169
Broken Access Control	114
Cross-Site Scripting (XSS)	104
Injection	90
Broken User Authentication	76
Cross-Site Request Forgery (CSRF)	56
Privilege Escalation	49
Server-Side Request Forgery (SSRF)	38
Denial of Service	34
Business Logic Error	31
Lack of Resource and Rate Limiting	28
Security Misconfiguration	20
Memory Corruption	19
Clear-text Storage of Sensitive Information	18
Cryptography Failures	13
Improper Validation of Array Index	11
Buffer Over-read	7
Open Redirect	7
Broken Object Level Authorization	7
Phishing	7

TABLE I
WORD CLOUD TOP 20 RESULTS FOR VULNERABILITY CATEGORIES

Vulnerability Category	Rank
Injection	467.25
Excessive Data Exposure	335.50
Broken User Authentication	263.25
Broken Access Control	254.25
Cross-Site Scripting (XSS)	248.25
Privilege Escalation	140.75
Cross-Site Request Forgery (CSRF)	103.75
Denial of Service	72.50
Server-Side Request Forgery (SSRF)	56.75
Security Misconfiguration	45.25
Memory Corruption	42.50
Business Logic Error	40.25
Broken Object Level Authorization	39.00
Clear-text Storage of Sensitive Information	34.75
HTTP Request Smuggling	29.25
Cryptography Failures	27.00
Lack of Resource and Rate Limiting	25.25
Open Redirect	19.75
Improper Assets Management	18.75
Buffer Over-read	16.25

TABLE II
RANKING OF THE TOP 20 VULNERABILITY CATEGORIES

Injection consists of sending malicious commands or codes to an API through a user input field, such as a text input or file upload. This allows malicious actors to send code or other executable commands to the API's interpreter, which can be used to bypass security, change permissions, access information, damage or disable the API. 8.5% of the reports with a severity ranking of 467.25 shows that it was

the most common and severe vulnerability in our dataset.

Excessive Data Exposure occurs when too much information is passed from the API to the client, with the client responsible for filtering what API resources and other information are displayed to the end-user. This can result in sensitive information being returned by the API. 16.0% of the reports with a severity ranking of 335.5 shows that it was the second most common and severe vulnerability in our dataset.

Broken User Authentication is an umbrella term for several vulnerabilities that attackers exploit to impersonate legitimate users online. These weaknesses can occur in two areas: session management and credential management. 7.2% of the reports with a severity ranking of 263.25 shows that it was the third most common and severe vulnerability in our dataset.

Broken Access Control occurs when users can act outside of their intended permissions, leading to unauthorized information disclosure, modification, or destruction of data, or performing business functions outside the user's limits. 10.8% of the reports with a severity ranking of 254.25 shows that it was the fourth most common and severe vulnerability in our dataset.

Cross-Site Scripting (XSS) attacks occur when an attacker uses a web application to send malicious code, typically in the form of a browser-side script, to a different end-user. Flaws that allow these attacks to succeed are widespread and occur when a web application uses input from a user within the output it generates without validating or encoding it. 9.8% of the reports with a severity ranking of 248.25 shows that it was the fifth most common and severe vulnerability in our dataset.

The other 5 vulnerabilities in the Top 10 are Privilege Escalation with a severity ranking of 140.75, Cross-Site Request Forgery (CSRF) with a severity ranking of 103.75, Denial of Service with a severity ranking of 72.5, Server-Side Request Forgery (SSRF) with a severity ranking of 56.75, and Security Misconfiguration with a severity ranking of 45.25.

From Figure 2, it can be seen that there are 125 reports identified as critical. Out of those reports, 27.2% are related to Injection, 12% are related to Broken User Authentication, 11.2% are related to Excessive Data Exposure, 10.4% are related to Broken Access Control, and 10.4% are related to Cross-site Scripting.

There are 232 reports identified as important, out of which 15.5% are related to Excessive Data Exposure, 12.5% are related to Injection, 10.3% are related to Broken User Authentication, 9.9% are related to Broken Access Control, and 9.5% are related to Cross-site Scripting.

There are 433 reports identified as moderate, out of which 20.1% are related to Excessive Data Exposure, 12.0% are related to Cross-site Scripting, 11.8% are related to Broken Access Control, 7.4% are related to Broken User Authentication, and 7.4% are related to Cross-site Request Forgery.

