A First Look at Security and Privacy Risks in the RapidAPI Ecosystem

Anonymous Author(s)

ABSTRACT

With the emergence of the open API ecosystem, third-party developers can publish their APIs on the API marketplace, significantly facilitating the development of cutting-edge features and services. The RapidAPI platform is currently the largest API marketplace and it provides over 40,000 APIs, which have been used by more than 4 million developers. However, such open API also raises security and privacy concerns associated with APIs hosted on the platform. In this work, we perform the first large-scale analysis of 32,089 APIs on the RapidAPI platform. By searching in the GitHub code and Android apps, we find that 3,533 RapidAPI keys, which are important and used in API request authorization, have been leaked in the wild. These keys can be exploited to launch various attacks, such as Resource Exhaustion Running, Theft of Service, Data Manipulation, and User Data Breach attacks. We also explore risks in API metadata that can be abused by adversaries. Due to the lack of a strict certification system, adversaries can manipulate the API metadata to perform typosquatting attacks on API URLs, impersonate other developers or renowned companies, and publish spamming APIs on the platform. Lastly, we analyze the privacy non-compliance of APIs and applications, *e.g.*, Android apps, that call these APIs with data collection. We find that 1,709 APIs collect sensitive data and 94% of them don't provide a complete privacy policy. For the Android apps that call these APIs, 50% of them in our study have privacy non-compliance issues.

1 INTRODUCTION

With the rise of web and mobile applications, open APIs (application programming interface) that are published on the Internet and are free to access by developers are becoming popular [14]. Different from private APIs that are only accessible to an organization, open APIs are publicly available for all developers to access. This paradigm greatly benefits developers and accelerates the software development process, as evidenced by the increasing number of developers using open APIs. To meet the demands for rapid development, different open API ecosystems, such as RapidAPI [17], ProgrammableWeb [16], and OpenAPIHub [15] have been surged in recent years.

The RapidAPI platform is the largest and dominant open API platform and it provides over 40,000 APIs for more than 4 million developers [17]. There are over 1 trillion API calls per month on the RapidAPI platform. Such a large number of APIs benefit from the openness of the platform. Developers can not only use existing APIs on the platform but also develop and publish their APIs on the platform, enriching the platform for the entire developer community. However, the openness of the platform introduces security and privacy challenges when meeting numerous inexperienced developers. Similar issues also exist in other popular open platforms that allow third-party developers to publish different types of apps, *e.g.*, Apple App Store [27], Google Play [45], Chrome Extensions [19], and Amazon Alexa [24]. To prevent potential issues and ensure the safety of third-party applications, these platforms have defined a list of policy requirements to be adhered to by applications [1, 5, 9, 12]. After an application is submitted to the app store, it needs to pass a certification review process that validates if any policy violations exist. Even so, problematic apps are still found on these platforms [23, 25, 33]. This motivates us to explore the potential security and privacy risks existing in the emerging open API ecosystem.

On the RapidAPI platform, developers need to provide their RapidAPI keys in API requests for authorization. However, developers may inadvertently disclose such keys in their public code, such as the code on the GitHub platform or Android apps, leading to potential security risks. Since the RapidAPI platform doesn't provide a strict certification system to validate API quality, it may lead to API metadata abuse and problematic APIs being published to the platform. In addition, APIs may collect user data for different functionalities, such as using user location to find nearby restaurants. Developers should provide a privacy policy to notify users about the data collection behaviors. However, the RapidAPI platform doesn't define any requirements to restrict such data collection behaviors, leading to serious privacy non-compliance risks.

In this work, we aim to answer the following research questions: RQ1: How predominant is the RapidAPI key leak in the wild, and what attack can the adversary perform using leaked keys? RQ2: How can API metadata be abused and what are the consequences? RQ3: Do APIs on the RapidAPI platform and downstream applications, such as Android apps, have privacy issues regarding data collection? To answer the above questions, we perform a systematic security and privacy risk analysis of the RapidAPI ecosystem. In summary, we make the following contributions:

- To the best of our knowledge, this work is the first large-scale analysis of the security and privacy risks associated with open APIs [1] hosted on the RapidAPI platform. We conduct our analysis on a dataset with 32,089 APIs. In addition, we analyze the other two open API platforms and check if they have similar issues. We share our dataset, analysis tools and results with the research community for future research [2].
- We find that 3,533 keys used for API request authorization have been leaked by developers in GitHub or Android code and 98% of the leaked keys are still usable. We demonstrate that adversaries can utilize the leaked keys to launch different attacks, *e.g.*, Resource Exhaustion Running attack,

[1]For simplicity, we refer to the APIs published on the RapidAPI platform as "APIs" in our work.

[2]The materials of this work are available at https://github.com/RapidAPI-research/RapidAPI-research.

Theft of Service attack, Data Manipulation attack, and User Data Breach attack.

- We investigate the risks of API metadata abuse. Due to the poor certification process in the RapidAPI platform, adversaries can publish any APIs on the platform, such as typosquatting APIs to attack other APIs or spamming APIs for product promotion. Malicious developers can also impersonate well-known companies to mislead developers into calling their APIs.
- We analyze the privacy compliance of APIs on the RapidAPI platform and Android apps that call these APIs. We find 1,709 APIs that collect sensitive data and 94% of them don't provide a complete privacy policy. Such privacy compliance issues in APIs could propagate to downstream applications, leading to 50% of Android apps that call data collection APIs not providing a complete privacy policy.

2 BACKGROUND

2.1 RESTful API

REST (Representational State Transfer) is a software architectural style for building web services that allow communication between applications over the Internet [28]. Web service API conforming to the REST architectural style is called RESTful API. Due to the rise of web and mobile applications that need to communicate with each other, the use of RESTful APIs has become increasingly popular.

The RapidAPI platform uses the RESTful API for developers to access API functions and get responses easily. Developers only need to send requests using standard HTTP. Then, the server will receive the request and respond with a standard format. This makes it easy for developers using different program languages to communicate with each other. Another advantage of the RapidAPI platform is that for each API, it provides code snippets in 20 different programming languages, *e.g.*, C, C#, GO, HTTP, Java, JavaScript, Node.js, PHP, Python, R, and Ruby, or Shell. Developers can easily choose the appropriate language and directly use the code for their applications. An API on the RapidAPI platform contains multiple endpoints, each of which corresponds to a function. To invoke an API, developers only need to call an endpoint, which is the digital location where an API receives requests about a specific function. APIs on the RapidAPI platform can also provide access and operations to create, read, update, and delete (CRUD) the data stored by developers within the API.

2.2 Stakeholders of Open APIs

Three stakeholders are involved when an API is called and used: API provider, API consumer, and end user. In particular, developers may take two roles on the RapidAPI platform: the API provider or the API consumer, using the same account. Figure 1 shows the relationship between them during the API lifecycle.

API Provider: Developers can serve as the API providers if they develop APIs and publish them to the RapidAPI platform. After registering an account on the RapidAPI platform, API providers can create an API by setting up API metadata and defining API functions. In addition, API providers can add access control, such as authorization, to protect API security. API providers can also add pricing plans within an API to generate income. At last, API providers publish APIs to the RapidAPI platform so that other developers can access these APIs.

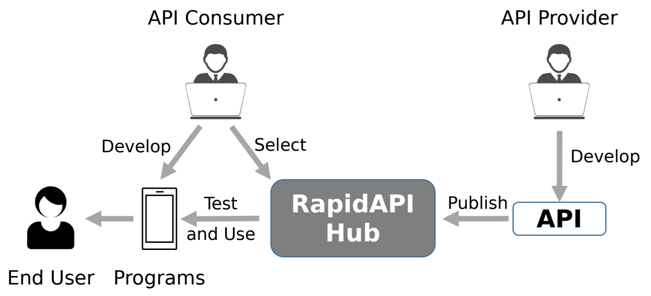


Figure 1: Different stakeholders of APIs on the RapidAPI platform.

API Consumer: Developers can also serve as the API consumers who develop software applications that use APIs from the RapidAPI platform. API consumers can search for an API or browse APIs in different categories on the RapidAPI platform. When API consumers find their desired APIs, they can subscribe to the APIs for testing. After choosing the appropriate programming language, API consumers can integrate the code snippets provided by the RapidAPI platform into their applications. In addition, the RapidAPI platform allows each developer to follow other developers or APIs, forming a developer following network

End User: End users use the applications that call APIs in the background. Although end users don't interact with APIs directly, the APIs may have an impact on the application functionalities and user experience. For example, if an API service is unavailable, the application can't work, and end users will be negatively influenced. Also, applications may collect user data to perform certain functionalities in APIs. In such cases, the application needs to provide a privacy policy to notify users of such data collection and data usage. For simplicity, we refer to the end users as users in our work.

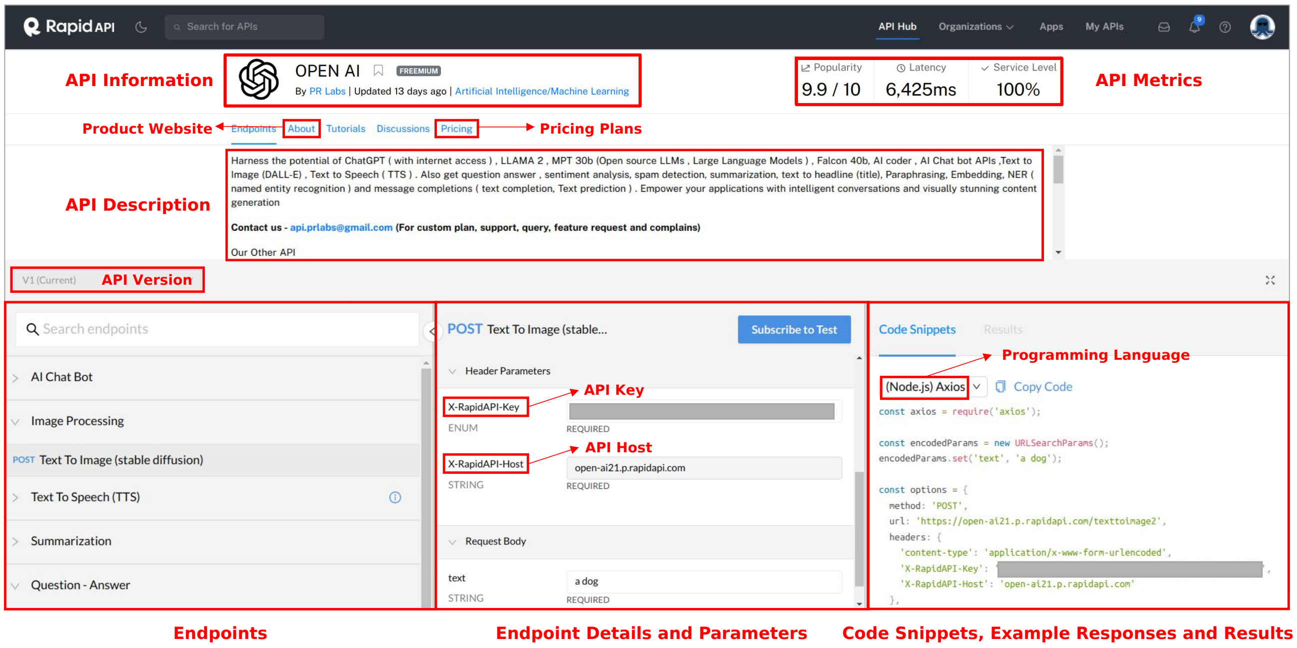
2.3 Threat Model

Our study focuses on understanding the security and privacy risks in the RapidAPI platform, including key leaks, API metadata abuse risks and privacy issues. In Section 4, we found key leaks in GitHub and Android code. Once an adversary obtains an API consumer's key and knows the APIs that the API consumer has subscribed to, the adversary can perform different attacks on the leaked keys and subscribed APIs. We assume the owner of the keys doesn't monitor the key usage data so that the adversary can call an API using the leaked key. In Section 5, we explore the risks of API metadata abuse. We assume the platform and developers' accounts are not compromised and adversaries can't hack other developers' accounts. However, adversaries can register their own accounts on the RapidAPI platform and publish their APIs on the platform.

