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 intend of this paper is to perform a large scale empirical study on bug bounty reports regarding API vulnerabilities and weaknesses and identified common critical and important API vulnerabilities discovered in the bug bounty process using Microsoft's security update severity rating system. Moreover, we have defined the method to determine the severity ranking of a vulnerability. We have cross-validate our findings with OWASP Top 10 to determine the change of occurrence of each vulnerability. Our research can help developers determine the real world vulnerabilities and weaknesses, assign the severity to the common API vulnerabilities, and validate the OWASP Top 10 regarding API. Furthermore, our study can provide a pathway for researchers to study and tackle the common vulnerabilities.**

I. INTRODUCTION

Secure software is the foundation of a quality software system. Business models have shifted to include security into the software development life-cycle due to the overall cost of insecure applications and systems, and cybercrime is expected to cost the world $10.5 Trillion annually by 2025 [30]. Despite the rise in cybercrime, on December 13th, 2016 Congress passed the 21st Century Cures Act [1] which requires health care providers to publish application programming interfaces (API) 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 health care standards organization, created the Fast Healthcare Interoperability and Resources (FHIR, pronounced "fire") API which built upon earlier versions of HL7 data format standards. However, FHIR API offered easy-to-use implementation through HTTP-based RESTful API protocol with resources represented in either RDF, XML, or JSON formatting that ultimately led to the FHIR API protocol being widely accepted across the healthcare industry. Research published by cybersecurity analyst Alissa Knight [2] 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 twenty-five thousand healthcare providers and payers.

Facebook announced a vulnerability of its API code which resulted in fifty million users' data being exposed and the API access token being stolen in September 2018 [3]. This event followed Cambridge Analytica's abuse of Facebook's API security infrastructure which allowed them to acquire user data on eighty million Facebook users [4]. Even when a patch exists which could secure the API framework, organizations large and small fail to implement the changes in a timely manner. Equifax failed to patch a vulnerability in the Apache Struts framework used to create their API and exposed the personal information and social security numbers of more than one hundred forty-three million US citizens with damages exceeding sixty-eight billion dollars [5].

The examples provided above are of large, known, security events that have been reported, but according to a survey conducted in 2017 on US companies with at least two hundred fifty employees or one million dollars in revenue, the average sized business manages as many as three hundred sixty-three public facing APIs [6]. Furthermore, the Postman API Platform - a platform used by more than seventeen 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 forty-six million as of January 2021 which indicates that API usage is rapidly increasing [7].

With the large number of businesses shifting their company's resources to external cloud computing servers to support client side and edge computing, compliance with government regulations, and access for remote workers, there is a greater reliance on third-party APIs which were designed for back-end applications not visible to the user [8]. However, generic reuse of existing third party APIs which were developed for a specific system may lead to vulnerabilities as identified by Knight [2]. Average sized businesses are more likely to skip research and development for a third party solution that is available. Unfortunately, Web Application Firewalls (WAF) and common security vulnerability and malware identification software most likely won't detect or prevent misuse of APIs because the attack vector is via the feature which makes APIs desirable. In order to secure APIs and protect existing zero trust networks, we need to compare mapped research publications regarding APIs with common vulnerabilities and weaknesses in API systems identified through bug bounty programs.

In this paper we investigate publicly available bug bounty reports from HackerOne[22], BugCrowd[23], and

*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

Pentester[24].land to identify common vulnerabilities and weaknesses regarding APIs and discuss the severity of the findings when compared to existing publications and known issues captured through common vulnerability and weakness enumeration utilizing mitre.org. This is in contrast to previous works which focus on performing surveys to identify demographics about API use, classifying previous API vulnerabilities, and balancing API security with API implementation as measured by functionality, speed, and performance. We agree with recent findings by Knight [2] that existing vulnerabilities can be classified into multiple categories aligned to the Open Web Application Security Project (OWASP[1]) API Security Top 10 [28]. 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 challenges developers and security researchers face by: determining real world vulnerability and weaknesses identified through applicable bug bounty reports, helping the developers assign severity to common API vulnerabilities, and validating existing research and security practices regarding APIs. Overall, developers and security researchers will have a better understanding in the area of API security from both academia and industry to enhance their own API security practices.

