Research Towards Key Issues of API Security

Ronghua Sun, Qianxun Wang, and Liang Guo^(✉)

Data Star Observatory Technology Co., Ltd., Beijing, China
guoliang@shuziguanxing.com

Abstract. With the mass application of virtualization, micro-services, and cloud-native technologies, the interaction between service entities through APIs has become a norm. Many platforms are still maintaining a large number of old APIs due to business needs. At the same time, many new APIs are gradually going online. Both of these statuses put forward higher requirements for API security. Focusing on old APIs' security protection and other issues, this article starts from the process of asset discovery, vulnerability detection, and security auditing. Aiming at the problem of API asset discovery, this article summarizes the technical methods of automatically clustering unowned API assets using the characteristics of various commonly used APIs. Aiming at new API vulnerability detection, a security analysis method based on finite state machine is proposed. For the first time, the cross-network communication taint propagation based on dynamic taint analysis technology and system-level simulation technology is realized, enabling sensitive data flow tracing in API communication become feasible. We designed a flowbased API security audit system to improve automated API protection. Finally, We analyzed technical opportunities and challenges of API security in detail and prospected for API security research's next direction and development trend.

Keywords: API security · Asset discovery · Vulnerability detection

1 Introduction

An application programming interface (API) is a collection of commands, functions, protocols, and objects. It interacts with external systems by performing common operations. API is flexible, easy to use, and efficient. As a bridge between modules, software, and developers, API is an integral part of modern mobile App, SaaS, and web applications. It is widely used in customer-oriented, partner-oriented, and internal applications [15], such as banking, retail, driverless cars, and smart homes. With the continuous deepening of various industries' networked processes, the API service model is becoming more and more popular.

Currently, using tools can quickly build applications, which means that even inexperienced developers can deploy or integrate applications. This kind of agile development generally doesn't have robust security design or application integration guidance, and they even fail to consider security impacts fully and may expose application logic vulnerability. For example, when designing and implementing extensions, improper constraints on resources or permissions may lead to denial of service attacks. Therefore, the

widespread use of APIs has increased the risk of user safety and privacy leakage, and APIs have gradually become the target of cyber attackers [10].

Although the application may implement robust input validation and access control in the native code, these data are usually not copied in the same class when sent to the server over API. Therefore, an attacker can bypass the client's control. API attacks mostly log in with a legal identity and then simulate normal operations such as multi-source and low-frequency requests. Therefore, the security mechanisms provided by traditional API security gateways such as identity authentication, authority management, rate limiting, and request content verification can not meet the security requirements. API security is a fundamental part of network security. Without secure API, it is impossible to achieve rapid innovation.

2 API Asset Discovery Based on Traffic

With the digital transformation, enterprise apps, mini apps, and light applications on various platforms have developed rapidly. Programmers would like to use the web due to its versatility, economy, and independence from specific platforms or SDKs. Compared with the binary version, the functions of Web products are more comprehensive and reliable [1]. Jeff Atwood, the founder of the StackOverflow website, once asserted, "All programming will be Web programming." According to the 2020 StackOverflow annual developer survey, JavaScript language occupies 67.7% of the share, and this is also the 8th year that JavaScript has been the most commonly used programming language [14]. As the primary language in Web development, JavaScript's widely used proved the universality and process of Web technology. Web development relies on good network interaction between the browser and the back-end server, and its essence is to build applications based on the HTTP protocol. Therefore, API design is critical in Web development. This article mainly discusses API security under the web framework.

At present, there are many security risks in API design, such as various attacks caused by out-of-date API, unauthorized users abusing the API, sensitive API calls, and version confusion. API security testing can understand and mitigate the risks. Therefore, it is necessary to implement security testing on APIs. Security testing generally collects assets and then analyzes the assets through the data flow. At present, the industry mainly relies on API fingerprints to identify API assets. For example, common API frameworks such as REST [4] and GraphQL [6] have some features that can be used to generate corresponding fingerprints. User-defined APIs can also complete matching recognition by customizing fingerprints.