2.4 Ethical Consideration

During our experiments, we employed the following strategies to minimize any risk to APIs or developers.

- **Data Collection.** In Section 3, we collected an API dataset for our analysis. During our data collection, we added a time interval



Harness the potential of ChatGPT (with internet access) , LLAMA 2 , MPT 30b (Open source LLMs , Large Language Models) , Falcon 40b, AI coder , AI Chat bot APIs ,Text to Image (DALL-E) , Text to Speech (TTS) . Also get question answer , sentiment analysis, spam detection, summarization, text to headline (title), Paraphrasing, Embedding, NER (named entity recognition) and message completions (text completion, Text prediction) . Empower your applications with intelligent conversations and visually stunning content generation

Contact us - api.prlabs@gmail.com (For custom plan, support, query, feature request and complains)

Our Other API

[redacted]

```
const axios = require('axios');

const encodedParams = new URLSearchParams();
encodedParams.set('text', 'a dog');

const options = {
  method: 'POST',
  url: 'https://open-ai21.p.rapidapi.com/texttoimage2',
  headers: {
    'content-type': 'application/x-www-form-urlencoded',
    'X-RapidAPI-Key': '[redacted]',
    'X-RapidAPI-Host': 'open-ai21.p.rapidapi.com'
  },
```

Figure 2: Listing page of the "OPEN AI" API published on the RapidAPI platform.

between each visit and ensured we did not influence the platform services or other developers.

- **Leaked Keys from Developers.** In Section 4, we searched for the leaked keys in the GitHub and Android code. We developed a codebook to store developers' data and extract the corresponding APIs that each key subscribes to. Then, we analyzed these APIs to exploit different attacks. We only used 100 keys to validate their usability in an ethical way and ensured that our action didn't have any impact on these keys.
- **Experiments on Attacks Using Our Keys.** In Section 4, to validate the feasibility of proposed attacks, we conducted experiments on the APIs that we developed. We ensured that we conducted all attacks using our own accounts. We used our private APIs for testing and didn't publish them to the platform. We didn't launch attacks against any public API on the platform.
- **Responsible Disclosure.** We have reported our findings, including mitigation strategies, to the RapidAPI platform. We found that the RapidAPI platform has removed almost all spamming APIs, possibly due to our reporting. We look forward to collaborating with them to fix the vulnerabilities and notify developers about the consequences of key leaks.

3 RAPID API DATA COLLECTION

3.1 Crawling API Listing Page

RapidAPI platform provides a listing page for each public API to present API information, as shown in Figure 2. We crawl all the content within the API listing page in our data collection. There are two main portions on the API listing page. The top portion of the webpage presents the basic API information, *e.g.*, API name, developer, updated time, and category. Below that, developers can navigate to different sections about the API, including "Endpoints", "About", "Tutorials", "Discussions", and "Pricing". The "Endpoints" tab displays the API description and the API client to test the API (lower portion of the listing page). The "About" tab shows the API documentation, API product website, and a way to contact the API provider. The "Tutorials" tab displays the tutorials created by the API provider and the "Discussions" tab presents the discussions posted by API consumers. At last, the "Pricing" tab lists all the pricing plans of an API that API consumers can subscribe to.

The lower half of the listing page shows the details of API endpoints to access different functions within an API. The left portion shows the selected API version and all endpoints in the API. After clicking an endpoint name, the middle portion will present endpoint details, *e.g.*, the HTTP method, the endpoint description, and the parameters in this endpoint. In particular, the parameters "X-RapidAPI-Key" (API key) and "X-RapidAPI-Host" (API host) are used for authorization and mandatory for each API request. The right portion provides a code snippet after developers select the preferred programming languages from a list of options, *e.g.*, C, Java, or Python. The headers and parameters for requests in code snippets are from the developer's input in the middle portion so that developers can easily integrate the API code snippets into their application code. The example responses and results are provided after developers execute a request test on the API listing page.

3.2 Collecting API Data

To help developers discover their desired APIs, the RapidAPI platform provides 49 categories, *e.g.*, Sports, Finance, and Data, with different functionalities of APIs. The RapidAPI platform also provides APIs in "Collections", which contains a list of APIs with a common characteristic, *e.g.*, "Popular APIs" or "Recommended APIs". In addition, developers can directly search with keywords, such as "ChatGPT", in the search bar and the platform would return the top-ranked APIs based on the API name and API description. After clicking a category/collection or searching for a keyword, a list of APIs will be displayed. Developers can select the API based on the API name, brief description, and three API metrics: popularity, latency, and service level (percentage of successful calls of the API in the last 30 days). Upon selecting an API from the search results, the API listing page will be displayed.

After collecting APIs from all 49 categories and 527 collections, we only obtained 16K unique APIs, which is less than the 40K APIs that the platform claims. This is because the RapidAPI platform only presents the first 1,000 APIs in each search result. Considering that API providers may use similar words to describe APIs with similar functionalities, we first downloaded the 16K APIs and built up a keyword list based on the words in their descriptions. Then, we used these keywords to search for more APIs. As a result, we used 16,904 words for API searching and obtained 32,089 APIs, which covers most of the APIs as the platform claimed.

We developed a web crawler that automatically visited each category or collection and searched for keywords. Then, we saved all the API links in the results. After that, the web crawler browsed each API listing page and obtained API details. We collected all the available API information discussed in Section 3.1, *e.g.*, API name, developer, update time, category, and endpoints. For each endpoint, we obtained its name, HTTP method, description, parameter, and the endpoint URL. For each parameter, we stored the parameter name, type, provided example, and description.

We successfully downloaded the 32,089 APIs from the RapidAPI platform. We removed 12,452 spamming APIs from our dataset (more details in Section 5.2.2) and did not use them in our analysis. On average, each API provider developed 1.5 APIs and each API provides 3.7 endpoints. In 49 categories, the "Data" and "Tool" categories contain the highest number of APIs, comprising 13% and 11% of the total number of APIs, respectively. 60% of APIs come with the performance statistics for popularity, latency and service level, which means API consumers have called them recently. The average latency is 6,313 ms for all APIs, and the largest latency is 1,016,856 ms (17 minutes) from an unpopular API. The average service level (successful call rate) is 74.5% for all APIs and calls to 2,581 APIs (13%) never succeeded in the last 30 days. 43% of APIs are Free APIs, 52% are Freemium APIs (APIs with a limited free tier), and 5% are Paid APIs. Interestingly, for the popularity, latency and service level, the Paid APIs perform worse on average than Freemium APIs.

4 ATTACKS WITH LEAKED KEYS

4.1 Keys in RapidAPI Authorization

To improve security during API invocations, the RapidAPI platform provides several authorization methods. By default, RapidAPI provides a basic authorization named "RapidAPI Auth", which is the simplest form of authorization for API consumers. In addition, RapidAPI provides other types of authorization that can be optionally added, such as authorization with API consumer's ID and password in header parameters, as shown in Figure 3, and authorization that requires API consumers to add query string parameters to API requests. Such additional authorizations ask for more data besides the API consumers' keys and can protect APIs from potential attacks with leaked keys.

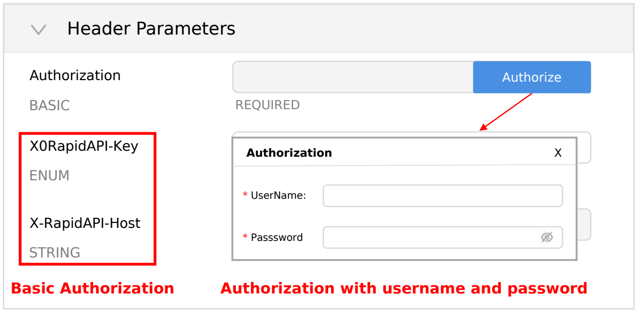


Figure 3: Different authorization methods provided by the RapidAPI platform.

In our dataset, we observed that most APIs only use the default authorization. *i.e.*, "RapidAPI Auth", provided by the platform. In "RapidAPI Auth", the headers named "X-RapidAPI-Host" and "X-RapidAPI-Key" must be sent along with each request when calling an API, as shown in Figure 3. The value of the "X-RapidAPI-Host" (API host) is unique to each API and it is automatically generated by the RapidAPI platform when an API is created. Usually, it combines the API name, an index number to differentiate APIs if any other API with the same name, and the domain "p.rapidapi.com", such as "open-ai21.p.rapidapi.com", where "open-ai" is the API name with the index number "21". The value of the "X-RapidAPI-Key" is a key from API consumers' accounts on the RapidAPI platform to authorize the source of API requests. After providing the necessary "X-RapidAPI-Host" and "X-RapidAPI-Key" values, API consumers can integrate the APIs into their applications and track the key usage in API calls. Since the app key belongs to developers and it is the only certification from developers when calling an API with the basic authorization, leaking such keys could be dangerous. However, we found many developers leak their keys in their code. The threat model of key leaks is discussed in Section 2.3.