This paper is organized as follows: In Section II, we describe the goal of the study and research questions that we intend to address. In Section III we describe the necessary background and 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 use security by reports, 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. A researcher or analyst will identify a vulnerability and then search for similar vulnerabilities and weaknesses. Additionally, this information will allow researchers and developers to identify when a vulnerability or weakness is discovered to be widely effective. This will allow developers 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. Most of the time, it is up to the developers to determine severity of each vulnerability for each project; however, scanning common vulnerability and weakness data sets and applying severity ratings is a time consuming task which increases overhead that must be allocated by each project. By 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 will validate prior research applicability 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 recently in Software Engineering[9]-[11]. An SMS offers a 'map' of the research fields by classifying papers on the basis of the relevant categories and counting the work in each of those categories. An SMS offers a summary of the research domain to support researchers to identify topics that are well studied and to identify gaps that need further analysis [12]. For this research, we use a peer reviewed SMS titled Security Bug Report Usage for Software Vulnerability Research: A Systematic Mapping Study [11]

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 works as the back-end framework for applications. By nature, APIs expose application logic and sensitive data such as Personally Identifiable information (PII) and because of this have increasingly become a target for attackers. API security refers to the practice of preventing or mitigating attacks on APIs. This is a specific security that focuses on strategies to mitigate 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 by which individuals can receive recognition and

[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.

compensation for reporting bugs, especially those pertaining to security exploits and vulnerabilities. These programs allow the developers to discover and resolve bugs before the general public is aware of them, preventing incidents of widespread abuse and data breaches. Bug bounty programs have been implemented by a large number of organizations, including Facebook, Google, Microsoft, and the internet bug bounty.[16][17]

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, ore move deeper into the network.[18] 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.[19]

5) Microsoft Security and Severity Rating System: Microsoft security is a built-in software to Windows OS and 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 update. It will continually scan for malware, viruses, and security threats. [20] A severity rating system is published by Microsoft that rates each vulnerability according to the worst theoretical outcome where that vulnerability can be exploited to help customers understand the risk associated with each vulnerability they patch. It is intended to provide a broadly objective assessment of each issue and distinct from the likelihood of a vulnerability being exploited. There are four levels of the rating, critical, important, moderate and low. Each of the ratings has different identifiers. Critical, a vulnerability whose exploitation could allow code execution without user interaction (warnings or prompts). Important, a vulnerability whose exploitation could result in compromise of the confidentiality, integrity, or availability of data or processing resources with warnings or prompts. Moderate, the impact of the vulnerability is mitigated to a significant degree by factors such as authentication requirements or non-default configuration. Low, the impact of the vulnerability is comprehensively mitigated by the characteristics of the affected component.[21]

B. Related Work

To prevent performing a study that someone else had already performed, we first identified related studies and reviews to justify the need to conduct the study and why we believe it is timely to do so.

Farzana et al. [11] studied 46 publications that use security bug reports through a systematic inclusion and exclusion criteria and identified three topics that are addressed in their set of selections as follow (i) vulnerability classification, (ii) vulnerability summarization, and (iii) vulnerability dataset construction. Their study has indicated that there are potential research opportunities for further development in vulnerability analysis. While the study relates to the vulnerability classification and analysis, it does not provide a detailed description of the threats and aspects of the API security but more of software security in general.

Davis et al. [8] explored how and why the insecurity of APIs is being overlooked in a zero-security environment. Bigelow et al . [13] highlighted the concept of data vulnerability from using insecure APIs in a secure environment and discussed the effects of breaches and data loss resulting from API insecurity. Farhan et al. [14] focused on the security of APIs, and discovered how the culture and behavior of users and developers might have benefited from API security, or lack of, awareness.Also, it showed some of the reasons behind API insecurity to users, developers, and organizations, and how to improve the security of APIs without sacrificing the ease and usability of the actual API. These studies provided detailed information about the reasons behind the insecurity of APIs, some results due to the insecurity of APIs and some improvement advice. However, they do not provide empirical evidence of the cause of the insecurity of APIs, or the consequences due to the insecurity of APIs, or detailed identification of the API vulnerabilities.

