Uncovering the App Cloud Access Risks under Recommended IAM Security Practices

Hengtong Lu
Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China,
luhengtong@iie.ac.cn

Yan Zhang
Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China,
zhangyan@iie.ac.cn

Qingfeng Tang
Macau University of Science and Technology, FIE
tangtsingfong@outlook.com

Pengwei Zhan
Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China,
zhanpengwei@iie.ac.cn

Abstract

The rapid development of mobile applications and cloud computing has led to the widespread adoption of cloud service platforms for mobile backend services. However, improper use of cloud credentials has frequently resulted in the leakage of application data on cloud servers. Despite security recommendations from cloud service providers, vulnerabilities persist. To assess the effectiveness of these measures, we propose a detection system to identify cloud credential leaks in mobile applications, including hard-coded credentials and those stored on servers. We analyzed 21,724 applications from Google Play and one Chinese market, revealing new attacks triggered by stolen cloud credentials. Our findings indicate that even temporary credentials recommended by cloud providers may pose security risks. We identified 893 applications using cloud credentials from the three major providers, with 945 credentials found. By analyzing these credentials, we uncovered severe vulnerabilities in 356 apps, such as personally identifiable information (PII) leakage, credential forgery, and remote code execution (RCE). These issues threaten user privacy and app security. We also evaluated developer adherence to recommended IAM best practices and provided suggestions for improving cloud credential security, highlighting issues such as improper permissions, insufficient protection, outdated versions, and regional variants.

Keywords

Mobile Apps, Cloud Service, Security Risk

1 Introduction

Nowadays, mobile applications are integral to our daily lives, offering services that span social, medical, entertainment, and various other fields. Cloud service platforms, due to their high scalability and low cost, have become the preferred choice for constructing the backend services of mobile apps. Mainstream cloud service providers, such as Amazon AWS and Alibaba Cloud, offer prepackaged mobile cloud solutions, including specialized software development kits (SDKs) and application programming interfaces (APIs), which developers can integrate into their apps with just a few lines of code.

However, in recent years, security issues in cloud backend service have led to numerous incidents of sensitive information leaks, including PII and company data. For instance, a cloud configuration error at Toyota led to the leak of personal information of 260,000 car owners in Japan[3]. Turkish low-cost airline Pegasus Airlines leaked approximately 6.5 TB of data due to a configuration error in an AWS S3 bucket[2].Despite the substantial efforts by cloud service providers to secure their services, the clients of these services, such as mobile apps, are often considered the weakest link.

Prior research has primarily focused on analyzing risks associated with long-term cloud credentials (such as hardcoded root keys or overprivileged regular user credentials). For instance, Zhou[41] et al. identified security vulnerabilities in AWS credentials within Android systems, while Wen[38] et al. discovered similar issues with persistent credentials in iOS environments. Zuo[42] et al. demonstrated that root credential misuse constitutes one of the primary causes of cloud data breaches. Additionally, Wang[37] et al. revealed that regular user credentials with excessive permissions represent another critical factor leading to cloud data leaks. However, these studies largely overlooked temporary credentials, despite cloud service providers recommending them as a security best practice. This research gap leaves unanswered whether temporary credentials genuinely mitigate risks or still suffer from permission configuration issues.

Furthermore, previous analyses of cloud credential permissions were limited in scope, typically concentrating on single services (e.g., storage buckets) or static permission configurations. Chen[26] et al. investigated six types of vulnerabilities in cloud storage buckets, while Yadmani[32] et al. conducted extensive research on confidential data leakage in cloud storage. Although Wang et al. inferred credential permissions through dynamic probing, they failed to systematically evaluate the relationship between temporary credentials and cross-service permissions - a gap that could enable novel attacks like log forgery or remote code execution (RCE). This study pioneers fine-grained analysis of both long-term and temporary credential permissions, and thoroughly examines how permission combinations across cloud services amplify risks - a dimension that prior research has insufficiently explored.

Proceedings on Privacy Enhancing Technologies 2025(4), 763–776
© 2025 Copyright held by the owner/author(s).
https://doi.org/10.56553/popets-2025-0156

In response to the security risks associated with using cloud services in mobile applications, cloud service providers have implemented various security measures within their Identity and Access Management (IAM) services[16][17][18]. In the field of IAM services, cloud service providers offer numerous cloud identities and intricate permission policy configurations, accompanied by detailed documentation. Our comprehensive analysis of IAM-related development documentation and practical case studies from Amazon AWS[9], Alibaba Cloud[6], and Tencent Cloud[19] reveals that these security measures primarily focus on two key aspects: (1) Providers encourage developers to adopt safer credential mechanisms. This includes using standard user cloud credentials instead of root credentials with maximum permissions and distributing temporary credentials to users via private servers; (2) They advocate for granting customized permissions to standard users. This approach emphasizes the principle of least privilege, ensuring appropriate permission configurations.

Despite the implementation and recommendation of various IAM-related security measures by cloud service providers, the effectiveness of these measures in the mobile app domain remains uncertain. This uncertainty leads us to two research questions:

Q1. What is the current usage rate of these security measures across various apps, and are they widely adopted and valued by developers?

Q2. Do the existing security measures continue to pose potential security risks, and do existing security measures cover all potential security risks of cloud backend services in mobile app?

To address these questions, we propose Cloudet, a semi-automated detection system for app cloud service security that aims to detect potential risks of compromised cloud services used by mobile applications. We analyze IAM practices across a wide range of mobile applications in the international app market, abstracting three commonly used cloud service access models. We then assess the effectiveness of security measures provided by cloud service providers within these models.

To evaluate the usage rate of these security measures, we conduct a large-scale analysis of cloud credential leakage in mobile applications from Google Play and Chinese mobile app markets. Additionally, we investigate the specific cloud services accessible via these credentials. Our analysis extends beyond the commonly known mobile back-end as a service (mBaaS) to include potential security risks in Infrastructure as a Service (IaaS) Cloud services, providing a comprehensive evaluation of the potential security risks resulting from cloud credential leakage.