There are 178 reports identified as low, out of which 18.0% are related to Excessive Data Exposure, 15.2% are related to Broken Access Control, 9.6% are related to Cross-site Scripting, 6.2% are related to Lack of Resource and Rate Limiting, and 5.1% are related to Server-side Request Forgery.

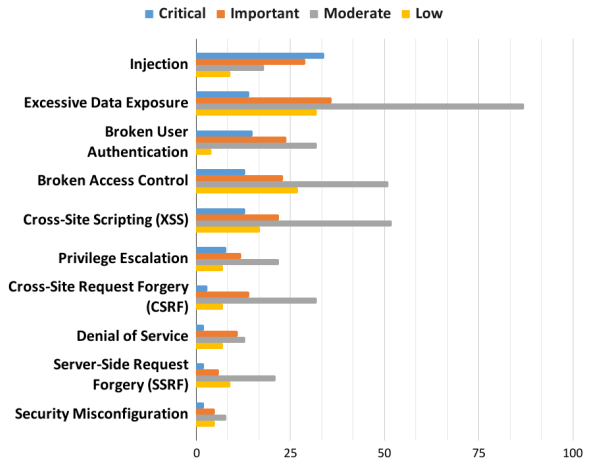

Fig. 2. Number of Occurrences and Severity for Top 10 Ranked Vulnerabilities

Therefore, it can be concluded that Injection, Excessive Data Exposure, Broken User Authentication, Broken Access Control, and Cross-site Scripting have contributed to the majority of findings with critical and important severity rating, 71.2% and 57.7% respectively. This is in agreement with the Top 10 table, as critical and important have a higher severity ranking.

By comparing Table 2 and Table 3, it can be seen that 8 vulnerabilities of the OWASP Top 10 2019 can be found in our dataset. Injection, which is in the 8th position in OWASP, is ranked as 1st position in our dataset, which might be due to an increased occurrence in the past three years. Excessive Data Exposure, which is in the 3rd position in OWASP, is ranked as 2nd position in our dataset, showing just a slight increase. Broken User Authentication, which is in the 2nd position in OWASP, is ranked as 3rd position in our dataset, showing a slight decrease in occurrence in the past three years. Security Misconfiguration, which is in the 7th position in OWASP, is ranked 10th in our dataset, showing a big decrease in occurrence. Broken Object Level Authorization, Improper Assets Management, Lack of Resource and Rate Limiting, and Broken Function Level Authorization, which are in the 1st, 9th, 4th, and 5th position in OWASP respectively, are ranked as 13th, 19th, 17th, and 25th in our dataset, indicating a large decrease in the occurrence of these vulnerabilities in the past three years, so less attention is needed. Mass Assignment and Insufficient Logging and Monitoring, which are in the 6th and 10th position in OWASP, are not found in our dataset, which might show that these two vulnerabilities have been tackled in the past three years. There are a total of 6 vulnerabilities that were not in the OWASP ranked in the Top 10 of our dataset, showing a great increase in the occurrence of

these vulnerabilities, which indicates that more attention is required from developers.

A. Answering RQ1, What are the common vulnerability and weakness types associated with application programming interfaces identified through publicly available bug bounty reports?

The most common vulnerabilities and weakness types associated with application programming interfaces were identified by grouping them into vulnerability types defined by OWASP and counting the number of vulnerabilities in each type. The 10 most common vulnerability and weakness types are:

1) Excessive Data Exposure (169 occurrences)
2) Broken Access Control (114 occurrences)
3) Cross-Site Scripting (XSS) (104 occurrences)
4) Injection Attacks (90 occurrences)
5) Broken User Authentication (76 occurrences)
6) Cross-Site Request Forgery (CSRF) (56 occurrences)
7) Privilege Escalation (49 occurrences)
8) Server-Side Request Forgery (SSRF) (38 occurrences)
9) Denial of Service (DOS) (34 occurrences)
10) Business Logic Error (31 occurrences)

B. Answering RQ2, Which common vulnerability and weakness types associated with application programming interfaces are rated critical or important using Microsoft's security update severity rating system?