4.2 Key Leaks in the Wild

While the key leak is a common issue across various platforms, our work focuses on the RapidAPI platform's key leak problem because of its importance in the RapidAPI platform and the potential serious consequences that arise from such leaks. More details about potential attacks based on leaked keys will be discussed in the following sections. To find the RapidAPI key leaks in the wild, we first searched for keys in code from the GitHub platform, which is one of the world's largest code-sharing platforms. As we mentioned in Section 4.1, when API consumers call the APIs on the RapidAPI platform, the "X-RapidAPI-Key" value and "X-RapidAPI-Host" must be provided in the request headers. Since the "X-RapidAPI-Host" always ends with "p.rapidapi.com", we first searched for the string "p.rapidapi.com" in all GitHub code and downloaded the repositories with such a string. Then, we looked for the corresponding key values in these files. If API consumers copy the code snippets from the RapidAPI platform, the parameter values usually follow the parameter names, such as the API host string "chatgpt52.p.rapidapi.com" follows the "X-RapidAPI-Host" and the key string follows the "X-RapidAPI-Key". This helps us quickly locate the API host and the key that calls the API. After collecting preliminary results and observing the obtained keys, we found that the RapidAPI keys are 50 bits and certain bits are always identical, *e.g.*, the 10th to 13th values are always "msh" and the 36th to 38th values are "jsn". This helps us accordingly identify the keys that do not strictly follow the "X-RapidAPI-Key" string. After considering both the key value length and attributes of keys, all the extracted keys from the code on GitHub are real RapidAPI keys. Compared to previous works that detected various key leaks on the GitHub platform [39, 46], our key identification method achieves a higher accuracy because of the obvious and unique pattern of RapidAPI keys. For each code file, we obtained the API it calls and its key value.

Surprisingly, we found 3,533 leaked keys from 6,495 GitHub repositories that call Rapid APIs. It is possible that some of these leaked keys are used only for debugging and are left in the code. However, we couldn't directly test the leaked keys in adversarial scenarios due to ethical reasons. Fortunately, we could design a way to assess whether a key is still valid without actually exercising the key. We used leaked keys to visit a non-existing endpoint in a free API that we developed. If a key is still valid, it will pass the key validation process and respond "endpoint does not exist". Since free APIs can be called using any valid keys, if it responds "You are not subscribed to this API", the key is no longer valid. Meanwhile, calling such a non-existing endpoint will not succeed, so the API call will not be counted, and the leaked key is not actually used. To avoid any potential impact on the leaked keys, we only used 100 keys for the testing. As a result, we found that 98% of keys are still usable. This indicates that developers who leak their keys usually do not delete their keys from the RapidAPI platform, and these leaked keys can be potentially abused by adversaries for attacks.

Among the 3,533 leaked keys, they are used to call 899 different APIs. For different price plans in APIs (Free plan or Paid plan), adversaries can perform different attacks toward these plans using leaked keys, *i.e.*, (A1) Resource Exhaustion Running Attack (§ 4.3), (A2) Theft of Service Attack (§ 4.4), (A3) Data Manipulation Attack and (A4) User Data Breach Attack (§ 4.5), as shown in Figure 4. The number of leaked keys and the consequent attacks are shown in Table 1. Note that one key can be used to subscribe to and call different APIs. Moreover, we found that there are 151 keys, where each of these keys has been used in several GitHub repositories. There even exists one key being used by 30 repositories owned by different developers. This is possible due to the code copy&paste and the developer's key being copied and used by others inadvertently, potentially resulting in unexpected attacks and financial losses to developers. "Bing News Search" API is the most commonly called API and has been called by 110 applications using different keys. As a free API, adversaries can use different leaked keys to perform the Denial of Service Attack on this API. For APIs that can be potentially attacked using leaked keys, 54% of them have a 9.5 or higher popularity score, showing that they are very popular and many developers will be impacted if these APIs are attacked.

In addition, we discovered key leaks in many Android apps. More details about Android apps downloading and processing will be introduced in Section 6.3. We found 526 Android apps that call APIs from the RapidAPI platform. Among them, 177 keys are leaked in source code and they are used to invoke 134 APIs, as shown in Table 1.

Type of APIs	Type of Attack	# of Leaked Keys	
		GitHub Repos	Android Apps
Free API	(A1) Resource Exhaustion Running Attack	1,055	73
Freemium API	(A1) Resource Exhaustion Running Attack	2,576	133
	(A2) Theft of Service Attack		
Paid API	(A2) Theft of Service Attack	196	23
APIs with Data storage	(A3) Data Manipulation Attack	169	7
	(A4) User Data Breach Attack		
Total		3,533	177

Table 1: Summary of key leaks in the wild.

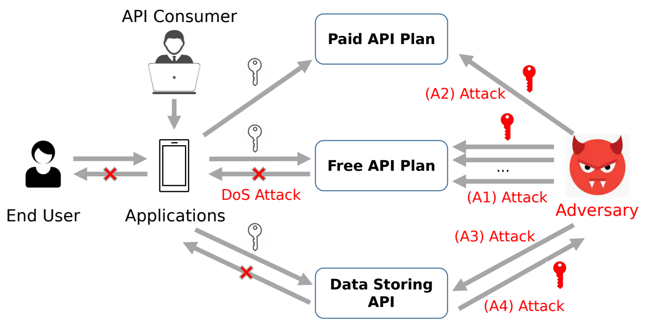


Figure 4: Attacks caused by leaked keys toward different types of APIs.

4.3 Attacks against Free Plan

For Free APIs, the keys from API consumers are limited to 1,000 requests per hour and 500K requests per month. Freemium APIs also have a limited free quota, although API consumers need to bind the credit card information in their accounts before testing. The adversary can first perform **(A1) Resource Exhaustion Running Attack**, which aims to run out of the free quota of the API consumer's key to an API, by sending a substantial number of requests to the API using the leaked key. After triggering the threshold from the platform, the leaked key will be limited and API consumers can't get effective responses from the API. This would subsequently lead to a Denial of Service Attack toward other RapidAPI stakeholders. For example, the API consumer's application may use the key to call the API and provide relevant services. Since the key is restricted to call the API, the application fails to provide expected functionalities, leading to more end users being impacted.

To verify whether the (A1) Resource Exhaustion Running Attack is a realistic threat, we conducted an experiment on a Free API that we developed (we discussed the ethical considerations in Section 2.4). During the testing, we first developed a Free API. Then, one researcher was assumed to be an API consumer and leaked his/her key. Another researcher obtained the leaked key and used the key to send requests to the Free API to launch the attack. After sending 1K requests, new requests using the same key couldn't get useful responses, so the (A1) Resource Exhaustion Running Attack was successful and led to a DoS attack.

In the GitHub code, we found 1,055 leaked keys were used to call Free APIs, which can be used to launch (A1) attack by adversaries. For the other 2,576 leaked keys used to invoke Freemium APIs, if API consumers only subscribe to the basic free plan, the (A1) attack also works. In the Android app code, 73 leaked keys were used to call Free APIs. We found an app named "App Hunt - Explore Apps Store" and it has been downloaded over 1M times. However, it called a free API and leaked its key. If an adversary performs the (A1) attack and DoS attack on the leaked key, the app's functionality could be compromised, affecting all its end users.

4.4 Attacks against Paid Plan

For Freemium and Paid APIs, API consumers can use their keys to subscribe to a paid plan. Table 2 shows the price plan from a Freemium API, which provides three paid plans: Pro, Ultra, and Mega, and each plan has different prices and limits. For APIs with paid plans and limitations, we found the leaked key would lead to different attacks.

Plan	Price	Request	Limit	Attacks
Basic	$0.00	50	Hard Limit	A1
Pro	$3.00	12K	Hard Limit	A1, A2
Ultra	$15.00	200K	+ $0.0003 each other	A2
Mega	$30.00	600K	+ $0.00003 each other	

Table 2: Price plan of a Freemium API (monthly).

If API consumers use a key to subscribe to a paid plan, the adversary can use the key to launch the **(A2) Theft of Service Attack**, which means they use the paid service for free. For example, as shown in Table 2, API consumers can subscribe to up to 600K requests per month. If their keys are leaked, the adversary can send 600K requests to the API freely instead of only 50 basic quotas per month using the Basic plan. What's more, for paid plans, API providers can set a hard limit (Pro Plan) or charge additional fees for each request (Ultra and Mega Plan), as shown in Table 2. In the first case, the adversary can still perform the (A1) Resource Exhaustion Running Attack and Dos Attack. In the latter case, after running out the free quotas, additional charges will accrue on the API consumer's account, resulting in monetary losses for the API consumer.

We discovered that 196 leaked keys in GitHub code and 23 leaked keys in Android apps were used to access Paid APIs. Adversaries can use the keys to launch the (A2) attack. In addition, the other 2,576 keys in GitHub and 133 in Android apps used to invoke Freemium APIs can potentially be exploited to launch the (A2) attack.

4.5 Attacks against API Data

In addition to the attacks toward different price plans, an adversary can use the leaked key to call API endpoints to manipulate the data within an API. We first present the **(A3) Data Manipulation Attack**, which edits API consumers' data, such as removing their data or providing misleading data to cause application crashes or malfunctions. Then we discuss the **(A4) User Data Breach Attack** targeting end users and an adversary can steal user data from APIs.

(A3) Data Manipulation Attack. While lots of APIs are providing different services to API consumers, *e.g.*, providing weather data or news, there are many APIs designed for data handling and processing. Some APIs provide endpoints for API consumers to store and edit data from API consumers or users. For example, we found one API that provides several endpoints for product information storage, *e.g.*, "Get Products", "Create Product", and "Delete Product" endpoints. API consumers can first store products in the API and then call different endpoints to get or edit product information. Similar to a database, such APIs provide the CRUD functions (create, read, update, delete) for API consumers to store or edit data in the API. An adversary can use the leaked keys to call endpoints of the API and edit the API consumer's data, *i.e.*, deleting all the data in an API. As a result, the API consumer's applications cannot get the necessary data, causing potential functional errors.

HTTP Method	Endpoint Name
PATCH	Update A Specified User's Information
DELETE	Delete the Specified User from the Database
POST	Store A User to the Database
GET	Get Specific User
GET	Get All Users

Table 3: Endpoints of an API that can store and edit user data.

(A4) User Data Breach Attack. User profile data can also be stored within an API database. In such cases, a user's personal data can be leaked, edited, or even deleted after the key is leaked. Table 3 shows the endpoints of a real-world API that can store and edit user data. Surprisingly, it provides an endpoint named "Get All Users", which can be used by adversaries to obtain user data easily. After getting the index to all users, adversaries can use the "Get Specific User" endpoint to get a user's detailed information or delete a specific user using the "Delete the Specified User from the Database" endpoint. For other APIs that don't provide full access to all user data, the adversary can also use certain endpoints (such as "GetUserByName") to infer the user name or register a new user account to infer the user index (*e.g.*, user ID) and get access to user data. In addition to the user's profile, other types of user-sensitive data, such as users' previous orders or activities, might also be stolen by the adversary.