In summary, our contributions are as follows:

- We conducted the first large-scale empirical study on temporary credentials in mobile applications, analyzing 21,724 apps from Google Play and 360 App Store. The results reveal that only 47% of apps implemented the temporary credential mechanism recommended by cloud providers, indicating widespread non-compliance with security best practices. Meanwhile, among apps using temporary credentials, 78.1% exhibited exposure windows where credentials remained vulnerable to short-term interception during their validity period. Furthermore, 27% of exposed temporary credentials were found to have excessive permissions, potentially leading to more severe consequences such as data breaches.
- Previous research on credential permissions has primarily focused on common cloud services that directly store and process personal information, such as cloud storage and push notification services, attempting to exploit permissions to access various types of sensitive data within these services. In our current study, we innovatively extend this exploration by utilizing obtained temporary or permanent credentials to investigate their permission boundaries when used with fundamental or auxiliary cloud services. This investigation revealed that permission abuse in basic cloud services can indirectly lead to severe personal information leakage vulnerabilities. We have identified two novel attack patterns: cloud log spoofing and remote code execution, accompanied by detailed case demonstrations.
- We established a multi-tiered response mechanism, implementing responsible disclosure for 356 vulnerable apps through channels like CERT and email notifications, while conducting root-cause investigations via developer surveys. This approach achieved an 89% vulnerability remediation rate and revealed fundamental causes such as time pressure (62%) through the survey analysis.

2 Modeling App Access to Cloud Services

With the rapid development of cloud computing technology, traditional app development models are increasingly challenged by scalability, flexibility, and cost-effectiveness demands. To meet the growing demands of the mobile application market, major cloud service providers have introduced mBaaS, providing mobile developers with a convenient development environment and robust backend support. mBaaS includes services such as data storage, user authentication, and push notifications, allowing developers to focus more on the front-end development without spending excessive effort on building and maintaining the underlying backend infrastructure. In addition to mBaaS, mobile developers also use foundational cloud computing services such as cloud servers and cloud databases, to provide higher levels of flexibility and scalability. However, this also means that if the app's cloud is attacked, it could lead to more severe data breaches and financial losses.

To facilitate developers using cloud services, mainstream cloud service providers offer SDKs that can be integrated into the apps. These mobile SDKs contain multiple cloud service APIs (Cloud APIs), allowing developers to access purchased cloud services by calling these. When using cloud services, it is necessary to identify the requester and verify their access permissions via IAM. Therefore, developers must generate cloud credentials in the cloud management console[31] and pass them to the app so that the cloud can verify the app's identity and allow access to specific resources.

As cloud service providers continuously introduce new security measures in IAM, the methods through which mobile applications obtain cloud credentials have also evolved. Through the analysis of 21,724 applications, we abstracted three workflow models for mobile applications accessing cloud services, as shown in Figure 1. These applications were sourced from two major markets: Google Play and the Chinese 360 Market. Specifically, we focused on the

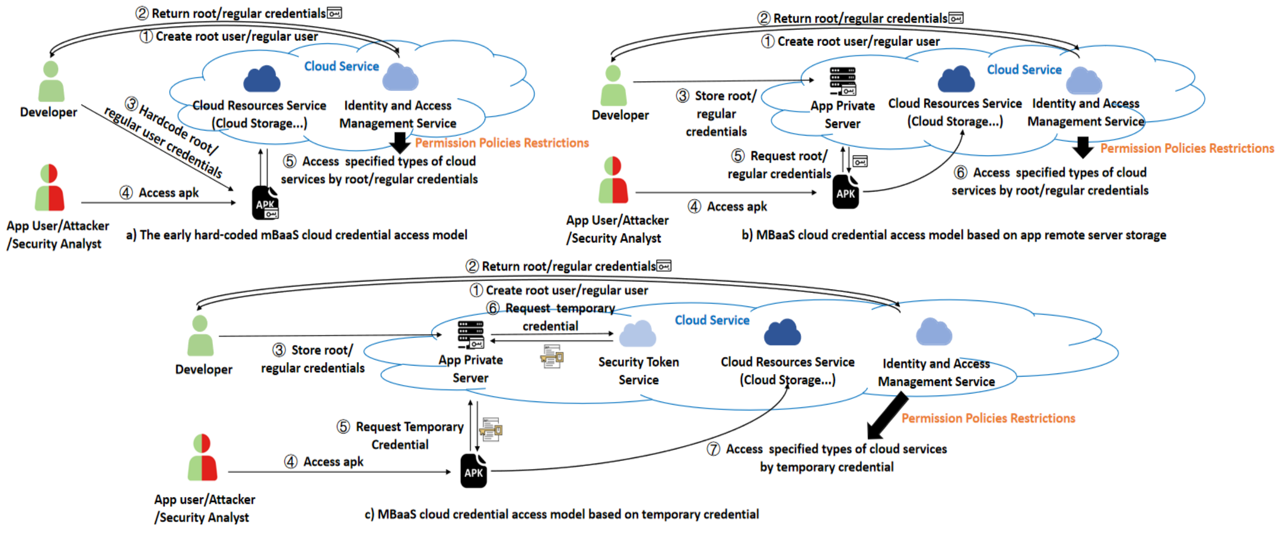

② Return root/regular credentials
① Create root user/regular user
Cloud Service
Developer
Cloud Resources Service (Cloud Storage...)
Identity and Access Management Service
③ Hardcode root/ regular user credentials
Permission Policies Restrictions
⑤ Access specified types of cloud services by root/regular credentials
④ Access apk
APK
App User/Attacker /Security Analyst
a) The early hard-coded mBaaS cloud credential access model
③ Store root/ regular credentials
App Private Server
⑤ Request root/ regular credentials
⑥ Access specified types of cloud services by root/regular credentials
b) MBaaS cloud credential access model based on app remote server storage
⑥ Request temporary credential
Security Token Service
⑤ Request Temporary Credential
⑦ Access specified types of cloud services by temporary credential
App user/Attacker /Security Analyst
c) MBaaS cloud credential access model based on temporary credential

Figure 1: Models of App Access to Cloud

most downloaded applications within each category, collecting 13,806 applications from 32 categories in Google Play and 7,918 applications from 19 categories in the Chinese 360 Market. These applications span a variety of types, including social media, shopping, music, and financial applications. For more detailed information regarding the sources and categories of the analyzed applications, please refer to Section 4.1, which provides an in-depth breakdown of the dataset.

Cloud credentials used in applications can be categorized into long-term and temporary credentials. Long-term credentials include root user credentials and regular user credentials. Root credentials can be generated by the root user through accessing cloud services and provide full control over the purchased cloud resources under their account. Regular user credentials can be generated by regular users, such as IAM users in AWS[12] or RAM users in Alibaba Cloud[13], with permissions to access specific cloud resources.Previous studies have demonstrated that the misuse of root credentials[42] and improper permission settings of regular user credentials[37] are major causes of cloud data breaches, corresponding to the model in Figure 1(a). These applications are distributed to users, attackers, and security analysts through various channels (e.g., mobile app markets or developer websites). Developers first generate root or regular user credentials through the IAM service (Steps 1-2) and hard-code these credentials into the application's source code during development (Step 3). When the application needs to use cloud services, it can use the cloud credentials to call cloud APIs and access other cloud services purchased by the developer (Steps 4-5). However, this approach introduces potential security risks and management challenges. Firstly, attackers or security analysts can reverse-engineer hard-coded credentials to perform unauthorized actions. Secondly, updating cloud credentials requires recompiling and republishing the application, resulting in management inconvenience and delays.