Diaz-Rojas et al. [15] conducted the review and discovered 66 threats to web APIs in the literature, 21 techniques, 11 API design patterns and 34 methods which can be applied at a design level to detect, resist, react to or recover from the discovered threats. They also stated the finding about the relationship between the discovered threats and the discovered techniques in regarding the reported effectiveness of certain techniques or difficulty of defending against certain threats. Although they have provided detailed information about the web API vulnerabilities and weaknesses, and also some analysis alongside, it was narrowed down to web APIs only while there are more other types of APIs.

As a result of these findings, we come to the conclusion that to identify the common security gaps related to the API, we would need to conduct our own study, taking into account a large scale empirical study on bug bounty reports, to identify the critical and important API vulnerabilities. Also, we will verify our findings by comparing that with current bug bounties for validating a continuing vulnerability with APIs.

IV. METHODOLOGY

We investigated and determined common vulnerability and weakness types associated with APIs using vulnerability databases and bug bounty platforms. Some industry leaders

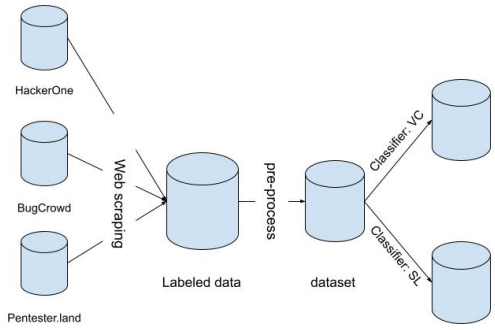

Figure 1: flow chart of the methodology

and government organizations publish patched security vulnerabilities which can be found in the National Vulnerability Database [26]. Others choose to ignore reporting guidelines or fail to take action to correct the vulnerability at all. A cybersecurity community effort to publish vulnerabilities to a public database, known as the Common Vulnerabilities and Exposures (CVE) database [27], has swept the world as a means to hold companies accountable for their cybersecurity and privacy responsibilities. Bug bounty platforms offer a unique look into the frequency, severity, and longevity of vulnerabilities after being published in the NVD and CVE databases. Our process consisted of the following 5 steps: (i) collect bug reports for API vulnerabilities; (ii) perform bug report quality assessment, sanitization, and classification using the OWASP [28] for API security; (iii) identify vulnerabilities with critical or important severity ratings using the Microsoft Security Update Severity Rating System; (iv) quantify and rank the bugs reported after being published in public vulnerability databases; and (v) validate 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 [25]. A total of 1028 reports have been extracted from HackerOne, 15 reports from BugCrowd, and 85 reports from Pentester.land which make a total of 1128 reports in the dataset. The data collected 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 amongst each other, the reports were able to be collected via web scraping, and allowed for convenient data processing within a python application [25], HackerOne and Pentester. land 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 with an AMD Ryzen 7 5700u, 32 GiB of DDR4-3200 SODIMM RAM, and 1.0TB M.2 SSD, running Ubuntu 20.04.5 LTS (64-bit) via a 1Gbps internet connection provided by Spectrum Charter Communications which earned speed test results using Google partnered with Measurement Lab of 717 Mbps download and 40 Mbps upload immediately after completion of data collection.

We collected bug reports from HackerOne using a Python application [25] that scraped the HackerOne hacktivity webpage providing a keyword of 'API', specifying a sort based on 'Popularity', and specifying a type of 'Disclosed'. The Python application [25] uses 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 host organization. The database was saved to an SQLite3 database for storage and exported to CSV for post processing.

We collected bug reports from BugCrowd manually because BugCrowd does not offer a keyword filter and only features 131 total disclosed bug reports to date. Each disclosed bug report was viewed independently by our team and bug reports associated with APIs were manually entered into a Google spreadsheet (Google Sheets). [32]

We collected bug reports from Pentester.land writeups by downloading the writeups as a JSON file [2] and used a Python application [25] to parse the bug reports for our keyword 'API' in the Title field. The results were saved as a CSV file for post processing.