We experimented to validate whether (A3) Data Manipulation Attack and (A4) User Data Breach Attack are realistic threats on the RapidAPI platform. We implemented the attack on our own accounts to avoid potential ethical risks to other APIs or users. Our assumption is that if we could use our key to edit the data of an API, the adversary can also use the leaked key to edit our data. First, one researcher used our key to visit the two APIs we mentioned above and created new data, simulating the behaviors of victims. Then, another researcher used the same key (leaked key) and corresponding endpoints to get, update, and delete user data using the same key as the adversaries. We used the endpoints "Get All Users" and "Get Specific User" to get user data details. Then we edited and updated the user data using other endpoints. As a result, we successfully obtained and edited the user data. Our experiments demonstrated that the (A3) Data Manipulation Attack and (A4) User Data Breach Attack are realistic threats using the leaked keys.

To find APIs that store data and provide endpoints for data updates, we check the corresponding HTTP methods of each endpoint of an API, *e.g.*, "POST", "GET" and "DELETE". Normally, APIs provide the "GET" method for API consumers to request and obtain data from an API. If an API has endpoints with "POST" methods, API consumers can submit data to the API server for processing, and thus we consider it an API that can store or edit data. For example, in Table 3, the API has an endpoint named "Store A User to the Database" with the POST method. We found that 846 APIs (4.3% out of the total APIs in our dataset) have endpoints with the "POST" method, in which 223 APIs use additional authorization to protect data better besides the key, as we discussed in Section 4.1. For the rest 623 APIs, we checked whether they are called in GitHub or Android code. Finally, we found 169 leaked keys in the GitHub code and 7 in the Android app that were used to call these vulnerable APIs. Adversaries can use these keys to launch the (A3) Data Manipulation Attack and (A4) User Data Breach Attack.

5 RISKS OF API METADATA ABUSE

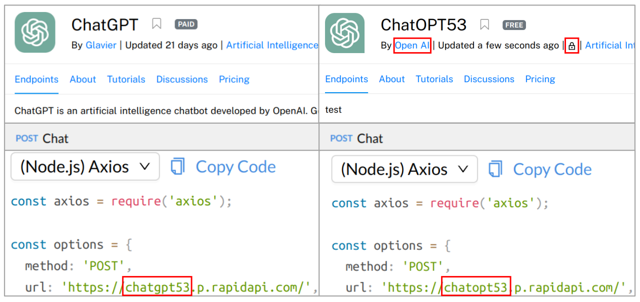
5.1 Typosquatting on Endpoints URLs

As discussed in Section 4.1, the RapidAPI platform assigns a unique host named "X-RapidAPI-Host" to each API. When an API is created, if its name doesn't exist on the RapidAPI platform, the assigned host is the API name. While the RapidAPI platform allows multiple APIs to have the same name, if the API name is already registered on the platform, the platform appends an incrementing index number to the API name within the host to distinguish them. For example, the first "ChatGPT" API will use the host "chatgpt.p.rapidapi.com" and the second "ChatGPT" API will use "chatgpt2.p.rapidapi.com". Each endpoint in an API has a URL defined by API providers and URLs are typically composed of the API host combined with endpoint names or parameters, such as "https://chatgpt.p.rapidapi.com/chat/completions". API consumers call the endpoint URLs to access API functions.

We discovered two weaknesses in the API name and API index number that can be abused and exploited by adversaries to perform the typosquatting attack on the endpoint URLs. The typosquatting attack is also known as URL hijacking and targets innocent users who incorrectly type a website address. When calling an API endpoint URL, API consumers may misspell some characters without noticing them. Since the endpoint URL consists of the API host and endpoint name/parameter, if the misspelled character is in the endpoint name or parameter, the request will be sent to the same API but a different endpoint or fail to find the endpoint. However, if the misspelled character is in the API host, the request will be sent to another API instead of the one that API consumers want. The vulnerabilities appear when two APIs have the same endpoints. In such cases, the only difference between the two endpoint URLs is the API host, which consists of the API name and index number for differentiation. For example, the request to endpoint URL "chatgpt.p.rapidapi.com/{endpoint}" will be sent to "ChatGPT" API while the request to "chatopt.p.rapidapi.com/{endpoint}" goes to "ChatOPT" API.

An adversary can first utilize the weakness in the API name by designing a new API that mimics another API through a slight misspelling in the API name and retains the same endpoint name. For example, the adversary can develop an API named "ChatOPT", which has the host "chatopt.p.rapidapi.com", to attack the API "ChatGPT".

Second, we found that the index number generated by the RapidAPI platform to differentiate APIs has a weakness since the number


ChatGPT PAID
By Glavier | Updated 21 days ago | Artificial Intelligence

Endpoints About Tutorials Discussions Pricing

ChatGPT is an artificial intelligence chatbot developed by OpenAI. G

POST Chat

(Node.js) Axios Copy Code

```
const axios = require('axios');

const options = {
  method: 'POST',
  url: 'https://chatgpt53.p.rapidapi.com/',
```

ChatOPT53 FREE
By Open AI | Updated a few seconds ago | | Artificial Int

Endpoints About Tutorials Discussions Pricing

test

POST Chat

(Node.js) Axios Copy Code

```
const axios = require('axios');

const options = {
  method: 'POST',
  url: 'https://chatopt53.p.rapidapi.com/',
```

Figure 5: A potential typosquatting attack on a popular ChatGPT API. The "ChatOPT53" API is designed for our testing only and the lock means it is not published.

is not assigned for all APIs. Through our extensive testing within our accounts, we found that adversaries cannot control the index number since it is incremental for all APIs, including newly created APIs that are not public on the platform. However, an adversary can utilize the weakness in index number and directly append a number at the end of the malicious API name, *i.e.*, "misspelled API name + index number of another API", such as "ChatOPT53", as shown in Figure 5. Since there doesn't exist any API with the same name, the RapidAPI platform won't add an index number to the API host ("chatopt53.p.rapidapi.com"). This makes the typosquatting attack much easier since the attack can be launched toward a specific targeted API. Note that the malicious API name doesn't influence the attack since the typosquatting attack happens when API consumers unknowingly call the endpoint URL, remaining unaware of the malicious API's existence.

To find potential typosquatting APIs on the current RapidAPI platform, we first scanned and extracted all the API hosts. We used the tool URLCrazy [18], which was designed to generate domain name typos and covers 15 generation modes, *e.g.*, character insertion, deletion, and substitution, to generate different possible typosquatting hosts for each API host. In our testing, we mainly tested single-character differences and the changed character is in the API name instead of the number in the host. Then, we checked whether the generated typosquatting host is used by other APIs. If so, we considered the two hosts as a pair and they can squat each other. We put host pairs together and grouped them based on their similarity. For each group, we identified the host that is used the least as the potential typosquatting attack host. For example, there exist several "ChatGPT" APIs while only one "ChatOPT" API exists, so the API name "ChatOPT" is possibly a typosquatting attack for other APIs.

As a result, we found 53 APIs that possibly use typosquatting to attack other APIs. 204 APIs use the name "Climate Change Live" and their hosts are "climate-change-live + {number}.p.rapid.com". We found other typosquatting APIs use hosts "climate-chamge-live", "clomate-change-live", or "climate-chane-live". The "Amazon Data Scraper" is used by 62 APIs, and other APIs use "Scrapper" or "Scaper" in their API names. Figure 9 in Appendix A shows a possible typosquatting API named "Amazon Data Scapper" on the RapidAPI platform. We also found 4 APIs that add numbers at the end of API names, which possibly exploited the weakness in the API index number we discovered. For example, the API named "Translator 7" has the host "translator-7.p.rapidapi.com" that can attack an

API "Translator" with the host "translator7.p.rapidapi.com". For the APIs with misspelled names, it is possible that API providers made mistakes inadvertently when setting the name for an API since we even found API providers named APIs with a single number ("1") or a RapidAPI key. However, for the weakness in the API index number, since the API name is carefully designed and the number is added intentionally, it is possible that the API is purposely used for typosquatting attacks to increase the chance of being invoked.

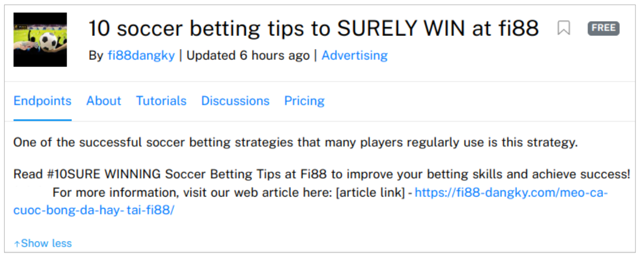
5.2 Issues Caused by Poor Certification

When API providers create an API on the RapidAPI platform, the API is private by default and can't be found by API consumers. This is useful for individuals or teams whose API is still under development or the API is not suitable for a wider audience. After creation, API providers can change the API visibility to the public and allow others to visit the API. Surprisingly, we found the *poor certification vulnerability* exists and the RapidAPI platform doesn't have a strict certification process on the API providers or their published APIs. API providers' information is not validated and they can casually change their names to impersonate other developers or famous companies. API providers can also exploit the vulnerability and arbitrarily publish any APIs on the platform.

5.2.1 Developers Impersonation. The RapidAPI platform allows any developer to register accounts and publish APIs on the platform. On the API listing page, the API provider's name is presented following the API name. API consumers can click to view the API provider's information, *e.g.*, his/her published APIs and following APIs. However, the platform has limited restrictions on the API providers, such as no rigorous validation of their information or their APIs. This allows adversaries to impersonate famous companies or service providers easily, leading to possible phishing attacks on users.

Although the RapidAPI platform provides an attribute named "tag" for all APIs, such as "Official" and "Verified", few APIs have such tags. There are only 221 APIs with the "Official" tag and 219 with the "verified" tag, which is negligible compared to the large number of APIs (40K) on the platform. Developers may not notice the "Official" tag but consider the APIs using known company names and icons as the official APIs. For example, while there are six APIs named "ChatGPT" that provide services about ChatGPT, none of them are official APIs published by the Open AI company. Instead, they are published by various individual developers.