Recognizing the insecurity of hardcoding credentials, some mobile app developers shift the long-term credentials to app private server and distribute them securely to mobile app clients, as shown in Figure 1(b). Developers generate root or regular user credentials via access control cloud services and store them on app private server (step 1-3). When the app needs to use cloud services, it requests the stored credentials from the private server in real-time, caching them in local storage such as memory or SharedPreferences (step 4-5). The app then uses these credentials to call Cloud APIs and access other cloud services (step 6). Although some app private servers may not be cloud-based, our modeling covers the majority of scenarios. This approach addresses the issues of hardcoding cloud credentials but introduces new potential risks. During the transmission of credentials from the private server to the client, attackers could intercept the credentials through man-in-the-middle attacks. Unencrypted transmissions and unverified clients can also lead to successful attacks.

Given the potential risks of misuse associated with long-term cloud credentials, cloud service providers have introduced a series of security mechanisms for different app scenarios. Consequently, using temporary credentials to access cloud services has become a new trend, represented in Figure 1(c). In this model, developers store long-term credentials on a private server (step 1-3). When the app needs to access specific cloud services, it requests a temporary credential from the private server, which generates this credential in real-time via secure token cloud services (step 4-7). The temporary credential is then returned to the app client. These credentials are time-limited, typically valid for no more than 36 hours, significantly reducing the risk of credential exposure. To further control access permissions, cloud service providers have introduced security mechanisms such as AssumeRole[8] and Policy[7]. However, the complexity of permission configuration can lead to developers mistakenly granting excessive permissions to temporary credentials.

Although private servers typically implement authentication mechanisms to restrict access, attackers or security analysts might still intercept the URL of the private server that issues temporary

Table 1: Commonly used cloud services for App

CSP	cloud service	Initialization APIs	Parameter Type	counts
AWS	S3(Simple Storage Service)	com.amazonaws.auth.BasicSessionCredentials: void <init>(String, String)	Long-term credentials	6
		com.amazonaws.auth.AWSAbstractCognitoDeveloperIdentityProvider: void <init>(String, String)		
	SNS(Simple Notification)	com.amazonaws.auth.BasicAWSCredentials: void <init>(String, String, String)		
		com.amazonaws.auth.AWSAbstractCognitoDeveloperIdentityProvider : void <init>(String, String)	Temporary credentials	
	CloudWatch	com.amazonaws.auth.CognitoCachingCredentialsProvider : void <init>(String, String)		
		com.amazonaws.mobileconnectors.cognitoidentityprovider.CognitoUserPool : void <init>(String, String)		
Alibaba	OSS(Object Storage Service)	com.alibaba.sdk.android.oss.common.auth.OSSFederationToken: void <init>(String, String, String)	Temporary Credentials	6
		com.alibaba.sdk.android.oss.common.auth.OSSStsTokenCredentialProvider: void <init>(String, String, String)		
		com.alibaba.sdk.android.common.auth.StsTokenCredentialProvider: void <init>(String, String, String)		
		com.alibaba.sdk.android.common.auth.OSSPlainTextAKSKCredentialProvider: void <init>(String, String)	Long-term Credentials	
		com.alibaba.sdk.android.oss.common.utils.OSSUtils: java.lang.String sign: (String, String, String)		
		com.alibaba.sdk.android.oss.common.auth.OSSAuthCredentialsProvider:(String)	Auth URL	
	SLS(Simple Log Servic)	com.aliyun.sls.android.producer.resetSecurityToken: void <init>(String, String, String)	Temporary Credentials	3
		com.aliyun.sls.android.producer.LogProducerConfig: void <init>(String,String)	Long-term Credentials	
		com.aliyun.sls.android.sdk.core.auth.PlainTextAKSKCredentialProvider: void <init>(String,String)		
	MNS(Message Servic)	com.aliyun.mns.client.CloudAccount: void <init>(String, String, String)	Long-term Credentials	1
Tencent	COS(Cloud Object Storage)	com.tencent.qcloud.core.auth.ShortTimeCredentialProvider: void <init>(String, String)	Long-term Credentials	3
	CLS(Cloud Log Service)	com.tencent.qcloud.core.auth.SessionQCloudCredentials: void <init>(String, String, String)	Temporary Credentials	
	CMQ(Cloud Message Queue)	com.qcloud.cos.auth.BasicCOSCredentials: void <init>(String, String, String)		

credentials through reverse engineering and man-in-the-middle (MITM) attacks. For instance, if the app client does not properly validate server certificates or uses weak authentication methods, attackers could impersonate the app and send requests to the private server.

3 mBaaS Analysis

The mBaaS cloud services offered by Amazon AWS, Alibaba Cloud, and Tencent Cloud share similar usage scenarios, which can be categorized into three main types: cloud storage services, cloud logging services, and cloud push notification services. Each of these cloud services has different aliases, as shown in the second column of Table 1. Each cloud service provides a series of Cloud APIs to meet a wide range of functional requirements. Based on the process of mobile apps accessing cloud services and the differences in Cloud APIs, we further divide the APIs into initialization APIs and resource access APIs.

Initialization APIs are used for authentication and authorization when interacting with cloud services, verifying user identities, and checking the permissions for accessing resources. Therefore, mobile apps must first call initialization APIs before invoking resource access APIs to operate on cloud resources. These Cloud APIs include parameters such as cloud credentials and other critical resource information (e.g., Region and EndPoint), making them essential for analyzing cloud credential leaks in mobile apps. Specifically, we examine 6 initialization APIs included in AWS, 9 in Alibaba Cloud, and 7 in Tencent Cloud. For instance, AWS's S3 cloud service commonly uses the BasicSessionCredentials initialization API to pass temporary credentials and the BasicAWSCredentials initialization API for long-term cloud credentials. Similarly, Alibaba Cloud uses the OSSFederationToken initialization API for temporary credentials, while Tencent Cloud employs the SessionQCloudCredentials API for similar operations. Furthermore, Alibaba Cloud embeds the corresponding initialization APIs directly into the SDKs for different cloud services, whereas AWS and Tencent Cloud provide dedicated SDKs for initialization APIs.