In the field of digital technology, an asset refers to any object of value to an organization. Digital assets are the essential components of business systems and networks. Digital assets are also the foundation for the regular operation of business systems. In recent years, with the rapid development of computer technology, digital assets are gradually becoming an essential tool and support for the operation and management of enterprises and organizations [5]. As the business of enterprises and organizations continues to grow, a large number of unowned API assets and zombie API assets are generated in the business system. Suppose these assets are not maintained for a long time. In that case, they may cause many vulnerabilities and configuration problems,

which affect the response speed to attacks and bring hidden dangers to the safety of enterprises and organizations (Table 2).

Table 1. 3 kind of critical API assets should be concerned

API Asset Type	Asset Characteristics and Threats
Unowned assets	Despite the lack of maintainers, such API assets are still part of the supporters' services due to their limited robustness. Once such API assets are exposed to vulnerabilities, problems such as difficulty in locating issues and difficulty in troubleshooting will arise. Such valuable but poorly maintained APIs should be arranged as soon as possible, and good documentation should be prepared adequately to support the development and maintenance of the team in the future.
Zombie assets	Such API assets also lack maintenance personnel. But compared with unowned assets, it no longer provides valuable services. In other words, the life cycle of this type of API asset has ended, but it has not been offline for some reason. Assets that do not produce value will naturally not be maintained, so such assets are often the target of attacks. Clearing out such assets and taking them offline will improve the security of the service system.
Critical assets	Such API assets are the core assets of Internet applications. A large number of apps integrate such assets. Therefore, the security, robustness, and maintainability of such assets are critical. Applications such as WeChat and Facebook provide APIs for cross-platform authentication and login of other apps, and these APIs appear in many third-party apps. Once such APIs have privacy leaks, there will be a massive number of victim groups.

Strengthening the management of API assets from the development of digital assets is the fundamental means to solve unowned assets. However, many enterprises or organizations have failed to manage from the development cycle's beginning, and many such assets have already existed. The API defines the interaction details between the front and back ends of the Web. Finding out the set of APIs in chaotic IT assets is the key to alleviating such management problems. However, APIs generally hide deeply, and it isn't easy to obtain a comprehensive collection by active scanning. Therefore, it is necessary to complement the asset discovery capabilities based on passive traffic analysis. The general process is first to collect and analyze HTTP/HTTPS traffic and then discover API assets with fingerprint technology. There are various methods of API asset identification. The technical basis is traffic collection. The auxiliary technologies

Table 2. API asset discovery technology: summary, pros and cons

Discovery Techs	Technical summary	Pros and Cons
Traffic capture and manual analysis	Traffic capture is the primary method of API asset discovery. Although there have been quite a few auxiliary methods, capturing traffic packets and performing manual analysis is still a fundamental and effective method.	The technical threshold is low. However, relying on manual analysis requires high technical personnel's experience, which is not conducive to automation.
API fingerprint matching	This is an auxiliary method based on traffic capture. The basic principle is to match the captured data stream according to the characteristics of some common APIs.	The technical solution is relatively mature, and commercial products have appeared. However, it can only identify some special APIs and cannot identify new APIs.
Deep learning	Before the API is officially launched, run a single API service on the test server, capture TCP/UDP sessions, and establish the correspondence between sessions and API features. According to this correspondence, a certain type of deep learning model is trained for future asset discovery.	The technology is relatively advanced, and the prospects are fairly broad. But it cannot deal with mixed API data streams. In practice, communication subjects often use multiple APIs for communication, so the data flow in this context is very different from single API communication, which brings significant challenges to deep learning algorithms.

include API feature matching, API message analysis, deep learning technology, etc. The technical summary and analysis of advantages and disadvantages are shown in Fig. 1.

In April 2021, the data of 500 million Facebook users were publicly sold on the dark web, including the user's nickname, email address, phone number, and home address information. Facebook responded that in 2019, a function of the online business API was misused, resulting in information leakage, affecting approximately 300 million users. Security assets are the most basic and vital carrier in information security management, and it is excellent to grasp secure asset information in all directions without blind spots.