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

B. Perform Quality Assessment, Sanitization, Classification

We removed 22 reports that did not contain details about the bug bug report as these may provide no value to our interest, we removed 11 reports that are challenge event writeup and we removed 29 reports where the company stated that the bug was knownrefiningosed, or previously identified but not yet corrected to ensure duplicate reports were not included in our study. We performed word sanitization to remove filler words and created a word cloud to easily identify key word concepts. Additionally, vulnerabilities were grouped by common CVE names to assist in the classification. This work was performed using an online tool written in python [29]. We first run the NLP on the dataset without stopwords. Then we extract the stopwords from the generated word cloud and then re-run the NLP on the dataset with the extracted stopwords. We have repeated these steps until a valuable word cloud is generated as shown in Table 1.

Vulnerability classification was cataloged into applicable terms and categories taken from OWASP and included: Broken Object Level Authorization (BOLA), broken user authentication, excessive data exposure, broken function level authorization, and mass assignment following the finding from [2]. We have manually renamed the weaknesses to match the categories from the OWASP such as Information Disclosure to Excessive Data Exposure, Improper Access Control to Broken Access Control. Also we have combined some weaknesses into the same categories such as Code/Command Injection and SQL Injection to Injection. The result can be found in Table 2.

C. Identify Vulnerabilities with Severity Ratings

Severity levels are critical, important, moderate and low following Microsoft's rating system [21]. After processing the data and refined the dataset according to this taxonomy, we find that out of the total of 1056 reports, 125 reports are labeled as critical which is 11.84%, 232 reports are labeled as important which is 21.97%, 433 reports are labeled as moderate which is 41.00%, 178 reports are labeled as low which is 16.86%, and 88 reports are not labeled with any severity level which is 8.33%.

D. Quantify and Rank the Bugs

The bug categories were quantified by the number of occurrences and ranked based on the associated severity level using the weighted values utilized by the Department of Defense Iron Bank in the Overall Risk Assessment Calculation [31]. The determination was weighted heavily on severity rating and not on the number of occurrences because even a single injection attack can be more damaging to an organization than ten security misconfigurations or insufficient logging or monitoring events. Critical severity holds a weight of 10, important severity holds a weight of 4, moderate holds a weight of 0.5, and low holds a weight of 0.25. We utilized the following equation to determine rank:

$$Rank = NSf$$

$$N = NumberOfOccurrences$$

$$Sf = SeverityFactor$$

The vulnerabilities were grouped by OWASP vulnerability type and each vulnerability was assessed for its severity rating. The mode for each severity rating within each vulnerability type was used to determine the most commonly occurring severity rating. In the event that two modes existed, we defaulted to a conservative approach of selecting the more severe severity rating as the mode for the vulnerability type. Then the most commonly occurring severity rating for each vulnerability type was multiplied with the number of occurrences to determine the vulnerability type's rank. The results of this calculation are available in Table 2.

V. RESULTS

Table 1: top 20 world cloud

Weakness	Count
Information Disclosure	158
Cross-Site Scripting (XSS)	118
Improper Authentication - Generic	76
Improper Access Control - Generic	67
Cross-Site Request Forgery (CSRF)	57
code/command injection	56
Privilege Escalation	54
Insecure Direct Object Reference (IDOR)	53
Server-Side Request Forgery (SSRF)	38
Violation of Secure Design Principles	38
Business Logic Errors	33
Denial of Service	31
Cleartext Storage/transmission of Sensitive Infor	19
Memory Corruption - Generic	17
Path Traversal	14
Cryptographic Issues - Generic	13
SQL Injection	11
Improper Restriction of Authentication Attempts	10
Out-of-bounds Read	10
Open Redirect	9

After processing the data and refined the dataset according to this taxonomy, we find that out of the total of 1056 reports, 158 reports are related to Information Disclosure which is 14.96%, 118 reports are related to Cross-Site Scripting (XSS) which is 11.17%, 76 reports are related to Improper Authentication which is 7.20%, 67 reports are related to Improper Access Control which is 6.34%, 57 reports are related to Cross-Site Request Forgery (CSRF) which is 5.40%, and rest 656 reports are related to other API vulnerabilities which is about 54.93%. We have provided the top 5 of the 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.