Resource access APIs are responsible for operations on cloud resources, including creating, deleting, updating, and querying resources to meet application-specific requirements and adapt to various business scenarios. For example, AWS provides cloud storage services through its Simple Storage Service (S3) with an SDK named AWS S3. This service allows users to use buckets as containers to store data objects and provides a series of Cloud APIs for data manipulation, including create, delete, update, and query operations, as well as supporting data synchronization and offline access functions. These APIs include operations such as listing buckets (s3:ListAllMyBuckets), downloading objects (s3:GetObject), and uploading objects (s3:PutObject), among 99 others[14]. Some of these APIs can perform sensitive operations, and improper permission settings may cause serious harm, which we analyze in detail in Section 4.3.

Although users cannot directly call the Cloud APIs to access these cloud services, the developer's unsafe operation make it possible for the users to obtain resources on the cloud server via app client. Improper use of root credentials, excessive permissions granted to regular user and temporary credentials can allow attackers to exploit leaked cloud credentials to access cloud resources, posing severe security threats. Attackers might use these permissions to perform various malicious operations, such as accessing multiple cloud server instances and launching RCE attacks.

4 Design and Implemention

This chapter presents the design and implementation of Cloudet, a detection system designed to evaluate the effectiveness of security measures implemented by major cloud service providers for mobile apps and to assess the potential severity of harm when these measures are bypassed. Cloudet semi-automates the detection of cloud data leakage risks in mobile apps, enhancing the understanding of cloud credential vulnerabilities. As depicted in Figure 2, Cloudet comprises four primary components: App dataset acquisition, static analysis, dynamic analysis, and vulnerability analysis. Each component plays a crucial role in identifying and analyzing the security

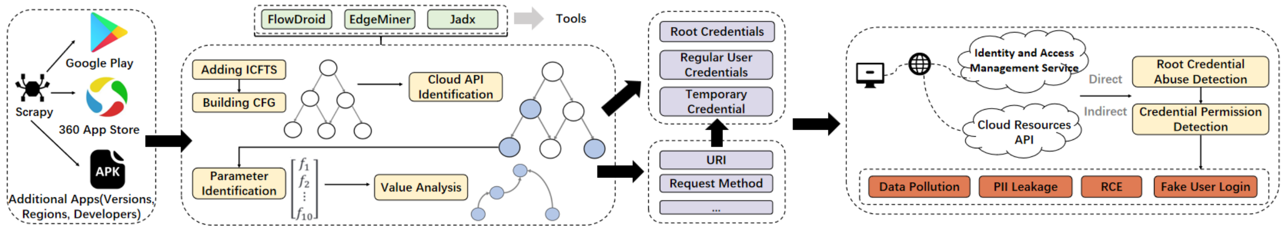


Figure 2: Overview of Cloudet

threats present in mobile cloud environments, thereby facilitating a comprehensive assessment of cloud data protection strategies.

4.1 App Dataset Collection

To achieve large-scale APK detection, we developed a web crawler using Scrapy[15] to collect a dataset of 21,724 apps between December 2022 and July 2024. Specifically, we focused on the most downloaded applications within each category, collecting 13,806 applications across 32 categories from Google Play[11] and 7,918 applications across 19 categories from the Chinese 360 Market[5]. To further enhance the depth and breadth of our dataset, we additionally gathered 600 applications: 200 historical versions, 200 applications from the same developers, and 200 regional variants of the same applications.

Specifically, for applications that use cloud APIs but have not been identified as vulnerable, we expanded the dataset in two dimensions: horizontally, by including additional applications from the same developer and various regional variants, and vertically, by gathering different historical versions of the same application. Horizontally, we collected additional applications from the same developer as well as various regional variants of the same application. Typically, for ease of management, a developer's applications share the same cloud resources. Developers can also create cloud credentials with specific permissions for each application, ensuring access is limited to authorized cloud resources. However, if a developer assigns excessive permissions to the cloud credentials, an attacker might exploit the leaked credentials to perform unauthorized operations, potentially compromising the security of multiple applications and their users. Vertically, we collected the earliest version of the APK and intermediate versions up to the latest release for analysis. Since early and updated versions of an application may share similar functionalities, earlier versions may have weaker security measures. For instance, WeChat has over 200 versions[4], some of which may exhibit weaker security, such as hardcoding sensitive information within the APK file.

4.2 Identification of Cloud Initialization APIs and Cloud Credentials

To systematically detect whether an app uses cloud credentials, the types of cloud credentials used, and the specific patterns of their usage, we initially focus on automated recognition and analysis of cloud-related initialization APIs and the cloud credentials they employ within the app. As discussed in the previous section, initialization APIs are used to configure the environment and parameters necessary for mobile applications to access cloud services. These APIs use either long-term cloud credentials or temporary credentials as parameters to engage in unidirectional or bidirectional authentication interactions with the cloud service. Based on the above observations, we utilize Soot[1] to transform the mobile app into a Jimple Intermediate Representation (IR) suitable for analysis. We further analyze the Soot Methods to identify the presence of cloud-related initialization APIs. Subsequently, we construct a global call graph of the mobile app and perform backward data flow analysis to identify cloud credentials and parameters related to cloud services.

In cases where cloud credentials are stored on the server side, retrieving them directly from the code using the aforementioned steps becomes infeasible. Instead, analyzing HTTP/HTTPS requests is required to extract relevant credentials or configuration information. Specifically, we investigate how network APIs, such as HttpClient.execute() and HttpURLConnection.connect(), are invoked to track the data flow dependencies between HTTP requests and their corresponding responses. In backward data flow analysis, HTTP response objects are treated as taint sources, aiding in identifying request parameters such as the URI, HTTP method (e.g., GET or POST), query strings, headers, and body. The detailed process is illustrated in Algorithm 1.

In this project, we addressed the following three challenges: Firstly, addressing the analysis barriers posed by code obfuscation techniques used in some apps. These techniques often obfuscate class names and method names of third-party SDK libraries, converting them into meaningless characters, rendering API name recognition ineffective for detecting obfuscated Cloud APIs. To tackle the challenge of code obfuscation, we employed API signature methods.