Unowned assets and zombie assets will directly affect network security and even affect the emergency response to attacks. By combing API assets, administrators can quickly restore the currently hidden web architecture to help them find vulnerabilities, make a quick response the first time, and quickly locate the scope of vulnerability. Currently, three key objects of concern in API assets are shown in Table 1.

3 API Vulnerability Detection Method

3.1 API Security Audit Based on Data Flow Tracing

API was established for network communication, so the vulnerabilities in API mainly come from data flow abnormalities. The spread of sensitive data through the web requires strict data flow inspections on related APIs. Therefore, leakage detection and in-depth tracking of sensitive data is an essential issue in API security research. In the traditional data stream tracking research, the taint analysis method has strong applicability [3]. Dynamic taint propagation is widely used in three major areas: malicious code detection and defense, software vulnerability analysis and mining, and sensitive information leakage detection. Like traditional taint analysis technology, various modern web-based software with sensitive data input, such as mini apps, distributed programs, can also use taint analysis after some redesign and transformation to complete data flow analysis.

Web technology is oriented to the interaction between multiple computers based on the network. In contrast, taint analysis technology is more suitable for tests running on a single computing system because of its dependence on memory space. In this contradiction, the traditional technology migration idea is to spread the taint separately on the client and the server and then analyze them separately. However, in this separate test, the API call flow between the tested subjects is not well tracked, and the focus is on the data flow of the client or the server itself.

API-based web communication relies on TCP/IP technology. Data packaged by the sender is routed and forwarded across the network, the data will be re-read into the receiver's memory. The taint analysis technology relies on the tainted mark at the memory byte-level or bit-level [2, 7], so the tainted spot will get into invalid after transmission across a network that cannot be directly traced. In addition, the underlying protocol stack of network transmission in the operating system is generally completed by kernel-mode code, and non-system-level taint marking cannot mark kernel-state variables. Therefore, a whole-system taint analysis platform becomes a necessary condition for API data flow taint tracking. The embarrassment of taint propagation and analysis methods in API data flow analysis urges academic and industrial exploration. A significant achievement of this research is to face this problem.

It can be seen from Fig. 1 that after the client's key variable *k* is sent through the send key over API function, it is essentially handed over to a socket and then transmitted over the TCP/IP network. In a non-RDMA communication network, the address of the sent variable *k* at the receiver cannot be perceived by the client. So the separated client-side taint propagation ends here, and the sensitive memory address cannot be detected. In this process, the call stack of the function send key over API is also sensitive element because this process is closely related to some specific sensitive API. To address this, we propose a method that can mark tainted data in cross-network communication and at

the same time track the API and part that initiated the cross-network contact. Under this method, the client and server can be tainted and traced synchronously on the supported system.

Fig. 1. Taint analysis of crossing network communication

The main idea of this method is based on the system-level emulation of the client and the server, and the communication layer of both is overhead so that the physical layer and link layer of traditional network communication is realized by software. For example, when the client sends a data packet to the server, the system-level emulator of the client will backtrack the variables that constitute this type of data packet in the memory space of the client; after the client sends, the communication layer intercepts the data, and directly it is copied to the buffer corresponding to the server. At the same time, the emulator on the server is notified that it needs to track the reading process of this buffer in advance (before the notification is completed, the reading process is not allowed to be executed); the server emulator will then track the reading process and Relevant variables, and mark these variables as tainted data. As a result, the spread of taint to communication in a set of APIs is realized.

3.2 Finite State Machine Model of Interaction by API

From a broad perspective, all interactions between the client and multiple frontend services can be boiled down to a finite state machine [12]. Finite state machine is a powerful tool used to model the behavior of an object. Its function mainly describes the state sequence that the object experiences during its lifecycle and how to respond to various events from the outside world. The wellknown finite state machine is the state machine of the TCP protocol. Before the new API is developed, FSM modeling should be performed first, and then the operation of the API should be simulated. Through FSM modeling and preliminary simulation, certain specific loopholes in the actual implementation of the entire set of API can be found. And it can also enable developers to understand complex and heavy development requirements more efficiently. As the protocol becomes more and more complex, how to accurately and comprehensively understand or even discover all the details of the entire protocol has become a very challenging problem. Although the principle is straightforward, and it is not difficult to convert API interaction logic to FSM, there are no good tools for how to efficiently and automatically execute

and test FSM. Concerning the idea of automated formal verification, we designed and implemented a framework for fast FSM semi-automatic construction of APIs.