We found two weaknesses of developer settings that adversaries can exploit to impersonate other developers. First, API providers can arbitrarily change their names and icons without triggering any validation so that adversaries can change their account name to a company name. For example, as shown in Figure 5, we can change our developer name to "Open AI" and publish APIs without triggering any trademark infringements from the platform. Then, users almost can't find such fraud (note that our API was created for testing and not published on the platform). It becomes more serious when such an API asks for user-sensitive data, such as asking for a user's account in the "ChatGPT" API. Once users consider an API as an official API, they may tend to provide their sensitive data, leading to consequent losses to users [43]. Second, the RapidAPI allows duplicate developer names so that several developers can use


10 soccer betting tips to SURELY WIN at fi88 FREE

By fi88dangky | Updated 6 hours ago | Advertising

Endpoints About Tutorials Discussions Pricing

One of the successful soccer betting strategies that many players regularly use is this strategy.

Read #10SURE WINNING Soccer Betting Tips at Fi88 to improve your betting skills and achieve success!
For more information, visit our web article here: [article link] - https://fi88-dangky.com/meo-ca-cuoc-bong-da-hay-tai-fi88/

↑Show less

Figure 6: A spamming API on the RapidAPI platform.

the same name. Such a setting can mislead users and users might think several APIs are developed by the same developer unless users check each developer's detailed webpage separately. We found that 81 names are shared by 175 accounts. For example, the name "Alex" is used by 6 developers and they developed 9 different APIs.

5.2.2 Spamming APIs. During our data collection, we found that the *poor certification vulnerability* has been explored by API providers and a significant number of APIs are not real APIs designed for developers. Instead, they are "spamming APIs" for promotions and direct API consumers to their product websites, such as gambling and property-selling websites. Figure 6 shows the details of a spamming API after translation (the original API is non-English) and the API is about gambling. The API doesn't have any endpoint to be called by API consumers but provides a link to their website in its description. After checking several spamming APIs, we found they typically have two characteristics: they don't provide any useful endpoint and contain links in their descriptions. We used the two filters to check each API and identified spamming APIs. Surprisingly, we found that 12,452 (39%) out of 32,089 APIs are spamming APIs, showing that spamming APIs are prevalent on the RapidAPI platform (as of June 2023). Among these spamming APIs, 94% of them were published within four months and almost all of such APIs were Free APIs. After checking the publishing time of spamming APIs, we found that 43 out of 100 newly published APIs were spamming APIs and 98 out of 198 APIs published within a single day were spamming APIs on the RapidAPI platform at the time of our data collection, as shown in Figure 10 in Appendix A. This indicates that the certification vulnerability has been exploited and used by advertisement companies and such behavior has become more serious as of June 2023. Due to the lack of certification toward APIs and API providers, API providers could publish lots of spamming APIs without getting any punishment. For example, there exists one developer who published over 400 APIs, and all of them are spamming APIs. We removed the spamming APIs from our dataset and did not use them in all of our analyses.

In addition to detecting spamming APIs using endpoints and descriptions, such APIs can also be potentially identified with the developer following network. The RapidAPI platform allows developers to follow an API or developer. However, we find that developers who publish a large number of spamming APIs prefer not to follow any other developer accounts and these spamming APIs normally have no followers, leading to spamming APIs and developers isolated in the following network. More details about developer network analysis can be found in Appendix B.

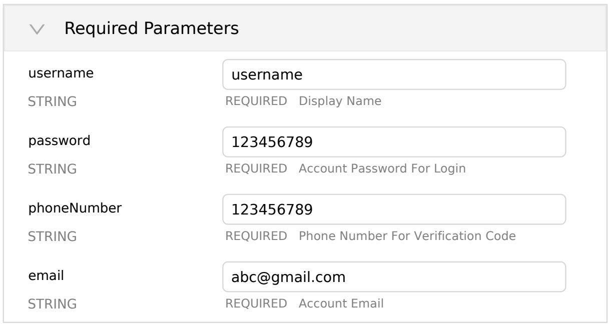
We found the spamming APIs emerging from 2023 and we reported our findings to the RapidAPI platform in October 2023, when the spamming APIs still existed. As of February 2024, we found that the RapidAPI platform has removed almost all spamming APIs, possibly due to our reporting. The platform removed all the APIs that don't provide an endpoint, which is similar to our filter. We randomly selected 100 spamming APIs we identified in 2023 and found that 99 of them had been removed.

6 PRIVACY NON-COMPLIANCE OF RAPID APIS

APIs may collect user information for functionalities, *e.g.*, using user location to find nearby restaurants or sending emails to the user account. To protect users and data privacy, API providers should carefully use such data and provide a privacy policy to allow API consumers and users to make informed decisions. Existing works show that many open application platforms suffer from privacy non-compliance issues, *e.g.*, the Android platform [25], Google Chrome extensions [23], mobile mini-programs [37], Amazon Alexa platform [33], and VR platform [49]. (importance of privacy compliance to platform, developers, and users.) In this section, we aim to discover the privacy non-compliance of APIs and applications that call APIs. We first identify APIs that collect data from users and involve sensitive data collection behaviors. Second, we check whether they have privacy non-compliance issues, such as whether a valid privacy policy link is provided and whether data collection is fully disclosed in the privacy policy. If not, we consider an API to have privacy non-compliance issues. Third, we check whether the privacy issues in APIs could propagate to downstream applications, *e.g.*, Android apps, and subsequently lead to privacy non-compliance issues.

6.1 Sensitive Data Collection Behaviors

If an API needs to collect user data for its functionalities, it can define data collection parameters within an endpoint and API consumers can provide user data as parameters in API requests. Figure 7 shows an API that collects several types of data and each data is stored in a parameter. In addition to the parameter name and the data type (such as number, string), API providers can also provide an example value and description for a parameter so that API consumers can follow the parameter format. In our work, we define the data collection APIs as APIs that collect any sensitive data within the scope of 14 types of PII (personally identifiable information) from a NIST (National Institute of Standards and Technology) report [38]. We also expand more derivative words that are commonly used, *e.g.*, "first name", to the word list. For each API, we first check whether any parameter name in an endpoint is related to data collection keywords shown in Table 4. If so, we consider the parameter as a possible data collection parameter. However, such a parameter is not necessary to collect data from human beings, *i.e.*, the parameter "name" can be used to get the product name and the parameter "address" may belong to a restaurant. To remove such cases, we check the descriptions of API, endpoint, and parameter to confirm if the data collection is related to users. If any data type follows the word "your" semantically in descriptions, we consider the parameter, endpoint and API collecting user data. For a data collection API, we check whether its privacy policy discloses the data collection behaviors and more details will be discussed in Section 6.2.


Required Parameters

username STRING	username REQUIRED Display Name
password STRING	123456789 REQUIRED Account Password For Login
phoneNumber STRING	123456789 REQUIRED Phone Number For Verification Code
email STRING	abc@gmail.com REQUIRED Account Email

Figure 7: An example of data collection parameters within an endpoint of an API.

Address, Name, Email, Account, Location, Phone number, Passport number, Driver license number, Bank account number, Debit card number, Credit card number, Taxpayer identification number, Social Security number (SSN), Vehicle identification number (VIN)

Table 4: Keywords related to personal data collection.

In addition, we found that there exist APIs collecting extremely sensitive data from API consumers or end users, *i.e.*, passwords or keys. If an API collects passwords from end users, both the API provider and API consumers can access the passwords from end users, posing serious privacy risks for the users. To mitigate these risks, API providers should use the account linking or authorization method provided by the RapidAPI platform, as shown in Figure 3 in Section 4.1, instead of asking for a password as a parameter. Another particular case for sensitive data collection is that official APIs, such as the official "Twitter" API, may ask for user passwords or keys for specific functionalities and we don't consider such data collection behaviors as violating privacy compliance. However, as mentioned in Section 5.2.1, almost all APIs on the RapidAPI platform don't have the verified "Official" tag. Therefore, we check whether the API name, developer name, and API description are consistent to validate whether an API is potentially published by an official developer. For example, if a developer "Twitter" developed an API named "Twitter" and described the Twitter functions in the description, we consider it as a potential official API. After excluding such potential official APIs, we found that 231 APIs ask for passwords through parameters in endpoints.

We also found 1,067 APIs asking for different types of keys from API consumers. Interestingly, 17 APIs ask for the RapidAPI keys in parameters. However, as mentioned in Section 4.1, the RapidAPI keys are designed to authorize the connection between API consumers and the RapidAPI platform. These keys should be sent in request headers, and only the RapidAPI platform can receive and validate the keys. Therefore, it is confusing why API providers ask for RapidAPI keys from API consumers and such behaviors may lead to key leaks, as discussed in Section 4. 97 potential official APIs ask for keys and we consider them normal behaviors. However, unofficial APIs also ask for the keys of other platforms. For example, 326 "Amazon Data Scraper" APIs ask for Amazon's API keys to perform the data collection task. It is unclear whether these APIs provided by third-party developers are credible, and thus providing keys to these APIs can be dangerous. In other APIs that collect different keys, 915 APIs don't provide necessary information about the expected source of keys, such as an example or description, which confuses developers about what key they should provide. For APIs that ask for keys, we also check their privacy compliance.

6.2 Privacy Non-Compliance in Rapid APIs

In Section 6.1, we detected the data collection APIs and revealed that thousands of APIs ask for extremely sensitive data. For APIs that collect data, their providers should provide a privacy policy document to disclose how the data is collected, used, and shared. On the one hand, since the API is a black box and API consumers can't obtain the inner details within an API, the privacy policy is the only way that API consumers can know how the collected data is used and shared. Moreover, if API consumers don't know the data usage in APIs, they can't explain such data collection behaviors to end-users in their applications' privacy policies. On the other hand, the data collection disclosure is required in certain regions by legal and lawful regulations, *e.g.*, General Data Protection Regulation (GDPR) [10], CalOPPA (California Online Privacy Protection Act) [7], CCPA (California Consumer Privacy Act) [6], COPPA (Children's Online Privacy Protection Act) [8], and HIPAA (Health Insurance Portability and Accountability Act) [13]. If an API collects data but doesn't provide a complete privacy policy that fully discloses data collection behaviors, the platform might be fined by the government. For example, Google was fined €50 million by the French government in 2019 because of failing to provide complete privacy policies that comply with the GDPR [11].

However, unlike other app platforms, *e.g.*, Apple App Store, Google Play, Amazon Alexa Skill Store, and Google Chrome Web Store, which all present each app's privacy policy link on the app's webpage, the RapidAPI platform doesn't have an area on API listing page to present the privacy policy. Furthermore, the RapidAPI platform doesn't mandate API providers to provide a privacy policy when publishing an API to the public, no matter whether it collects data or not. Instead, API providers can provide a link to the "Product Website", where a privacy policy can be included. Such a setting would undoubtedly increase the difficulty for users to access the privacy policy. Since the product website is not mandatory, only 10,444 of 19,637 APIs (53%) provide a product website in our dataset. For product website links, 1,615 APIs share a link with other APIs, showing that API providers provide such a website for a list of products instead of a unique API. The "News&Media" category has the highest percentage of APIs with a product website (84%). On the contrary, the "Database", "Storage", and "Email" categories, which may collect and process user data, have a lower percentage.