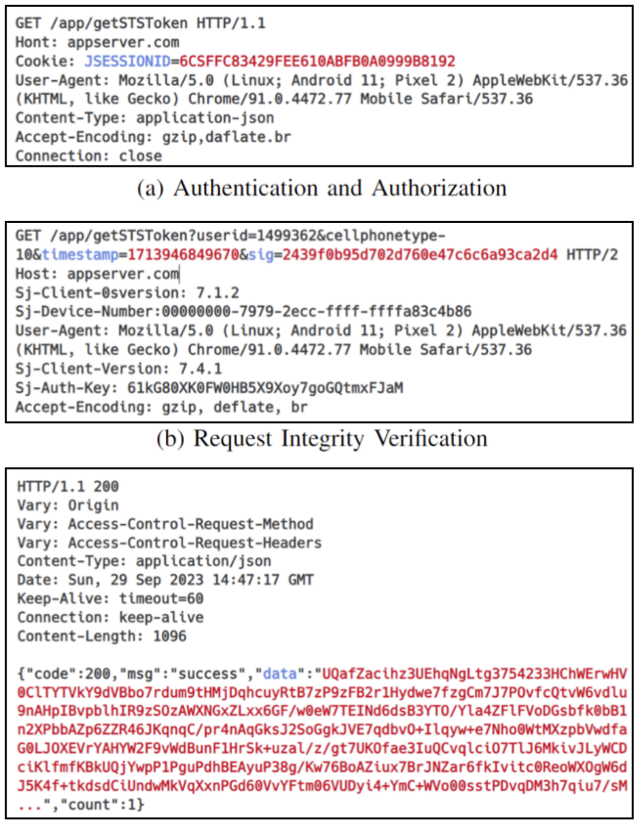
Secondly, precisely analyzing the asynchronous callback function relationships when constructing the global call graph. Some apps, when initiating network requests to cloud backend services to retrieve cloud credential values in real-time, use asynchronous callback mechanisms to avoid thread blocking. FlowDroid[22] does not include control flows related to asynchronous callbacks due to the incomplete handling of callback mechanisms. Therefore, we utilized the results of Edgeminer[24], which addresses issues with asynchronous and implicit events and identified 19,647 additional callbacks, whereas FlowDroid only identified 181 callbacks. By incorporating Edgeminer's results into FlowDroid's configuration files, we generated an extended call graph of the mobile apps. When intra-procedural data flow analysis failed to identify specific cloud credential values, inter-procedural data flow analysis was employed, utilizing an extended call graph to compute the corresponding data

Algorithm 1 Cloud Credential Identification

```
Input: $f_c$: Initialize cloud API; $f_b$: Android Callback API; $A_d$: apps
       using cloud services;
Output: $A_k$: Cloud Credential String Value; $H_r$: HTTP request
        related strings
  Function Process($M_s$, $s$):
 1: for apps $\in A_d$ do
 2:     $C_g \leftarrow$ BuildCFG($apps$, $f_b$)
 3:     for $C_a \in f_c$ do
 4:         $f_s \leftarrow$ gensig($C_a$)
 5:         if $f_s \in C_g$ then
 6:             $d_g \leftarrow$ BuildDDG($f_s$, $C_g$)
 7:         end if
 8:         if HttpResponse $\in d_g$ then
 9:             Type, url, args $\leftarrow$ ComputeHttpRequest($d_g$)
10:             $H_r \leftarrow$ Type
11:             $H_r \leftarrow$ url
12:             $H_r \leftarrow$ args
13:             return $H_r$
14:         else
15:             $A_k \leftarrow$ ComputeStringValue($d_g$)
16:             return $A_k$
17:         end if
18:     end for
19: end for
```

dependency graph. By tracing the instructions in the data dependency graph, we ultimately obtained the cloud credential string values or the HTTP/HTTPS request-related string values.

Third, we examine the protection mechanisms employed by applications for Web APIs, with the goal of obtaining temporary credentials. To effectively prevent malicious users from accessing Web APIs, applications implement multiple protection methods. The most common methods[29][34][30] include: authentication and authorization, encrypted communication, and request integrity verification. As shown in Figure 3(a), authentication, typically implemented using OAuth or API keys, ensures that only legitimate users can access the API. As depicted in Figure 3(b), for encrypted communication, besides using the HTTPS/TLS protocol, some apps implement additional encryption (e.g., AES) on the data. As illustrated in Figure 3(c), for request integrity verification, some apps incorporate digital signatures or random numbers into the requests to resist man-in-the-middle and replay attacks. To overcome these challenges, we employ dynamic analysis to obtain temporary credentials. Specifically, we manually register for user credentials, using methods such as phone number registration or third-party OAuth. To manage encrypted communication and integrity verification, we install BurpSuite certificate files and Frida on a Pixel device running Android 9.0. By hooking into the Web APIs that generate encryption or integrity verification parameters, we gain access to these APIs to obtain temporary credentials.



```
GET /app/getSTSToken HTTP/1.1
Hont: appserver.com
Cookie: JSESSIONID=6CSFFC83429FEE610ABFB0A0999B8192
User-Agent: Mozilla/5.0 (Linux; Android 11; Pixel 2) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/91.0.4472.77 Mobile Safari/537.36
Content-Type: application-json
Accept-Encoding: gzip,daflate.br
Connection: close
```

(a) Authentication and Authorization

```
GET /app/getSTSToken?userid=1499362&cellphonetype-
10&timestamp=1713946849670&sig=2439f0b95d702d760e47c6c6a93ca2d4 HTTP/2
Host: appserver.com
Sj-Client-0sversion: 7.1.2
Sj-Device-Number:00000000-7979-2ecc-ffff-ffffa83c4b86
User-Agent: Mozilla/5.0 (Linux; Android 11; Pixel 2) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/91.0.4472.77 Mobile Safari/537.36
Sj-Client-Version: 7.4.1
Sj-Auth-Key: 61kG80XK0FW0HB5X9Xoy7goGQtmxFJaM
Accept-Encoding: gzip, deflate, br
```

(b) Request Integrity Verification

```
HTTP/1.1 200
Vary: Origin
Vary: Access-Control-Request-Method
Vary: Access-Control-Request-Headers
Content-Type: application/json
Date: Sun, 29 Sep 2023 14:47:17 GMT
Keep-Alive: timeout=60
Connection: keep-alive
Content-Length: 1096

{"code":200,"msg":"success","data":"UQafZacihz3UEhqNgLtg3754233HChWErwHV
0ClTYTVkY9dVBbo7rdum9tHMjDqhcuyRtB7zP9zFB2r1Hydwe7fzgCm7J7POvfcQtvW6vdlu
9nAHpIBvpblhIR9zSOzAWXNGxZLxx6GF/w0eW7TEINd6dsB3YTO/Yla4ZFlFVoDGsbfk0bB1
n2XPbbAZp6ZZR46JKqnqC/pr4nAqGksJ2SoGgkJVE7qdbvO+Ilqyw+e7Nho0WtMXzpbVwdfa
G0LJOXEVrYAHYW2F9vWdBunF1HrSk+uzal/z/gt7UKOfae3IuQCvqlciO7TlJ6MkivJLyWCD
ciKlfmfKBkUQjYwpP1PguPdhBEAyuP38g/Kw76BoAZiux7BrJNZar6fkIvitc0ReoWXOgW6d
J5K4f+tkdsdCiUndwMkVqXxnPGd60VvYFtm06VUDyi4+YmC+WVo00sstPDvqDM3h7qiu7/sM
...","count":1}
```