Each vulnerability was manually assessed and assigned a severity rating using Microsoft's security update severity rating system. The vulnerability's description, impact, complexity, CVE ID, bounty, and upvotes were considered when assigning the severity rating. Then, the mode severity rating for each vulnerability type was determined by assessing the majority of severity ratings. Vulnerability types with less than five vulnerabilities were identified as crucial for understanding the data. As shown in Figure 1, the majority severity rating of each vulnerability was taken into consideration.

- Broken Object Level Authorization (Critical: 3 and Important: 2 out of 6 total, 71%)
- Broken Function Level Authorization (Critical: 1 out of 1 total, 100%) - Less than five vulnerabilities in this type.
- Buffer Overflow (Critical: 1 and Important 1 out of 3 total, 67%) - Less than five vulnerabilities in this type.
- HTTP Request Smuggling (Critical: 2 and Important 2 out of 7 total, 57%)
- Improper Assets Management (Critical: 1 and Important: 2 out of 5 total, 60%)
- Injection (Critical: 34 and Important: 29 out of 90 total, 70%)
- Broken User Authentication (Critical: 15 and Important: 24 out of 75 total, 52%)

C. Answering RQ3, Which OWASP Top 10 API Vulnerabilities are captured in the publicly available bug bounty reports for application programming interfaces?

The OWASP Top 10 API Vulnerabilities include Broken Object Level Authorization, Broken User Authentication, Excessive Data Exposure, Lack of Resource & Rate Limiting, Broken Function Level Authorization, Mass Assignment, Security Misconfiguration, Injection, Improper Assets Management, and Insufficient Logging and Monitoring. To represent the top vulnerabilities within the data, a system was utilized to quantify and rank the bugs based on severity ratings. The top 10 API vulnerabilities as assessed in this research using publicly available bug reports are:

1) Injection (rank 467.25)
2) Excessive Data Exposure (rank 335.5)
3) Broken User Authentication (rank 263.25)
4) Broken Access Control (rank 254.25)
5) Cross-Site Scripting (rank 248.25)
6) Privilege Escalation (rank 140.75)
7) Cross-Site Request Forgery (CSRF) (rank 103.75)
8) Denial of Service (rank 72.5)
9) Server-Side Request Forgery (SSRF) (rank 56.75)
10) Security Misconfiguration (rank 45.25)

Our findings align with four of the OWASP Top 10 API Vulnerabilities: Injection, Excessive Data Exposure, Broken User Authentication, and Security Misconfiguration. Additionally, three more OWASP Top 10 API Vulnerabilities were identified as within the top twenty vulnerability types: Improper Assets Management, Lack of Resource & Rate Limiting, and Broken Object Level Authorization. One OWASP Top 10 API Vulnerability, Broken Function Level Authorization, was identified as within the top twenty-five vulnerability types. Unfortunately, two of the OWASP Top 10 API Vulnerabilities, Mass Assignment, and Insufficient Logging and Monitoring, did not make the list.

VI. Discussion

A. Validate Prior Research

We cross-validated and compared our findings on vulnerability categories, number of occurrences, and severity levels with the OWASP Top 10 API Weaknesses. By comparing Table 2 with the OWASP Top 10, we found that only 8 out of the top 10 vulnerabilities have been identified in our findings. Four of them are in our top 10 vulnerabilities and the remaining four are in our top 25. To easily visualize the difference between our finding and OWASP Top 10, we have separated the table with sections as follows:

- OWASP Top 10 which were also in our top 10 findings.
- OWASP Top 10 which were identified in our findings but not in the top 10.
- OWASP Top 10 which were not identified in our findings.

B. THREATS TO VALIDATE

Due to limitations and the setup of the platforms, we were unable to retrieve more data than we currently have, which

OWASP Top 10	Assessed Rank
Injection	1
Excessive Data Exposure	2
Broken User Authentication	3
Security Misconfiguration	10
Broken Object Level Authorization	13
Lack of Resource and Rate Limiting	17
Improper Assets Management	19
Broken Function Level Authorization	25
Mass Assignment	NA
Insufficient Logging and Monitoring	NA

TABLE III
OWASP TOP 10 API VULNERABILITIES WITH ASSESSED RANK

may make our findings slightly off. The OWASP Top 10 vulnerabilities were adopted from Knight [3] and are based on the 2019 OWASP release, which may have changed since then. Therefore, our validation with OWASP Top 10 might show a large increase or decrease in occurrences that have been identified by the newer release by OWASP. Additionally, many bug reports are not disclosed to the public. During the data collection phase, we found that some reports have been closed to the public by request from the subject of that vulnerability due to safety concerns. Furthermore, some bugs that are applicable to APIs may have been missed due to our filtering keywords.