From Table 2, we introduce our Top 5 vulnerabilities with some definition and the percentage of reports with severity ranking. Also, we have included other 5 vulnerabilities that

Table 2: Ranking of top 20			
Ordered Rank	Ordered Weakness	Count	POS
467.25	Injection	90	1
335.5	Excessive Data Exposure	169	2
263.25	Broken User Authentication	76	3
254.25	Broken Access Control	114	4
248.25	Cross-Site Scripting (XSS)	104	5
140.75	Privilege Escalation	49	6
103.75	Cross-Site Request Forgery (CSRF)	56	7
72.5	Denial of Service	34	8
56.75	Server-Side Request Forgery (SSRF)	38	9
45.25	Security Misconfiguration	20	10
42.5	Memory Corruption	19	11
40.25	Business Logic Error	31	12
39	Broken Object Level Authorization	7	13
34.75	Cleartext Store of Sensitive Information	18	14
29.25	HTTP Requst Smuggling	7	15
27	Cryptographic Failures	13	16
25.25	Lack of Resource & Rate Limiting	28	17
19.75	Open Redirect	7	18
18.75	Improper Assets Management	5	19
16.25	Buffer Overread	7	20

are in our Top 10 with their severity ranking. To easily visualize the difference between our finding and OWASP Top 10, we have marked different background colors following the rules:

- Those found in our Top 10 and OWASP Top 10 have been marked with a Green background color.
- Those found in our Top 10 but not in OWASP Top 10 have been marked with a Blue background color.
- Those found in OWASP Top 10 but not in our Top 10 have been marked with a Red background color.
- All the others have been marked with a White background color.

The full table of ordered weaknesses in our dataset can be found in Appendix.

Injection consists of sending an API malicious commands/codes through a user input field, whether a text input, file upload, or other means. It 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% with a severity ranking of 467.25 shows that it was the most common severe vulnerability in our dataset.

Excessive Data Exposure is that too much information is passed on from the API to the client, with the client bearing the responsibility of filtering what API resources and other information are displayed to the end-user thus, an API may return sensitive information. 16.0% of the reports with a severity ranking of 335.5 shows that it was the second most common severe vulnerability in our dataset.

Broken User Authentication is an umbrella term for several vulnerabilities that attackers exploit to impersonate legitimate users online. Broadly, broken authentication refers to weaknesses 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 severe vulnerability in our dataset.

Broken Access Control is that users can act outside of their intended permissions and therefore, lead to unauthorized information disclosure, modification, or destruction of all data or performing a business function 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 severe vulnerability in our dataset.

Cross-Site Scripting (XSS) attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browin ser side script, to a different end user. Flaws that allow these attacks s to succeed are quite widespread and occur anywhere 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 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 severity ranking of 45.25.