(c) Response Data Encryption

Figure 3: Protection Methods for Web APIs

4.3 Analysis of Cloud Access Vulnerabilities

Having discussed how to obtain and identify cloud credentials, this Section addresses ways to detect data leakage and cloud resource attacks leveraging obtained cloud credentials.

In terms of data breach attacks, we focus on the leakage of PII and user credential information. Various types of data are stored across different cloud services. For instance, cloud storage often holds user-uploaded media and crucial operational files, which may also include data from web apps and backup files. Credentials with excessive permissions can thus facilitate broader data exposure. Additionally, cloud logs, which store vital system operation and user activity details, if compromised, can provide attackers with access to user browsing records and more. Notably, we identified a new risk of unauthorized user logins stemming from cloud log data leaks, where user tokens could be misused by attackers to impersonate others. This scenario is further explained in Section 5.4.2.

Regarding cloud resource attacks, we focus on operations that could disrupt the normal functioning of cloud services, such as data pollution, RCE. Specifically, we check whether the cloud credentials can use APIs related to cloud resource operations, including the management of cloud storage buckets and server instances.

To detect these attacks, we assess the potential risks by analyzing the cloud credentials' access to resource APIs. Solely examining the called APIs might overlook sensitive ones, thus underestimating the threat level. Hence, we initially identify powerful List-type APIs in various cloud services, such as Alibaba Cloud's oss:ListBuckets and oss:ListObjects, which list all buckets and their contents, respectively. Due to their potency, these APIs are typically restricted to specific operations. Then, we further analyze APIs involved in reading, writing, and deleting operations, along with other sensitive resource access APIs.

To identify the presence of vulnerabilities, we propose a novel privilege detection method to assess the extent of data and resource exposure resulting from cloud credential leaks. It is worth noting that this method does not actually attack the cloud service. Initially, we use identity authentication APIs provided by CSPs, such as the GetCallerIdentity API, to determine if the identified credentials are root, regular, or another type. Subsequently, we execute user permission policy query APIs within IAM services to ascertain the actual permissions of the identified credentials, analyzing the potential extent of leaks and attacks based on the query results.

In cases where regular user credentials or temporary access credentials cannot invoke IAM-related APIs, we indirectly detect cloud credential permissions through dynamic testing. Since executing sensitive resource access APIs successfully could lead to data contamination or leakage within the cloud services, we verify the permissions without successful execution. Specifically, we employ the technique of inducing errors[37] to test whether these APIs can be executed. Cloud services typically follow a specific sequence to determine if a resource access API can be executed. By providing incorrect cloud resource IDs at certain stages to induce errors, we can infer whether the cloud credentials can execute the corresponding API.

Using these methods, Cloudet conducts vulnerability attack analysis on identified cloud credentials and their associated cloud services. Principally, once a cloud credential is identified, the app is considered at risk. However, in this study, we classify an app as vulnerable (Vul App) only if Cloudet confirms that they can be attacked by means such as data pollution, PII leakage, RCE, and fake user login.

5 Results

In this section, we address the research questions related to the data collected in our study. All the applications analyzed in our research were executed on a dedicated PC equipped with an AMD Ryzen 7 5800H @ 3.20 GHz processor and 32 GB of memory. To answer our research questions, we utilized Cloudet to analyze the cloud credentials of all 21,724 applications. The detection time was proportional to the size of the APK files, which ranged from 195 KB to 272 MB. The runtime varied from three minutes for smaller applications to a maximum of two hours for larger ones. We manually analyzed 453 Web API instances used to obtain cloud credentials. To acquire cloud credentials, we registered user accounts for 232 applications. For the applications with Web API protection, we used Frida to hook 73 applications, successfully obtaining 69 temporary credentials and 4 long-term credentials from the server.

5.1 The Usage and Security of Temporary Credentials

Using Cloudet to identify cloud credentials for each application, we analyzed the security of cloud credentials across 21,724 apps, with the results summarized in Table 2. Our analysis revealed that 893 apps interacted with at least one of the three evaluated cloud services. From these apps, we extracted 945 distinct cloud credentials—comprising 79 root, 463 standard, and 403 temporary credentials from 860 apps. Some apps contained multiple credentials, particularly those with different access levels, which explains why the total number of credentials exceeded the number of apps involved. For an additional 53 apps, we detected cloud API calls but were unable to obtain the corresponding cloud credentials. This was due to various reasons, including the failure of the app's Web API interface, the inability of regular app users to access the corresponding Web API, and cloud credentials being stored in .so files.

Further categorization analysis of apps revealed that Shopping, Social, and Music & Audio apps were the largest users of cloud services (Figure 7), likely due to their reliance on realtime data exchange and extensive data storage. Conversely, Finance apps were least likely to utilize cloud services, possibly due to stringent data security and privacy regulations, favoring private over public cloud solutions.

Column 4 of Table 2 shows the proportion of temporary access credentials among all cloud credentials, with temporary credentials accounting for 42.6%, nearly half. Among the 860 apps analyzed, approximately 53% used hard-coded cloud credentials, 4% obtained long-term credentials via private app servers, and 43% acquired temporary credentials through private servers. Figure 4 illustrates the distribution of different types of cloud credentials. Notably, 3(0.3%) root and 21(2.2%) regular user credentials and all temporary credentials were sourced from the apps' private server.

Following the method outlined in Section 4.2, we manually reconstructed the Web API information for 453 applications to obtain cloud credentials, including 403 temporary credentials and 24 long-term credentials. This process was based on the HTTP/HTTPS request information identified by Cloudet. As shown in Figure 5, we found that 354 (78.1%) applications lacked sufficient protection for their Web APIs. Among these, 52 Web APIs could be accessed via a simple GET request without any additional parameters, while 92 Web APIs had specific parameter format requirements but did not strictly verify the authenticity of these parameters. These vulnerabilities pose significant security risks, as developers cannot perform trace analysis even if attackers obtain the cloud credentials. Additionally, 210 applications implemented user authentication for their Web APIs, but any registered app user could still access and obtain temporary credentials.