C. BENEFITS

Our goal is to show researchers the gap between industry-facing and academic-focused research. We hope to encourage more research in the area of common vulnerabilities, specifically in terms of their causes and ways to prevent or counter them.

We also want to warn developers about common vulnerabilities and their severity ranking, so they can pay more attention to these possible vulnerabilities during their development process. Additionally, we have shown them what to look for during the testing and maintenance phase.

Finally, we aim to encourage instructors to focus more on common vulnerabilities during their teaching process, so that students who will become developers or researchers in the future will have an understanding of these vulnerabilities and possible solutions to fix or prevent them.

VII. CONCLUSIONS

We conducted a large-scale empirical study on bug bounty reports related to API vulnerabilities and weaknesses. We identified common critical and important API vulnerabilities discovered in the bug bounty process using Microsoft's security update severity rating system. We also established a method to determine the severity ranking of a vulnerability by using the weighted values from the Department of Defense Iron Bank in the Overall Risk Assessment Calculation [32]. We cross-validated our findings with OWASP Top 10 to determine the change in occurrence of each vulnerability. Our research can assist developers in identifying real-world vulnerabilities and weaknesses, assigning severity to common API vulnerabilities, and validating the OWASP Top 10 for API vulnerabilities. Additionally, our study provides a pathway for researchers to study and tackle common vulnerabilities.

VIII. FUTURE WORK

In the future, we aim to collect more data on API vulnerabilities from various bug bounty platforms and companies. In addition to cross-validation with OWASP Top 10, we will conduct validation with existing mapped research publications on API vulnerabilities. We will also analyze trends in API security based on the date range and number of reports we collect. Furthermore, we will examine whether public colleges and universities or public security courses are teaching students about the causes of these vulnerabilities and how to prevent them.

REFERENCES

[1] D. Freeze, "Cybercrime to cost the world $10.5 trillion annually by 2025," Cybercrime Magazine, 27-Apr-2021. [Online]. Available: https://cybersecurityventures.com/cybercrime-damages-6-trillion-by-2021/. [Accessed: 29-Oct-2022].

[2] 21st Century Cures, vol. 114th. Congress, 2016.

[3] Alissa V. Knight, Playing With FHIR: Hacking and Securing FHIR API Implementations. Knight Ink, Las Vegas, NV, 2021.

[4] APIsecurity.io, "Issue 74: Vulnerability in login with Facebook, API security talks," API Security News, 12-Mar-2020. [Online]. Available: https://apisecurity.io/issue-74-vulnerability-in-login-with-facebook-api-security-talks/. [Accessed: 01-Oct-2022].

[5] E. Chickowski, "2018 sees API breaches surge with no relief in sight," Security Boulevard, 04-Dec-2018. [Online]. Available: https://securityboulevard.com/2018/12/2018-sees-api-breaches-surge-with-no-relief-in-sight. [Accessed: 01-Oct-2022].

[6] T. Brewster, "How hackers broke equifax: Exploiting a patchable vulnerability," Forbes, 14-Sep-2017. [Online]. Available: https://www.forbes.com/sites/thomasbrewster/2017/09/14/equifax-hack-the-result-of-patched-vulnerability/?sh=4ed3ca1a5cda. [Accessed: 01-Oct-2022].

[7] OnePoll, API Security Survey: A Survey of 250 IT Managers and Security Professionals. Imperva Inc., San Mateo, CA, 2017. https://www.slideshare.net/Imperva/api-security-survey.

[8] M. Bettendorf, "API growth rate continues to skyrocket in 2020 and into 2021," Postman Blog, 06-Apr-2022. [Online]. Available: https://blog.postman.com/api-growth-rate/. [Accessed: 01-Oct-2022].

[9] R. Davis, "Insecure API cloud computing: The causes and solutions," ExtraHop, 23-Jan-2020. [Online]. Available: https://www.extrahop.com/company/blog/2020/insecure-apis-cloud-computing-cause-solutions/. [Accessed: 01-Oct-2022].

[10] HackerOne, "#1 trusted security platform and hacker program," HackerOne. [Online]. Available: https://www.hackerone.com/. [Accessed: 20-Oct-2022].