In API communication, take IoT smart home devices as an example. Device A will be placed in state S1 through the user's physical click. In S1, device A will become a WiFi access point and wait for a connection. After that, when the user's mobile phone joins this WiFi, and the device is notified of the WLAN password, device A will enter state S2, in which it will wait for the Internet connection. If getting the Internet connection successfully, device A will enter state S3.1, in which it will wait for cloud device ID registration. Otherwise, it will enter state S3.2, that is, the password is incorrect, couldn't connect to the IoT cloud, cloud refused, etc., then device A will returns to state S2 and continues to wait. In S3.1, if a successful ACK of cloud binding can be received, it will enter state S4 and running regularly. The specific process, such as user reconfiguration, will not be repeated here. It can be found that the device can transfer between S1, S2, S3.1, S3.2, and S4 due to various trigger conditions. However, a set of APIs does not only contain one device. We still take IoT as an example. There are usually three objects in the cloud, mobile app, and device [17]. It is not enough to model the state of one thing. By associating the state transitions of three objects, a directed diagram of state transitions can be formed or called an FSM. In this diagram, we define safe initialization conditions, such as the user pressing the reset button and then traversing the path according to all possible trigger conditions until all feasible paths are completed. If an event defined as dangerous is triggered in the FSM diagram, such as the device goes offline and falls into an unavailable state, the API is problematic. By constructing FSM, the model can be tested quickly, and the design flaws of API can be found, and the vulnerabilities can be found in time.

3.3 Demonstration

In the traditional formal verification of network protocols, security investigations are entirely dependent on the input which describe the whole protocol. So when the input is unreliable, the result of the formal verification is also unpredictable. The principle is straightforward. After all, all model checking and system simulation can only prove the insecurity of a protocol, but not the security of the protocol. The same reasoning is fully applicable to the API security field. According to the new API attack chain discovered by Jice Wang et al. [16], we try to replicate the experiment based on our theory. Jice's paper proposes a new attack vector: multiple APIs integrated in the same App will share the same virtual address space, so there are attacks between each other. And the attack is mainly focused on the theft of user's data, which undoubtedly poses a great threat to the current compassionate privacy data protection! However, the primary method of this work is to reverse engineer and statically analyze the Android APK file, so fewer attack instances were found.

As shown in Fig. 2, the tested Android App is constructed by us to test whether our API will be attacked. Like WeChat, an open platform that provides logins, applicants must be qualified when providing open services. Therefore, apps with WeChat login qualifications are considered won't steal WeChat user's data. In the Cross-Library API call, the caller and the caller are in the same memory space, so a library can naturally call functions in another library through specific APIs. In this case, API security becomes

particularly critical. A malicious library can call our API and then reasonably get all kinds of data outside the security boundary from our server. In the narration of the first two sections of this chapter, we emphasized the importance of dynamic taint tracking. We set a certain type of Token as tainted data. When a request is sent on the App side, the caller related to the Token will appear in the tracking list. By setting up a reasonable tracking list, you can easily find illegal cases that appear in the taint mark list. Because our system overloads the network transmission layer, when the Token is sent to the receiver via TCP, it can directly find the caller's information on the server, thereby intercepting the attack. Unlike Jice's work, we focus on the security of the API instead of emphasizing the existence of the attack chain.

The above case demonstrates the importance of system-level taint tracking tools for API protection. Combined with the automata shown in Fig. 2, vulnerabilities outside the protocol can be found. This is of great positive significance for improving API protection capabilities.