The remaining 99 applications (21.9%) provided better protection for their temporary credentials. Among these, 26 applications required registration via high-privilege accounts (e.g., VIP users), preventing us from obtaining the cloud credentials for this subset of apps. In addition, 73 applications performed integrity verification or encryption of HTTP/HTTPS responses. Specifically, 44 applications added extra parameter verification to ensure the integrity of HTTP/HTTPS requests initiated by the client, while 12 applications performed symmetric encryption (e.g., AES or DES) on both

Table 2: Overall results

Cloud Services	Root Credentials	Vul Apps	Regular Credentials	Vul Apps	Temporary Credentials: No Protection	Temporary Credentials: Authentication	Temporary Credentials: C & I	Vul Apps	All
S3	12(3.8%)	12	201(64.6%)	65	31(9.9%)	45(14.4%)	22(7.0%)	17	311
ClousWatch	0(0%)	0	6(50%)	1	3(25%)	3(25%)	0(0%)	1	12
SNS	2(4.7%)	1	31(73.8%)	3	4(9.5%)	4(9.5%)	1(2.3%)	1	42
OSS	42(10.6%)	42	156(39.4%)	77	71(17.9%)	105(26.5%)	21(5.3%)	61	395
SLS	2(6.8%)	2	13(44.8%)	7	6(20.6%)	5(17.2%)	3(10.3%)	7	29
MNS	0(0%)	0	1(100%)	0	0(0%)	0(0%)	0(0%)	0	1
COS	21(15.4%)	21	46(33.8%)	23	25(36.2%)	32(46.3%)	22(31.8%)	22	146
CLS	0(0%)	0	3(100%)	1	0(0%)	0(0%)	0(0%)	0	3
CMQ	0(0%)	0	6(100%)	2	0(0%)	0(0%)	0(0%)	0	6
Overall	79(8.4%)	75	463(49.0%)	169	140(14.8%)	194(20.5%)	69(7.3%)	109	945

HTTP/HTTPS requests and responses. Although these protective measures partially safeguarded cloud credentials from malicious users, our manual analysis using Frida allowed us to retrieve all cloud credentials by extracting encrypted symmetric keys or bypassing integrity verification.

These findings address **Q1**, indicating that currently, less than half of the developers employ the recommended security strategies for temporary credentials. Most applications utilizing cloud services still rely on the primitive method of hardcoding cloud credentials. Additionally, only a minority of developers have implemented adequate protection for their Web APIs. Malicious users can use reverse engineering techniques to obtain cloud credentials stored on the server, thereby facilitating further attacks.

5.2 Cloud Service Vulnerability Identification

For the acquired cloud credentials, we conducted dynamic analysis of their permissions and found that 872 out of 945 credentials were valid, while the remaining 73 had been rectified by the developers prior to our testing. All temporary credentials dynamically distributed by cloud services proved to be valid. Using the method described in Section 4.3, we evaluated the permissions of the cloud credentials without accessing users' sensitive data. Figure 6 illustrates the proportion of different types of cloud credentials that were found to be vulnerable. An application is considered vulnerable if it exhibits any of the four security risks described in Section 4.C. Among all credentials, 41% of regular user credentials were found to be vulnerable, whereas the proportion for temporary credentials was 27%. Additionally, we did not find any cases of temporary credentials with access to cloud servers or databases, while such issues were frequently observed with long-term credentials. This is attributed to developers failing to distinguish between the use of cloud credentials in mobile applications and other types of applications. These findings suggest that the use of temporary credentials effectively reduces the occurrence of vulnerabilities.

Although these credentials are temporary, they can still pose significant security risks if granted excessive permissions. Our analysis indicates that a substantial portion of temporary credentials (27%) were granted full administrative control over their respective services, allowing operations such as listing and reading all files across multiple cloud storage buckets or reading and writing all files within a single bucket. The remaining 73% of credentials were assigned more appropriate permissions. However, malicious users who successfully obtain these temporary credentials may still be able to perform write operations, leading to issues such as data pollution. These findings underscore the importance of proper credential configuration by developers, who may mistakenly assume that using temporary credentials alone is sufficient for security, thereby neglecting critical permission settings.

These issues affect a wide range of cloud services, with the identified vulnerabilities impacting 356 applications. The affected services are categorized as follows: cloud servers in 34 applications, cloud storage in 321 applications, cloud databases in 29 applications, cloud logging in 19 applications, and cloud push notification services in 21 applications, as shown in figure 8. To address potential security risks, we analyzed whether these applications were susceptible to the specific types of attacks outlined in Section 4.3, and used case studies to provide more detailed insights into the security flaws.

The above analysis addresses **Q2**, indicating that the temporary credentials recommended by major cloud service providers can only partially mitigate security issues. If developers do not assign appropriate permissions or adequately protect cloud credentials, data leakage and other risks may still occur. Furthermore, leaked cloud credentials from mobile applications can lead to security vulnerabilities in cloud servers and cloud databases, resulting in more severe consequences.

5.3 Extended dataset analysis

This study enhances the depth and breadth of the analysis by vertically expanding the dataset to include historical versions, regional variants, and associated applications. However, this extension introduces potential threats to the validity of the study. First, applications in historical versions may contain more security vulnerabilities because developers' security awareness may have increased over time. Second, regional variants may exhibit differences in security practices due to localization requirements or varying regulatory demands. Finally, associated applications, which share the same cloud credentials, may experience security vulnerabilities in one application affecting others. These factors may impact the generalizability of the findings.

To assess the impact of the vertical dataset extension on the research results, we compared the distribution of poor practices and vulnerabilities between the non-expanded and expanded datasets.