[11] Bugcrowd, "#1 crowdsourced cybersecurity platform," Bugcrowd, 06-Jul-2022. [Online]. Available: https://www.bugcrowd.com/. [Accessed: 20-Oct-2022].

[12] Pentester, "Offensive InfoSec," Pentester, 23-Aug-2022. [Online]. Available: https://pentester.land/. [Accessed: 20-Oct-2022].

[13] Open Web Security Application Project (OWSAP), OWSAP API Security Project, [Online] Available: https://owasp.org/www-project-api-security/. [Accessed: 20-Oct-2022].

[14] N. M. Mohammed, M. Niazi, M. Alshayeb, and S. Mahmood, "Exploring software security approaches in software development lifecycle: A systematic mapping study," Comput. Standards Interfaces, vol. 50, pp. 107–115, Feb. 2017.

[15] A. Rahman, R. Mahdavi-Hezaveh, and L. Williams, "A systematic mapping study of infrastructure as code research," Inf. Softw. Technol., vol. 108, pp. 65–77, Apr. 2019.

[16] S. Zein, N. Salleh, and J. Grundy, "A systematic mapping study of mobile application testing techniques," J. Syst. Softw., vol. 117, pp. 334–356, Jul. 2016.

[17] F. A. Bhuiyan, M. B. Sharif, A. Rahma, "Security Bug Report Usage for Software Vulnerability Research: A Systematic Mapping Study" IEEE Access, Feb. 2021.

[18] HackerOne, "The Hacker-Powered Security Report - Who are Hackers and Why Do They Hack", June. 2018, p.23

[19] Aaron Yi Ding, De Jesus, G. Limon, M. Janssen, "Ethical hacking for boosting IoT vulnerability management: a first look into bug bounty programs and responsible disclosure", Proceedings of the Eighth International Conference on Telecommunications and Remote Sensing - ICTRS '19. Ictrs '19. Rhodes, Greece: ACM Press: 49–55, 2019

[20] Definition of exploit, TrendMicro, [Online] Available: https://www.trendmicro.com/vinfo/us/security/definition/exploit

[21] Vulnerability (computing), Wikipedia, [Online] Available: https://en.wikipedia.org/wiki/Vulnerability_(computing)

[22] Microsoft, "Security update severity rating system," Microsoft. [Online]. Available: https://www.microsoft.com/en-us/msrc/security-update-severity-rating-system. [Accessed: 29-Oct-2022].

[23] Security Update Severity Rating System, [Online] Available: https://www.microsoft.com/en-us/msrc/security-update-severity-rating-system

[24] S. Bigelow, "6 cloud vulnerabilities that can cripple your environment" [Online]. Available: https://www.techtarget.com/searchcloudcomputing/tip/6-cloud-vulnerabilities-that-can-cripple-your-environment.

[25] Farhan A. Qazi, "Insecure Application Programming Interfaces (APIs) in Zero-Trust Networks", Capitol Technology University, Dec. 2021.

[26] J. A. Diaz-Rojas, J. O. Ocharan-Hernandez, J. C. Perez-Arriaga, X. Limon, "Web API Security Vulnerabilities and Mitigation Mechanisms: A Systematic Mapping Study", 2021 9th International Conference in Software Engineering Research and Innovation (CONISOFT), 2021, p.207-218

[27] Information Technology Laboratory, National Vulnerability Database. [Online]. Available: https://nvd.nist.gov/. [Accessed: 20-Oct-2022].

[28] Common Vulnerabilities and Exposures. [Online]. Available: https://cve.mitre.org/. [Accessed: 20-Oct-2022].

[29] Craig Opie, Infosec_reports: web scraping tool, Github repository, [Online] Available: https://github.com/CraigOpie/infosec_reports

[30] Final Data, Google Sheet, [Online] Auvailable upon request: https://docs.google.com/spreadsheets/d/1ZKu8xTl5bDPMVredObPHrSF5rkej982l4xxl0hoXzP8/edit?usp=share_link

[31] Anthony Peruma, NLP_example, Github repository, [Online] Available upon request: https://github.com/iSQARE-Lab/NLP_Examples

[32] Iron Bank Value Stream, "Overall Risk Assessment (UNCLASSIFIED)," Platform One, 04-Apr-2022.