Fig. 2. API under cross library attacks

3.4 Relationship Between FSM Testing and Data Flow Taint Analysis

FSM modeling and possible simulation of the communication time sequences between network objects based on API communication can find the loopholes of the protocol at a relatively macro level. It is also conducive to simplifying and reducing some APIs in the later stage. Full system-level emulation is performed on the network entities running the entire set of APIs, and the taint analysis method that overloads TCP/IP communication can be used to discover the moving traces of sensitive data in the API. The above two methods have certain similarities, both of which have carried out different degrees of modeling, and the models need to be executed. However, the granularity of the two ideas is entirely different. The FSM-based method focuses on the macro logic of the API system. By analyzing the topology of the FSM, some hidden logical vulnerabilities

can be found. For example, by using FSM to investigate the application of MQTT in IoT, Yan Jia [9] and others found that during the process of binding and unbinding intelligent home devices, the MQTT protocol will enable the former authorized person to regain control. Emulating the client and server running the API and tracking the sensitive data that the programmer cares about (after this referred to as the EMU method) operated at a more fine-grained level. Although the EMU method also needs to be abstracted, it only removes some unnecessary communication intermediate processes. The specific code is still executed in the actual runtime environment. So, the EMU method will find vulnerabilities related to the tested API more precisely. In the EMU method, whole-system simulations are required for the client and server, and some symbolic execution methods are even needed to bypass the input and output of the peripherals of the complex client. Therefore, there are many automation problems in the design.

4 API Security Audit System Based on Traffic

At present, it is not easy to manage APIs safely and effectively. Building and managing APIs quickly and easily has become an essential issue in developing web-based programs [13]. Traditional API security gateways are too expensive to deploy and maintain. They lack effective strategies to build defenses and cannot efficiently respond to the threats of emerging automated tools. The development of automated auditing API methods to improve security is still challenging. In response to the above problems, this section proposes an API security audit system based on the Internet traffic, which is designed based on intelligence asset collection to provide research ideas for the security protection of massive API assets.

4.1 Research Ideas

API exists in the form of digital data. Using API as a form of digital asset management can more effectively improve API security. From the perspective of API assets, API security audit is divided into three parts, as shown in Fig. 3.

The API asset discovery module uses various methods such as traffic analysis, docking data, and importing data to identify the data processing modes of APIs, such as RESTful and GraphQL. This module will enable the full discovering of API assets, especially for the discovery of unknown APIs. What's more, precise identification is given to prevent the access of obscure APIs from causing the unavailability of functions.

API asset portrait uses data analysis to accurately portray the API's functions and permissions, such as user login, registration, data query, and administrator permissions. According to the API profile list, we can quickly check the status of each API, such as usage, access source, exception, etc.

API safety detection and protection module use active and passive methods to audit the API security of application and business dimensions. Remote command/code execution, data leakage, unauthorized access, unauthorized access, logic defects, etc., will be detected during this part. According to the audit results, we will respond dynamically and implement protective measures to increase the difficulty of attacks.

Fig. 3. Security audit process

4.2 System Framework Design

When designing the system, we should emphasize the network, data, and business APIs. On the network side, connecting the internal and external application terminals through APIs would expand network attacks, increases risk transmission paths and make the system more vulnerable to malicious. All of these could lead to server intrusion and business continuity interruption. In terms of data, once the open API has design flaws or improper permission settings, malicious attackers may illegally obtain user data. In terms of business, as the amount of API openness increases, service interfaces may be used beyond the scope, increasing the probability of business compliance risk events.

Starting from the characteristics of data processing, taking into efficient reading and writing, data conversion, transaction processing, and caching strategies, we can divide the system into four layers from bottom to top: collection layer, pre-processing layer, detection layer, and the user interfaces layer. The structure is as follows, shown in Fig. 4.

Fig. 4. Security audit framework

The collection layer is mainly used to collect data in four ways. Traffic collection takes deploying collection nodes and PCAP import, which is the essential way. Tool importing is by capturing data directly in contact with software such as testing tools and proxy tools. The API also communicates information through hardware devices and captures data from web application firewalls and API gateways utilizing device docking. Intelligence data on various platforms is also an essential component of API assets. Intelligence-based API asset collection is mainly from the following aspects: cooperation with APP security vendors to obtain API asset information in APP; obtain API asset information in applets; get API asset information leaked in well-known open code warehouses; Cooperate with Internet terminals, browsers, cloud platforms, Internet of Things, Internet of Vehicles and other companies to obtain API asset information. The above four methods can establish an API asset information system more comprehensively.