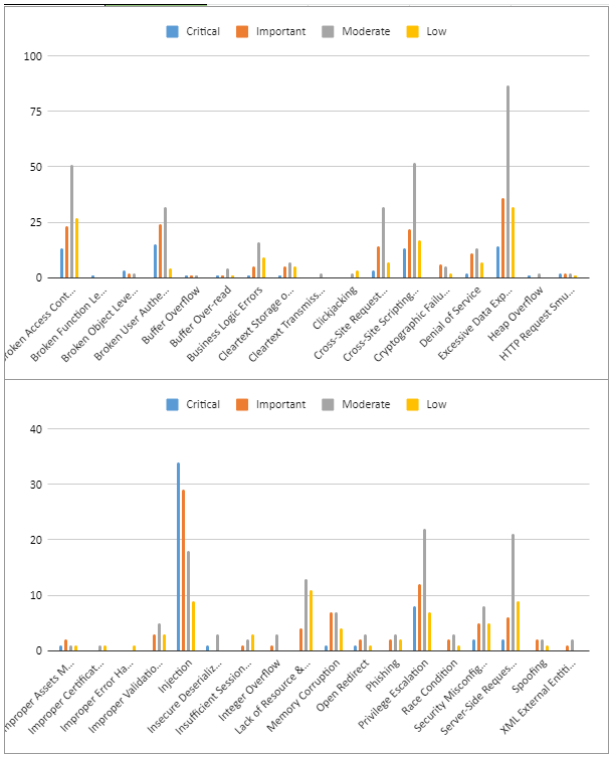

Figure 1: Number of Occurrences of Severity for each vulnerability ty

From Figure 1, we can find that there are 125 reports are identified as critical, out of those reports, 27.2% is related to Injection, 12% is related to Broken User Authentication, 11.2% is related to Excessive Data Exposure, 10.4% is related to Broken Access Control, and 10.4% is related to Cross-site Scripting.

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

There are 433 reports are identified as moderate, out of

those reports, 20.1% is related to Excessive Data Exposure, 12.0% is related to Cross-site Scripting, 11.8% is related to Broken Access Control, 7.4% is related to Broken User Authentication, and 7.4% is related to Cross-site Request Forgery.

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

Therefore, we can conclude that Injection, Excessive Data Exposure, Broken User Authentication, Broken Access Control, and Cross-site Scripting have contributed to the majority finding with the critical and important severity rating, 71.2% and 57.7% respectively. This agrees with our Top 10 table as critical and important have a higher severity ranking.

By comparing Table 2 and Table 3 we can see that 8 vulnerabilities of the OWASP Top 10 2019 can be found in our dataset. Injection which is the 8th position in OWASP ranked as 1st position in our dataset which might be due to the increased occurrence in the past three years. Excessive Data Exposure which is the 3rd position in OWASP ranked as 2nd position in our dataset which shows just a slight increase. Broken User Authentication which is the 2nd position in OWASP ranked as 3rd position in our dataset which shows a slight decreased occurrence in the past three years. Security Misconfiguration which is 7th position in OWASP ranked 10th in our dataset which shows a big decrease in occurrence. Broken Object Level Authorization, Improper Assets Management, Lack of Resource & Rate Limiting, and Broken Function Level Authorization which are 1st, 9th, 4th, and 5th position in OWASP respectively ranked as 13th, 19th, 17th, and 25th in our dataset which means a large decrease in occurrence of these vulnerabilities in the past three years so that less attention is needed. Mass Assignment and Insufficient Logging & Monitoring which are 6th, and 10th 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 Top 10 of our dataset showing a great increase of occurrence of those vulnerabilities which indicates more attention is required from the 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 identified through publicly available bug bounty reports were assessed by grouping vulnerabilities into vulnerability types as 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 has been manually assessed and assigned a severity rating using Microsoft's security update severity rating system. Information pertaining to the vulnerability's description, impact, complexity, CVE ID, bounty, and upvotes were considered when assigning the severity rating. Then each vulnerability type was assessed for the mode severity rating, we failed conservative and defaulted to the more severe severity rating in the event of more than one mode, to determine the most common severity rating for each vulnerability type. We determined that vulnerability types with less than five vulnerabilities should be identified because this information is crucial for fully understanding the data. As shown in Figure 1, taking the majority severity rating of each vulnerability.

- 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 have a better representation of the top vulnerabilities within our data, we utilized a system that quantified and ranked 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 agree 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 to be within our top twenty vulnerability types. Those vulnerabilities are Improper Assets Management, Lack of Resource & Rate Limiting; and Broken Object Level Authorization. One OWASP Top 10 API Vulnerability namely Broken Function Level Authorization was identified to be within our top twenty-five vulnerability types. Unfortunately, two of the OWASP Top 10 API Vulnerabilities did not make our list: Mass Assignment and Insufficient Logging and Monitoring.

VI. Discussion

A. Validate Prior Research

We cross-validated and compared the vulnerability categories, number of occurrences, and severity levels with OWASP Top 10 API Weaknesses. By comparing Table 2 and Table 3, we find only 8 vulnerabilities out of the Top 10 have been identified in our finding while 4 of them are in our Top 10 as well while the other 4 are in our Top 25.