We identified multiple issues on the product websites of APIs, indicating that the websites are not validated by the platform. 71 APIs provide a localhost domain link, such as "http://192.168.10.62:8088/". Another interesting issue is that when API providers submit the website, the RapidAPI platform prefixes URLs with "https://rapidapi.com/" instead of "https://", which leads to 524 APIs providing a non-functional website like "https://rapidapi.com/www.google.com". Figure 8 shows an API that provides such a website. In total, we found that 6,374 product websites (61%) of APIs are not accessible. We also found that most websites are not relevant to APIs themselves but company websites. Such websites don't provide any details about APIs, and thus the privacy policies in these websites are normally not relevant to the APIs.

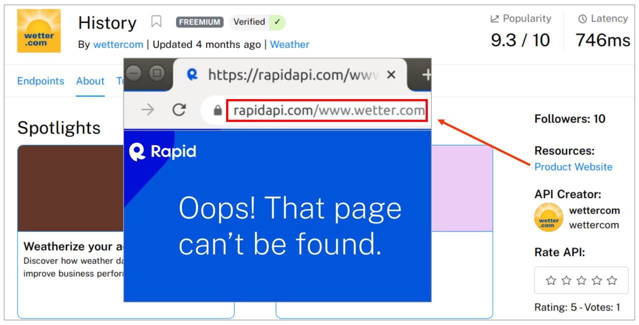


Figure 8: An API that provides a broken product website.

We found that a total of 1,709 APIs contain data collection behaviors. Among them, 498 APIs don't provide the product websites and 674 APIs provide a website without a privacy policy link. In addition, 183 APIs provide a broken privacy policy link. There are only 354 APIs (21%) that provide an effective privacy policy link on product websites. For these APIs, we performed an analysis of their privacy policy content. We applied the PoliCheck [21], a commonly used privacy policy analysis tool, to analyze sentences in privacy policies and extract collected data types. If an API's data collection behavior is not mentioned in any sentences within its privacy policy, we consider it an incomplete privacy policy. We found that 251 APIs have an incomplete privacy policy that doesn't fully disclose their data collection behaviors. As a result, only 103 out of 1,709 APIs (6%) collect data and provide a complete privacy policy. The different issues of API product websites and privacy policies are summarized in Table 5.

Privacy Policies of APIs	# of APIs	Percentage
APIs with data collection behaviors	1,709	-
Don't provide a product website	498	29%
Provide a product website without a privacy policy link	674	39%
Provide a broken privacy policy link	183	11%
Provide an incomplete privacy policy	251	15%
Provide a complete privacy policy	103	6%

Table 5: Privacy policy issues of data collection APIs.

6.3 Privacy Non-Compliance in Applications Calling Rapid APIs

Since the RapidAPI platform is designed for API consumers to accelerate their app development during their work, we are interested in how API consumers use these APIs in their applications and whether these applications have any privacy issues. As we discussed in Section 6.2, privacy issues in APIs could propagate to downstream applications within the software supply chain and cause difficulties for downstream app developers in providing a complete privacy disclosure, subsequently leading to privacy non-compliance issues. In this section, we use Android apps to demonstrate the potential consequences caused by Rapid APIs' privacy issues. We selected the Android apps on the Google Play platform for our testing, which is one of the most popular app stores.

To analyze Android apps, we first obtain an Android app list from the AndroZoo dataset [20], which contains over 23 million Android apps collected since 2011. However, due to the substantial time required for downloading and processing such a huge number of apps, we randomly sampled 1 million apps for our analysis. We downloaded the APK file of each app using the link provided by Androzoo. To decode the APK files and obtain the decompiled code of Android apps, we used the Apktool [4], which is for reverse engineering Android APK files. At last, we used the same method in Section 4.2 to detect the API hosts and keys in the code. We identified the API hosts and endpoints in the decompiled code and then checked whether the parameters in the endpoints collect sensitive data. To find the privacy non-compliance issues in Android code, we visited the websites of each Android app on Google Play and downloaded their privacy policies. In addition, Google Play added a section named "Data Safety" for each app since 2022 and developers can provide the shared and collected data in this section. If no data is collected or shared, developers can claim that "this app doesn't collect/share user data" in this section. Since Google states such a section is to "provide developers a way to show users if and how they collect, share, and protect user data", we consider it the same as a privacy policy and check it with the privacy policy together. For each app, if its data collection is not mentioned in the privacy policy or data safety section, we consider it to have potential privacy non-compliance issues and an incomplete privacy policy. We still used the PoliCheck for Android privacy policy analysis and detected data collection disclosure in privacy policies.

As a result, we found 526 out of 1 million Android apps calling 316 APIs on the RapidAPI platform. The key leaks in Android code have been discussed in Section 4. As for data collection behaviors, we found 54 apps calling 11 APIs that contain data collection behaviors. Most apps collect name information and the others collect email or location. Among them, 16 apps lack a privacy policy, 11 apps have an incomplete privacy policy, and only 27 apps (50%) have a complete privacy policy. Out of the 16 apps that lack a privacy policy, 5 apps also don't provide a data safety section, although the privacy policy and data safety section are mandatory on the Google Play platform. The remaining 11 apps claim that they don't collect data in the data safety section, which can mislead users about the actual data collection behaviors in apps. For example, the "TokApi - mobile version" API collects user names but doesn't provide a privacy policy link within the company webpage. Consequently, the Android app "TikMate Video Downloader" calls the API but lacks a privacy policy and falsely claims that no data is collected or shared in the data safety section. In such a case, the API's failure to disclose its data collection results in the propagation of false disclosure to the downstream Android app. For the 11 apps that have an incomplete privacy policy, they don't mention the collected data or only mention the "personally identifiable information" (PII) instead of detailed data types in their privacy policies. We also found that none of the apps that call data collection APIs mention that the data will be shared with third-party APIs. However, they share user data with APIs from the RapidAPI platforms. Considering that some of these apps have over 1 million downloads, the privacy non-compliance risk in these apps, *i.e.*, undisclosed data collection and data sharing, may impact millions of users.

7 DISCUSSION

7.1 Investigation of Other Open API Platforms

To understand whether the risks and issues observed in the RapidAPI platform also exist in other open API platforms, we checked various open API platforms and their APIs. However, we found that several large API platforms, such as OpenAPIHub, API Store, and API guru, are designed only for API searching and developers still need to visit each API's website to get the API usage and key.

We discovered two platforms, Apidog [3] and apideck [2], that have a working model similar to the RapidAPI platform, and we compared them with the RapidAPI platform using our methodology and tools. The comparison results are shown in the Table 6. The Apidog platform has 1,806 APIs and the apideck has only 176 APIs, which are both limited compared to the RapidAPI platform. We didn't find any leaked keys in the GitHub code for these two platforms, possibly because very few developers use these platforms. On the other hand, the two platforms could have a strict review process for the limited number of APIs, which significantly improves the quality of APIs and prevents spamming APIs from being published. For API metadata, both platforms don't present the API provider information. However, we found 272 duplicate APIs (out of 1,806) on the Apidog platform and API consumers might find it hard to distinguish them when API provider information is not provided. We didn't find the typosquatting towards endpoints in the two platforms. Lastly, on both platforms, there exist APIs that collect data for functionalities. The Apidog platform doesn't provide company websites or privacy policies for APIs, while the apideck platform provides the company website for every API and users are able to find privacy policies within the website, similar to the RapidAPI platform.

When comparing the RapidAPI platform with the other two open API platforms, we found that they have different operational mechanisms. Although the RapidAPI platform has more APIs and is more famous, which attracts more developers to engage in their ecosystem, it also leads to malicious developers abusing API metadata. This demonstrates the importance of the certification process. While the RapidAPI platform doesn't have a strict vetting system, they are checking the API quality and removing spamming APIs now. The increased developer number also results in more unprofessional and untrained developers leaking their keys in the wild. APIs with data collection exist on all three API platforms, but the Apidog platform doesn't provide any information related to the API providers, such as the developer names or company websites. This makes it hard for API consumers to know how collected data is used.

	RapidAPI	Apidog	apideck
API number	32,089	1,806	176
Key leaks in wild	✓	✗	✗
API metadata abuse	✓	✗	✗
API data collection	✓	✓	✓
Company website	53%	0%	100%

Table 6: Comparison between three open API platforms.

7.2 Mitigation

Based on our measurements and findings, we provide the following recommendations to mitigate the issues and vulnerabilities in the RapidAPI platform as well as other open API platforms.

API Vetting Process. Since the RapidAPI platform allows any user to arbitrarily publish any API to the API hub, it is almost inevitable that low-quality APIs, such as spamming APIs or APIs with low success rates, will be published. Similar to how we found spamming APIs in Section 5.2.2, the certification process can find problematic APIs by checking the API metadata, *e.g.*, API name, description, and endpoint, similar to how they removed spamming APIs. The platform can also scan APIs periodically or deploy automated testing tools to invoke APIs and identify potential low-quality APIs, notifying API providers to enhance and republish them again. For API providers who publish an excess of low-quality APIs, the platform could also monitor and validate whether they are actual benign developers.

Protection of Keys. The RapidAPI platform and developers could work together to protect keys better. The RapidAPI platform can generate keys in different formats, such as adding an encryption process for the keys, to increase the difficulty of searching for keys in the wild. The platform can also use environment variables to store API keys in configuration files instead of hard-coding them in code. We have demonstrated that most leaked keys are still useful, motivating the platform to implement expiration dates for keys. For developers calling RapidAPIs, they may employ obfuscated or encrypted keys in their code or read keys from setting files.

Requirements on Privacy Policy. The RapidAPI platform doesn't mandate an API to provide a privacy policy regardless of its data collection behaviors. The platform also doesn't provide an area for displaying it on the API listing page. Such an overlook may result in potential violations of lawful regulations, *e.g.*, CalOPPA, CCPA, COPPA, HIPAA, and GDPR, leading to penalties from authorities and propagating privacy issues to downstream applications. We suggest the platform to add more requirements to API providers about privacy policy and set up an area for presenting it on the API listing page, similar to what the Google Play platform does.