4.3 Key Techniques

The core of the API security audit system proposed in this paper is flow-based. First, collect traffic according to the underlying driver. Then, recreate an API asset library using URL, request templates, etc.; the original traffic can be directly constructed to form a traffic library that can be massively stored, quickly searched, and visualized. Last, Perform data pre-processing on the API asset library and traffic library.

Extract the assets in the API asset library for active detection, analyze the availability of the detection results, and mark if they are not available. For the available assets, conduct functional analysis and permission analysis and construct the API vulnerability library for the problems found. Analyze the traffic content in the traffic database, extract sensitive information, and write the issues into the API vulnerability database. Through a series of API audits around traffic, API security issues can be handled quickly and efficiently.

According to the OWASP API Security Top 10 2019 [11], it can be found that API security issues are more concentrated in the business security field, such as ultra vires, data leakage, and invalid identity authentication. The security of conventional applications mainly relies on the detection rules to analyze the request and response and then judge the risk. However, API business security needs to dynamically compare and analyze multiple consecutive business request data to discover hidden business security hazards such as unauthorized access and user identity authentication failure. To improve the efficiency of traffic auditing, deep learning methods can be used to detect such problems continuously.

When the API sample data is sufficient, we can continuously accumulate positive and negative samples during the API security audit and constantly train and modify the security audit model through cluster analysis, supervised learning, and other methods. Then we can use the security audit model to discover more unknown API security threats.

5 Opportunities and Challenges

With the rapid development of mobilization, openness, and IoT, API provides the impetus for today's digital transformation. With the changes in business forms, the open

cooperation of business APIs has become an inevitable trend, and new security opportunities often accompany emerging businesses. Although digital transformation promotes API in various industries, malicious threats target API more than ever. There is a big gap between the current API security situation and actual needs, and the API ecosystem faces different challenges and threats.

Applications and logic transform to the cloud to provide services for more users with exposing more attack surfaces of APIs and triggering a variety of potential attacks. Attackers can use network traffic, reverse code, security vulnerabilities, and other means to carry out attacks. For example, in terms of authentication, attackers obtain API login credentials through phishing, social engineering, botnets, etc., to steal customer data or personal information. In terms of vulnerability exploitation, common attack methods such as SQL injection and XSS could be used to steal sensitive data or destroy the system. API attacks have been instrumentalized. Attackers can use tools to collect a list of domain names and APIs used for attacks and then use other means to find or delete sensitive data.

Table 3. Issues and methods of API security research.

Issues	Methods
API security framework	Service grid
Accurate recognition and portrait	Deep learning, reinforcement learning
Automatic correlation	Machine learning
Security situation prediction	Automatic perception and Self-learning
Cross-platform interconnection	Security value sharing
Monitoring and visualization	Big data
API vulnerability mining	Automated mining
API protection system	New artificial intelligence technology

The number of APIs continues to grow, and API security research needs to continue to keep pace with the times and innovate. Table 3 summarizes some complicated problems and possible solutions in API security research. Build an API security framework through the service grid, classify, share and aggregate API information, and improve the efficiency and security of API management [8]. Service grid access control has advantages and can enhance the authentication and authorization process. API security management needs to identify all aspects of API information accurately, build an API asset attribute library through deep learning and other methods, automatically sort out API asset portraits, and provide rich analysis data for API asset security analysis. The intelligent model based on machine learning solves the problem of automatic association between new API and existing assets through active scanning, passive traffic monitoring, etc., and improves automation. Security posture assessment is an essential part of network security, and API security posture plays a significant role in security decision-making. In the face of massive network attacks, it uses automatic perception to avert

dangerous situations in time and improves prediction accuracy through self-learning adjustments. Interconnection is an effective means to realize the integration of data resources in various industries, enriching data and business functionality through value sharing and integration. In the process of opening to the outside world and integration in the form of API, unified security life-cycle management and fine-grained security control can help the organization to achieve effective governance inside and outside. API security needs to be monitored throughout the entire process. The data obtained by using big data visualization is convenient for managers to intuitively and accurately grasp the current security status. Excavating the vulnerabilities of the existing large-scale and complex API systems and discovering the vulnerabilities of newly launched APIs are two fundamental issues in the academic world. It will also be a hot topic of continuous attention. Currently, the API protection system is still immature. With the constant emergence of new scenarios, security solutions are not fully applicable in open service scenarios. Use a new generation of artificial intelligence technology to solve multi-level security protection and security threats and increase the intelligence level of API security.