Table 3: OWASP Top 10 2019

POS	API Weaknesses	Rank
1	Injection	8
2	Excessive Data Exposure	3
13	Broken Object Level Authorization	1
3	Broken User Authentication	2
19	Improper Assets Management	9
17	Lack of Resource & Rate Limiting	4
25	Broken Function Level Authorization	5
10	Security Misconfiguration	7
-	Mass Assignment	6
-	Insufficient Logging and Monitoring	10

B. THREATS TO VALIDATE

Due to the limitation and setup of the platforms, we were unable to retrieve more data than we have right now which makes the finding possibly slightly off. The vulnerability OWASP Top 10 was adopted from Knight [2] which is based on the 2019 OWASP's release which might have been changed. Thus the validation with OWASP Top 10 might show a large increase or decrease in occurrence that has been identified by the newer release by OWASP. Majority of Bug Reports are not Disclosed to the public, during the data collecting phase we have found some reports have been closed to the public by the request from the subject of that vulnerability due to the safety concern. Additionally, some bugs that are applicable to APIs may have been missed due to our filtering keywords.

C. BENEFITS

We intend to show the researchers the gap between industry facing and academic focusing. Thus, encouraging more research to be conducted on the area of the common vulnerabilities in terms of the cause and ways to prevent or counter those vulnerabilities.

We want to warn the developers of the common vulnerability with the severity ranking so they can pay more attention to these possible vulnerabilities during their development process. Also, we have shown them what they should be looking for during the testing and maintenance phase.

We try to encourage the instructors to focus more on the common vulnerabilities during their teaching process. So the students who will be the developers or researchers in the future will have the idea about those vulnerabilities and also some possible solutions to fix or prevent those vulnerabilities.

VII. CONCLUSIONS

We have performed a large scale empirical study on bug bounty reports regarding API vulnerabilities and weaknesses and identified common critical and important API vulnerabilities discovered in the bug bounty process using Microsoft's security update severity rating system. Moreover, we have defined the method to determine the severity ranking of a vulnerability using the weighted values from the Department of Defense Iron Bank in the Overall Risk Assessment Calculation [31]. We have cross-validate our findings with OWASP Top 10 to determine the change of occurrence of each vulnerability. Our research can help developers determine the real world vulnerabilities and weaknesses, assign the severity to the common API vulnerabilities, and validate the OWASP Top 10 regarding API. Furthermore, our study can provide a pathway for researchers to study and tackle the common vulnerabilities.

VIII. FUTURE WORK

We plan to collect more data in regards to the API vulnerabilities from more bug bounty platforms and companies. Besides cross validation with OWASP Top 10, we will conduct the validation with existing mapped research publications regarding API vulnerabilities. We will determine the trend in API security based on date range and number of reports that we collected. Also, we will determine if public colleges and universities or public security courses are teaching students about the cause of these vulnerabilities and how to prevent those vulnerabilities.

References

[1] 21st Century Cures, vol. 114th. Congress, 2016.
[2] Alissa V. Knight, Playing With FHIR: Hacking and Securing FHIR API Implementations. Knight Ink, Las Vegas, NV, 2021.

[3] 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].

[4] 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].

[5] 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].

[6] 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.

[7] 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].

[8] 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].

[9] 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.

[10] 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.

[11] 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.

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

[13] 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.

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

[15] 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

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

[17] 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

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

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

[20] 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].