7.3 Limitation

Our work is the first systematic work to investigate the issues and vulnerabilities in the RapidAPI platform. However, several limitations still exist that can be improved in our future work. First, we didn't analyze all the APIs on the RapidAPI platform since we could only collect part of the APIs (32,089 APIs) while the RapidAPI platform claims that they have over 40K APIs. Second, we only checked the API usage in Android apps. The RapidAPI platform is one of the largest API platforms and APIs may be used in applications from other platforms, such as desktop programs. However, such programs are not published on a marketplace, making privacy policy analysis and privacy compliance checking harder. We plan to collect more programs from other platforms and find their privacy issues in the future. Third, our method for API data collection detection and privacy policy analysis can be improved by applying more advanced machine-learning or deep-learning methods. As our future work, we also plan to design dynamic testing tools to automatically test and analyze APIs' actual behaviors.

8 RELATED WORK

To our knowledge, our work is the first of its kind to systematically analyze and evaluate the issues on an open API platform. Our methodology can also be applied to other emerging open API platforms for discovering security and privacy risks. In previous works, researchers only used APIs on the RapidAPI platform to perform certain tasks. Qin *et al.* [42] used RapidAPI as an instruction tuning dataset to facilitate the ability of large language models to generate API instructions. Du *et al.* [26] utilized over 16,000 APIs to help design a large language model agent to revolutionize the utilization of a vast array of tools for addressing user queries. Nikam *et al.* [41] applied a COVID-related API to build an Android chatbot.

Researchers have investigated on privacy issues of third-party applications on many other open platforms, *e.g.*, IOS apps [27, 50], Android apps [22, 45], Chrome extensions [19, 31, 35], and Amazon Alexa skills [24, 30, 52]. Chia *et al.* [25] did a large-scale study on Android application permissions. Carlini *et al.* [23] performed an evaluation of Google Chrome extension security architecture. Lu *et al.* [37] revealed the risks in Mobile App-in-App ecosystems. Lentzsch *et al.* [33] analyzed the vulnerabilities and privacy non-compliance in the Amazon Alexa voice-app ecosystem. Trimananda *et al.* [49] proposed a method to analyze the privacy issues on the virtual reality (VR) platform and apps. Zuo *et al.* [55] exploited the data leaks in the cloud from mobile apps and found the services of 15,098 apps were subject to data leakage attacks. In addition, researchers mined key leakages in the wild, such as GitHub [39, 46]. Sinha *et al.* [46] detected and mitigated the secret key leaks in GitHub source code repositories. Zhang *et al.* [54] demonstrated 40,880 key leaks in WeChat mini-programs and they may lead to diverse attacks. Researchers also exploited the risks and vulnerabilities about typosuqatting [29, 32, 40]. Liu *et al.* [36] explored the typosquatting attack in container registries. Szurdi *et al.* [48] performed a comprehensive study of typosquatting domain registrations. Additionally, API misuse also commonly appears in different applications [44, 47, 53]. Xu *et al.* [51] analyzed the deep learning APIs on smartphone apps. Li *et al.* [34] employed an active learning algorithm to rank API usage and help programmers find API misuses. However, none of these works addressed the security and privacy risks on the RapidAPI platform.

9 CONCLUSION

In this work, we performed the first large-scale analysis of 32,089 APIs collected from the RapidAPI platform, which is one of the largest open API platforms. We revealed the importance of RapidAPI keys in the API authorization process and identified 3,533 leaked keys in the wild, which can be used by adversaries to launch various attacks. We also discovered the risks of API metadata abuse. Due to the lack of a strict certification system, adversaries can perform typosquatting attacks on APIs, impersonate other developers, or publish spamming APIs. Finally, we investigated the privacy non-compliance issues in APIs and applications that call APIs with data collection. We found that privacy issues in APIs could propagate to downstream applications, e.g., Android apps, and subsequently lead to privacy non-compliance issues. Based on our findings, we also provide mitigations to the platform to reduce the vulnerabilities of the platform.

REFERENCES

[1] Alexa Policy Requirements. https://developer.amazon.com/fr-FR/docs/alexa/custom-skills/policy-requirements-for-an-alexa-skill.html.

[2] apideck. https://www.apideck.com/.

[3] Apidog. https://apidog.com/.

[4] Apktool. https://apktool.org/.

[5] App Store Review Guidelines. https://developer.apple.com/app-store/review/guidelines/.

[6] California Consumer Privacy Act (CCPA). https://oag.ca.gov/privacy/ccpa.

[7] California Online Privacy Protection Act (CalOPPA). https://consumercal.org/about-cfc/cfc-education-foundation/california-online-privacy-protection-act-caloppa-3/.

[8] Children's Online Privacy Protection Rule (COPPA). https://www.ftc.gov/legal-library/browse/rules/childrens-online-privacy-protection-rule-coppa.

[9] Chrome Program Policies. https://developer.chrome.com/docs/webstore /program-policies/#: :text=Extensions

[10] General Data Protection Regulation. https://gdpr-info.eu.

[11] Google fined €50 million for GDPR violation in France. https://www.theverge.com/2019/1/21/18191591/google-gdpr-fine-50-million-euros-data-consent-cnil.

[12] Google Play Developer Policy Center. https://play.google.com/about/developer-content-policy/.

[13] Health Insurance Portability and Accountability Act of 1996 (HIPAA). https://www.cdc.gov/phlp/publications/topic/hipaa.html.

[14] Open API Market Size is projected to reach USD 13.21 Billion by 2030, growing at a CAGR of 23.83%: Straits Research. https://www.globenewswire.com/en/news-release/2022/08/18/2501038/0/en/Open-API-Market-Size-is-projected-to-reach-USD-13-21-Billion-by-2030-growing-at-a-CAGR-of-23-83-Straits-Research.html.

[15] OpenAPIHub. https://www.openapihub.com/en-us/.

[16] ProgrammableWeb. https://www.mulesoft.com/ programmableweb.

[17] Rapid. https://rapidapi.com/.

[18] URLCrazy. https://www.morningstarsecurity.com/ research/urlcrazy.

[19] Shubham Agarwal. Helping or hindering? how browser extensions undermine security. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 23–37, 2022.

[20] Kevin Allix, Tegawendé F Bissyandé, Jacques Klein, and Yves Le Traon. Androzoo: Collecting millions of android apps for the research community. In *Proceedings of the 13th international conference on mining software repositories*, pages 468–471, 2016.

[21] Benjamin Andow, Samin Yaseer Mahmud, Justin Whitaker, William Enck, Bradley Reaves, Kapil Singh, and Serge Egelman. Actions speak louder than words:{Entity-Sensitive} privacy policy and data flow analysis with {PoliCheck}. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 985–1002, 2020.

[22] Sven Bugiel, Stephen Heuser, and Ahmad-Reza Sadeghi. Flexible and fine-grained mandatory access control on android for diverse security and privacy policies. In *22nd USENIX Security Symposium (USENIX Security 13)*, pages 131–146, 2013.

[23] Nicholas Carlini, Adrienne Porter Felt, and David Wagner. An evaluation of the google chrome extension security architecture. In *21st USENIX Security Symposium (USENIX Security 12)*, pages 97–111, 2012.

[24] Long Cheng, Christin Wilson, Song Liao, Jeffrey Young, Daniel Dong, and Hongxin Hu. Dangerous skills got certified: Measuring the trustworthiness of skill certification in voice personal assistant platforms. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 1699–1716, 2020.

[25] Pern Hui Chia, Yusuke Yamamoto, and N Asokan. Is this app safe? a large scale study on application permissions and risk signals. In *Proceedings of the 21st international conference on World Wide Web*, pages 311–320, 2012.

[26] Yu Du, Fangyun Wei, and Hongyang Zhang. Anytool: Self-reflective, hierarchical agents for large-scale api calls. *arXiv preprint arXiv:2402.04253*, 2024.

[27] Manuel Egele, Christopher Kruegel, Engin Kirda, and Giovanni Vigna. Pios: Detecting privacy leaks in ios applications. In *NDSS*, pages 177–183, 2011.

[28] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. University of California, Irvine, 2000.

[29] Mohammad Taha Khan, Xiang Huo, Zhou Li, and Chris Kanich. Every second counts: Quantifying the negative externalities of cybercrime via typosquatting. In *2015 IEEE Symposium on Security and Privacy*, pages 135–150. IEEE, 2015.

[30] Deepak Kumar, Riccardo Paccagnella, Paul Murley, Eric Hennenfent, Joshua Mason, Adam Bates, and Michael Bailey. Skill squatting attacks on amazon alexa. In *27th USENIX security symposium (USENIX Security 18)*, pages 33–47, 2018.

[31] Pierre Laperdrix, Oleksii Starov, Quan Chen, Alexandros Kapravelos, and Nick Nikiforakis. Fingerprinting in style: Detecting browser extensions via injected style sheets. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2507–2524, 2021.

[32] Victor Le Pochat, Tom Van Goethem, and Wouter Joosen. A smörgåsbord of typos: exploring international keyboard layout typosquatting. In *2019 IEEE Security and Privacy Workshops (SPW)*, pages 187–192. IEEE, 2019.

[33] Christopher Lentzsch, Sheel Jayesh Shah, Benjamin Andow, Martin Degeling, Anupam Das, and William Enck. Hey alexa, is this skill safe?: Taking a closer look at the alexa skill ecosystem. *Network and Distributed Systems Security (NDSS) Symposium2021*, 2021.

[34] Ziyang Li, Aravind Machiry, Binghong Chen, Mayur Naik, Ke Wang, and Le Song. Arbitrar: User-guided api misuse detection. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1400–1415. IEEE, 2021.

[35] Yuxi Ling, Kailong Wang, Guangdong Bai, Haoyu Wang, and Jin Song Dong. Are they toeing the line? diagnosing privacy compliance violations among browser extensions. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, pages 1–12, 2022.

[36] Guannan Liu, Xing Gao, Haining Wang, and Kun Sun. Exploring the unchartered space of container registry typosquatting. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 35–51, 2022.

[37] Haoran Lu, Luyi Xing, Yue Xiao, Yifan Zhang, Xiaojing Liao, XiaoFeng Wang, and Xueqiang Wang. Demystifying resource management risks in emerging mobile app-in-app ecosystems. In *Proceedings of the 2020 ACM SIGSAC conference on computer and communications Security*, pages 569–585, 2020.

[38] Erika McCallister. *Guide to protecting the confidentiality of personally identifiable information*, volume 800. Diane Publishing, 2010.

[39] Michael Meli, Matthew R McNiece, and Bradley Reaves. How bad can it git? characterizing secret leakage in public github repositories. In *NDSS*, 2019.

[40] Shradha Neupane, Grant Holmes, Elizabeth Wyss, Drew Davidson, and Lorenzo De Carli. Beyond typosquatting: An in-depth look at package confusion. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 3439–3456, 2023.