6 Conclusion

The core functions and micro-service architecture of mobile applications, websites, applications, etc., are inseparable from the support of APIs. The scrum development model is a mainstream development mode. In improving the speed and flexibility of innovation, API risks are underestimated, and API construction security is ignored.

This work sorts out API security protection-related methods and proposes a finite state machine analysis method that can help improve the API security design and safety evaluation. An API security audit framework was proposed, the core of which is API auditing based on traffic, which provides ideas for studying the best practices for protecting APIs. Finally, it explains the existing problems in the current research and looks forward to its future development.

References

1. Corral, L., Sillitti, A., Succi, G., Garibbo, A., Ramella, P.: Evolution of mobile software development from platform-specific to web-based multiplatform paradigm. In: Proceedings of the 10th SIGPLAN Symposium on New ideas, New Paradigms, and Reflections on Programming and Software, pp. 181–183 (2012)
2. Davanian, A., Qi, Z., Qu, Y., Yin, H.: Decaf++: elastic whole-system dynamic taint analysis. In: 22nd International Symposium on Research in Attacks, Intrusions and Defenses ({RAID} 2019), pp. 31–45 (2019)
3. Ding, Y., Wei, T., Xue, H., Zhang, Y., Zhang, C., Han, X.: Accurate and efficient exploit capture and classification. *Sci. China Inf. Sci.* **60**(5), 1–17 (2017)
4. Fielding, R.T.: Architectural styles and the design of network-based software architectures. University of California, Irvine (2000)
5. Gandert, N., et al.: Method and system for searching for digital assets, uS Patent 9,251,172 (2016)
6. Hartig, O., Pérez, J.: An Initial Analysis of Facebook's GraphQL Language (2017)

7. Henderson, A., Yan, L.K., Hu, X., Prakash, A., Yin, H., McCamant, S.: Decaf: a platform-neutral whole-system dynamic binary analysis platform. *IEEE Trans. Softw. Eng.* **43**(2), 164–184 (2016)
8. Hussain, F., Li, W., Noye, B., Sharieh, S., Ferworn, A.: Intelligent service mesh framework for API security and management. In: 2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON). pp. 0735–0742. IEEE (2019)
9. Jia, Y., et al.: Burglars' IoT paradise: understanding and mitigating security risks of general messaging protocols on IoT clouds. In: 2020 IEEE Symposium on Security and Privacy (SP). pp. 465–481. IEEE (2020)
10. Mendoza, A., Gu, G.: Mobile application web API reconnaissance: web-to-mobile inconsistencies & vulnerabilities. In: 2018 IEEE Symposium on Security and Privacy (SP), pp. 756–769. IEEE (2018)
11. OWASP Foundation (2019). <https://owasp.org/www-project-api-security/>
12. Ramesh, G., Menen, A.: Automated dynamic approach for detecting ransomware using finite-state machine. *Decis. Supp. Syst.* **138**, 113400 (2020)
13. Song, Y.: Resarch and implementation of monitoring of monitoring oriented open API service. Ph.D. thesis, Beijing University of Posts and Telecommunications (2017)
14. StackOverflow Survey (2020). <https://insights.stackoverflow.com/survey/>
15. Sucuri Guides (2021). <https://sucuri.net/guides/owasp-top-10-security-vulnerabilities-2021/>
16. Wang, J., et al.: Understanding malicious cross-library data harvesting on android. In: 30th {USENIX} Security Symposium ({USENIX} Security 21) (2021)
17. Zhou, W., et al.: Discovering and understanding the security hazards in the interactions between iot devices, mobile apps, and clouds on smart home platforms. In: 28th {USENIX} Security Symposium ({USENIX} Security 19), pp. 1133–1150 (2019)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.