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

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

[23] Bugcrowd, "#1 crowdsourced cybersecurity platform," Bugcrowd, 06-

Jul-2022. [Online]. Available: https://www.bugcrowd.com/. [Accessed: 20-Oct-2022].
[24] Pentester, "Offensive InfoSec," Pentester, 23-Aug-2022. [Online]. Available: https://pentester.land/. [Accessed: 20-Oct-2022].
[25] Craig Opie, Infosec_reports: web scraping tool, Github repository, [Online] Available: https://github.com/CraigOpie/infosec_reports.
[26] Information Technology Laboratory, National Vulnerability Database. [Online]. Available: https://nvd.nist.gov/. [Accessed: 20-Oct-2022].
[27] Common Vulnerabilities and Exposures. [Online]. Available: https://cve.mitre.org/. [Accessed: 20-Oct-2022].
[28] Open Web Security Application Project (OWSAP), OWSAP API Security Project, [Online] Available: https://owasp.org/www-project-api-security/. [Accessed: 20-Oct-2022].
[29] Anthony Peruma, NLP_example, Github repository, [Online] Available upon request: https://github.com/iSQARE-Lab/NLP_Examples
[30] 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].
[31] Iron Bank Value Stream, "Overall Risk Assessment (UNCLASSIFIED)," Platform One, 04-Apr-2022.
[32] Final Data, Google Sheet, [Online] Auvailable upon request: https://docs.google.com/spreadsheets/d/1ZKu8xTl5bDPMVredObPHrSF5rkej982l4xxl0hoXzP8/edit?usp=share_link

APPENDIX

Table 4 and Table 5

	Critical	Important	Moderate	Low
Broken Access Control	13	23	51	27
Broken Function Level Authorization	1	0	0	0
Broken Object Level Authorization	3	2	2	0
Broken User Authentication	15	24	32	4
Buffer Over-read	1	1	4	1
Buffer Overflow	1	1	1	0
Business Logic Errors	1	5	16	9
Cleartext Storage of Sensitive Information	1	5	7	5
Cleartext Transmission of Sensitive Information	0	0	2	0
Clickjacking	0	0	2	3
Cross-Site Request Forgery (CSRF)	3	14	32	7
Cross-Site Scripting (XSS)	13	22	52	17
Cryptographic Failures	0	6	5	2
Denial of Service	2	11	13	7
Excessive Data Exposure	14	36	87	32
Heap Overflow	1	0	2	0
HTTP Request Smuggling	2	2	2	1
Improper Assets Management	1	2	1	1
Improper Certificate Validation	0	0	1	1
Improper Error Handling	0	0	0	1
Improper Validation of Array Index	0	3	5	3
Injection	34	29	18	9
Insecure Deserialization	1	0	3	0
Insufficient Session Expiration	0	1	2	3
Integer Overflow	0	1	3	0
Lack of Resource & Rate Limiting	0	4	13	11
Memory Corruption	1	7	7	4
Open Redirect	1	2	3	1
Phishing	0	2	3	2
Privilege Escalation	8	12	22	7
Race Condition	0	2	3	1
Security Misconfiguration	2	5	8	5
Server-Side Request Forgery (SSRF)	2	6	21	9
Spoofing	0	2	2	1
XML External Entities (XXE)	0	1	2	0

Severity Rating Count for each vulnerability

Count	Ordered Rank	Ordered Weakness
90	900	Injection
169	85.5	Excessive Data Exposure
76	38	Broken User Authentication
114	57	Broken Access Control
104	53	Cross-Site Scripting (XSS)
49	24.5	Privilege Escalation
56	28.5	Cross-Site Request Forgery (CSRF)
34	17	Denial of Service
38	19	Server-Side Request Forgery (SSRF)
20	10	Security Misconfiguration
19	76	Memory Corruption
31	16	Business Logic Error
7	70	Broken Object Level Authorization
18	9	Cleartext Store of Sensitive Information
7	70	HTTP Requst Smuggling
13	52	Cryptographic Failures
28	14	Lack of Resource & Rate Limiting
7	3.5	Open Redirect
5	20	Improper Assets Management
7	3.5	Buffer Overread
11	5.5	Improper Validation of Array Index
3	30	Buffer Overflow
4	2	Insecure Deserialization
3	1.5	Heap Overflow
1	10	Broken Function Level Authorization
7	3.5	Phishing
6	3	Race Condition
6	3	Spoofing
6	1.5	Insufficient Session Expiration
4	2	Integer Overflow
3	1.5	XML External Entities (XXE)
5	1.25	Clickjacking
2	1	Cleartext Transmission of Sensitive Information
2	1	Improper Certificate Validation
1	0.25	Imporer Error Handling

Severity Ranking for all vulnerabilities in the Dataset with count