[41] Swati Nikam, Digvijay Lakade, Aakash Ahire, Jayesh Somwanshi, and Rushikesh Late. Covid-19 android chatbot using rasa. In *2022 3rd International Conference for Emerging Technology (INCET)*, pages 1–7. IEEE, 2022.

[42] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*, 2023.

[43] Erwin Quiring, Alwin Maier, and Konrad Rieck. Misleading authorship attribution of source code using adversarial learning. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 479–496, 2019.

[44] Xiaoxue Ren, Xinyuan Ye, Zhenchang Xing, Xin Xia, Xiwei Xu, Liming Zhu, and Jianling Sun. Api-misuse detection driven by fine-grained api-constraint knowledge graph. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, pages 461–472, 2020.

[45] Gulshan Shrivastava, Prabhat Kumar, Deepak Gupta, and Joel JPC Rodrigues. Privacy issues of android application permissions: A literature review. *Transactions on Emerging Telecommunications Technologies*, 31(12):e3773, 2020.

[46] Vibha Singhal Sinha, Diptikalyan Saha, Pankaj Dhoolia, Rohan Padhye, and Senthil Mani. Detecting and mitigating secret-key leaks in source code repositories. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, pages 396–400. IEEE, 2015.

[47] Amann Sven, Hoan Anh Nguyen, Sarah Nadi, Tien N Nguyen, and Mira Mezini. Investigating next steps in static api-misuse detection. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pages 265–275. IEEE, 2019.

[48] Janos Szurdi, Balazs Kocso, Gabor Cseh, Jonathan Spring, Mark Felegyhazi, and Chris Kanich. The long {"Taile"} of typosquatting domain names. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 191–206, 2014.

[49] Rahmadi Trimananda, Hieu Le, Hao Cui, Janice Tran Ho, Anastasia Shuba, and Athina Markopoulou. {OVRseen}: Auditing network traffic and privacy policies in oculus {VR}. In *31st USENIX security symposium (USENIX security 22)*, pages 3789–3806, 2022.

[50] Yue Xiao, Zhengyi Li, Yue Qin, Xiaolong Bai, Jiale Guan, Xiaojing Liao, and Luyi Xing. Lalaine: Measuring and characterizing {Non-Compliance} of apple privacy labels. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 1091–1108, 2023.

[51] Mengwei Xu, Jiawei Liu, Yuanqiang Liu, Felix Xiaozhu Lin, Yunxin Liu, and Xuanzhe Liu. A first look at deep learning apps on smartphones. In *The World Wide Web Conference*, pages 2125–2136, 2019.

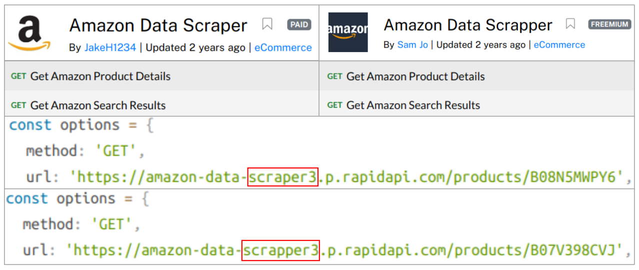
[52] Jeffrey Young, Song Liao, Long Cheng, Hongxin Hu, and Huixing Deng. {SkillDetective}: Automated {Policy-Violation} detection of voice assistant applications in the wild. In *31st USENIX Security Symposium (USENIX Security 22)*, 2022.

[53] Ying Zhang, Md Mahir Asef Kabir, Ya Xiao, Danfeng Yao, and Na Meng. Automatic detection of java cryptographic api misuses: Are we there yet? *IEEE Transactions on Software Engineering*, 49(1):288–303, 2022.

[54] Yue Zhang, Yuqing Yang, and Zhiqiang Lin. Don't leak your keys: Understanding, measuring, and exploiting the appsecret leaks in mini-programs. *arXiv preprint arXiv:2306.08151*, 2023.

[55] Chaoshun Zuo, Zhiqiang Lin, and Yinqian Zhang. Why does your data leak? uncovering the data leakage in cloud from mobile apps. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1296–1310. IEEE, 2019.

A EXAMPLE APIS WITH METADATA ABUSE



Amazon Data Scraper PAID
By JakeH1234 | Updated 2 years ago | eCommerce

Amazon Data Scrapper FREEMIUM
By Sam Jo | Updated 2 years ago | eCommerce

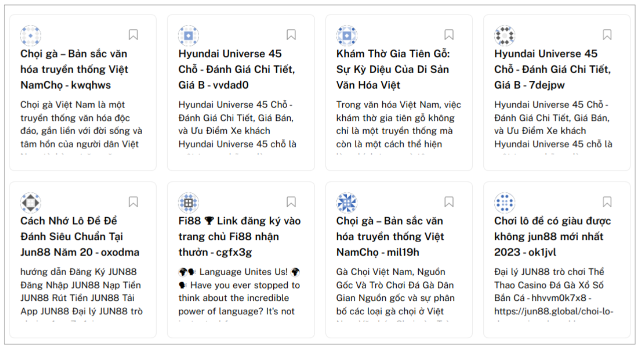
GET Get Amazon Product Details
GET Get Amazon Search Results

GET Get Amazon Product Details
GET Get Amazon Search Results

```
const options = {
  method: 'GET',
  url: 'https://amazon-data-scraper3.p.rapidapi.com/products/B08N5MWPY6',
```

```
const options = {
  method: 'GET',
  url: 'https://amazon-data-scrapper3.p.rapidapi.com/products/B07V398CVJ',
```

Figure 9: A possible typosquatting API named "Amazon Data Scrapper" on the RapidAPI platform.



Chọi gà – Bản sắc văn hóa truyền thống Việt NamChọ - kwqhws
Chọi gà Việt Nam là một truyền thống văn hóa độc đáo, gắn liền với đời sống và tâm hồn của người dân Việt

Hyundai Universe 45 Chỗ - Đánh Giá Chi Tiết, Giá B - vvdad0
Hyundai Universe 45 Chỗ - Đánh Giá Chi Tiết, Giá Bán, và Ưu Điểm Xe khách Hyundai Universe 45 chỗ là

Khám Thờ Gia Tiên Gỗ: Sự Kỳ Diệu Của Di Sản Văn Hóa Việt
Trong văn hóa Việt Nam, việc khám thờ gia tiên gỗ không chỉ là một truyền thống mà còn là một cách thể hiện

Hyundai Universe 45 Chỗ - Đánh Giá Chi Tiết, Giá B - 7dejpw
Hyundai Universe 45 Chỗ - Đánh Giá Chi Tiết, Giá Bán, và Ưu Điểm Xe khách Hyundai Universe 45 chỗ là

Cách Nhớ Lô Đề Để Đánh Siêu Chuẩn Tại Jun88 Năm 20 - oxodma
hướng dẫn Đăng Ký JUN88 Đăng Nhập JUN88 Nạp Tiền JUN88 Rút Tiền JUN88 Tải App JUN88 Đại lý JUN88 trò

Fi88 🏆 Link đăng ký vào trang chủ Fi88 nhận thưởn - cgfx3g
🌍🗣 Language Unites Us! 🌍🗣 Have you ever stopped to think about the incredible power of language? It's not

Chọi gà – Bản sắc văn hóa truyền thống Việt NamChọ - mil19h
Gà Chọi Việt Nam, Nguồn Gốc Và Trò Chơi Đá Gà Dân Gian Nguồn gốc và sự phân bố các loại gà chọi ở Việt

Chơi lô đề có giàu được không jun88 mới nhất 2023 - ok1jvl
Đại lý JUN88 trò chơi Thể Thao Casino Đá Gà Xổ Số Bắn Cá - hhvvm0k7x8 - https://jun88.global/choi-lo-

Figure 10: Most newly published APIs were spamming APIs, showing that spamming APIs were prevalent on the RapidAPI platform in June 2023. As of February 2024, we found that the RapidAPI platform has removed almost all spamming APIs, possibly due to our reporting.

B DEVELOPER FOLLOWING NETWORK ANALYSIS

Similar to social media platforms like Twitter, the RapidAPI platform allows developers to follow an API or developer. However, we found that developers can abuse such a function, and there are potential abnormal behaviors when developers follow others. While it is common for regular developers to follow other developers to discover useful APIs, we found 43 developers following more than 100 developers. Surprisingly, almost all of the developers they followed neither published any API nor followed other developers. We named these followed developers as inactive accounts. Figure 11 shows the following network between all developers and their APIs. We marked the developer who follows 2,053 other developers, in which only 25 developers own a published API and only 150 developers follow other developers. Table 7 shows detailed information about several abnormal developers, such as their published API, their followed developers and followed developers' published APIs. We hid their names and links for ethical considerations. To understand why these developers follow so many developer accounts, we further checked the details of these potential inactive accounts. We found that there is a developer following 201 other developers, and these developers only have 3 published APIs. However, 99 of them follow the origin developer's API. It is possible that these 99 developer accounts are created only for following specific APIs published by a developer so that these followed APIs can have a higher ranking when recommending APIs for users.

As we mentioned in Section 5.2.2, spamming API and their developer accounts can also be easily identified based on their network activity. For normal APIs, they can be followed by different developers. However, we find that spamming APIs are more likely to be not followed by other developers because they are not useful. Meanwhile, the developer accounts of these spamming APIs are created for promotion and they prefer not to follow other developers or APIs. These features lead to spamming APIs and their developers being isolated in the following network. Figure 11 shows an isolated cluster in the left bottom corner that only contains the spamming APIs and their developer and doesn't have any connection with the other developers and APIs.

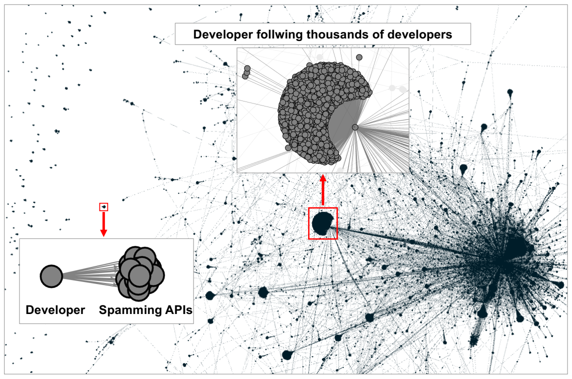


Figure 11: The following network of all developers and APIs in our dataset.

Developer	# of Published API	# of Followed Developer	# of Followed Developers' Published API
A	1	2,054	25
B	1	330	2
C	0	295	3
D	0	206	1
E	0	143	0
F	1	101	0

Table 7: Potential abnormal developers and inactive developers followed by them. The actual developers are masked for ethical considerations. Developers' following network can be used for identifying spamming APIs and their developers.