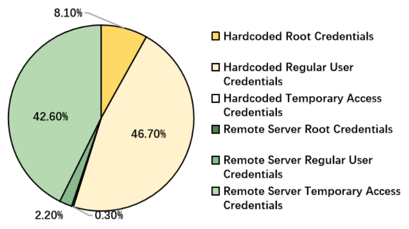


Figure 4: Proportion of cloud credentials from hardcoding or private server

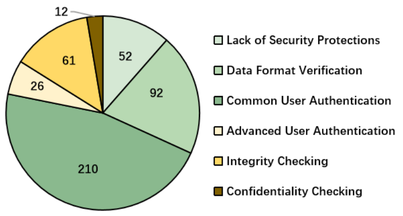


Figure 5: WEB API security analysis for obtaining cloud credentials

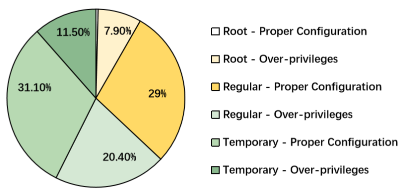


Figure 6: Proportion of cloud credentials permission configuration

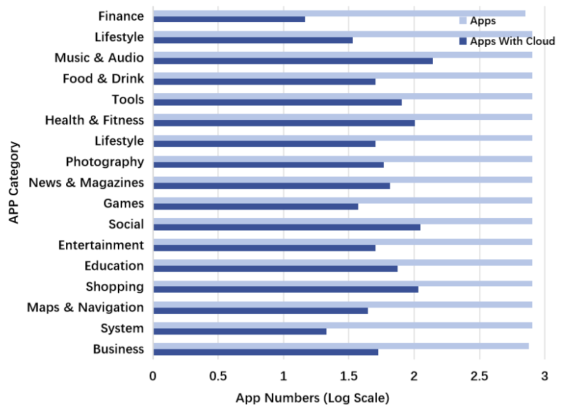


Figure 7: Distribution of Cloud Service Usage by Different Types of Applications

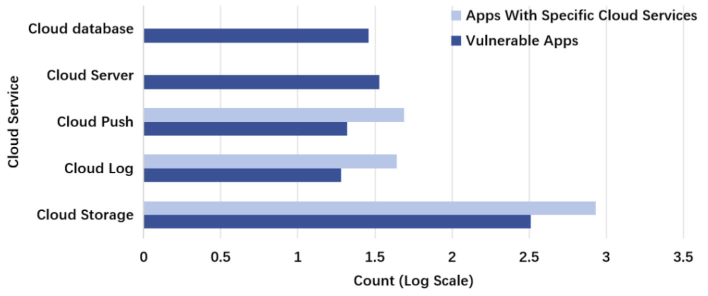


Figure 8: Proportion of Applications with Vulnerabilities Using Specific Cloud Services

Table 3 presents the distribution of hardcoded credentials, temporary credential usage, and improper permission configurations across both datasets. The results show that applications in historical versions are more likely to use hardcoded credentials (65%), while this percentage drops to 47% in the latest versions. Additionally, applications in regional variants exhibit significant differences in improper permission configurations. In some regions, applications place more emphasis on privacy protection due to localization requirements, resulting in a lower proportion (12%) of improper permission configurations.

Distribution of PII leakage, fake login, and remote code execution (RCE) vulnerabilities in the unexpanded and expanded datasets. The proportion of PII leakage and RCE vulnerabilities is significantly higher for historical versions of the application than for the latest version, while the proportion of fake login vulnerabilities is higher for regional variants. These results show that vertical expansion of the dataset does affect the distribution of research results, but it also provides a more comprehensive view of the evolution of bad practices and vulnerabilities.

5.4 Case Studies

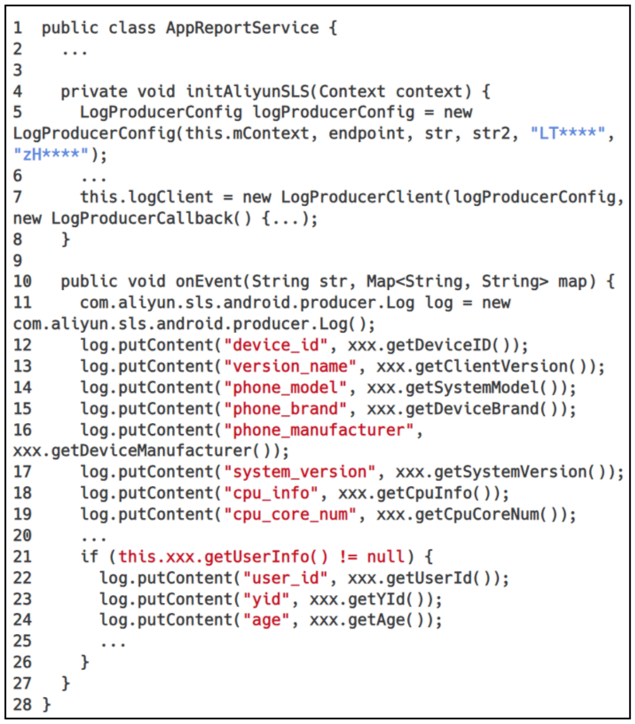
5.4.1 ***Exposure of PII***. Many apps on smartphone keep track of personal sensitive data Our analysis found a video app, com. Anonymized1, which utilizes Alibaba Cloud storage services to collect user-uploaded images and other data. It was noticed that developers had hardcoded AWS standard user credentials within the app with excessively high permissions, such as s3:ListBuckets. This permission enabled anyone possessing these credentials to access and list all files within the storage buckets. Our examination of 121 bucket names and filenames in com.Anonymized1 revealed multiple instances of exposure to PII. Despite the app uploading files to only three buckets, other buckets contained data from additional apps or sensitive data like system backups. We have informed the developers of these findings via email and provided repair suggestions on the restoration.

5.4.2 ***Fake User Login***. An increasing number of apps are inclined to use cloud logging services to collect in-app information. However, our research reveals that many apps neglect data security and privacy protections during the logging process, leading to potential serious security risks. Figure 9 illustrates a typical unauthorized user login caused by cloud log data leakage. The APK first calls the initialization API using cloud credentials (Line 5). Subsequently, in the onEvent method, it logs information such as device model and IP address (Lines 12-19). If the app user is already logged in, it further uploads user information to the log (Lines 21-24).

For instance, in our evaluation of a social media app, com. Anonymized2, the app utilized temporary credentials to upload application execution and user actions data to Alibaba Cloud's logging database. Crucially, the app stored unfiltered HTTP request messages in the logs, exposing sensitive user information and JWT tokens used for

Table 3: Expanded dataset distribution

Category	Non-Expanded Dataset	Historical Versions)	Regional Variants
Hardcoded Credentials	47%	65%	52%
Temporary Credentials	53%	35%	48%
Improper Permission Configurations	40%	55%	12%



```
public class AppReportService {
  ...

  private void initAliyunSLS(Context context) {
    LogProducerConfig logProducerConfig = new LogProducerConfig(this.mContext, endpoint, str, str2, "LT****", "zH****");
    ...
    this.logClient = new LogProducerClient(logProducerConfig, new LogProducerCallback() {...});
  }

  public void onEvent(String str, Map<String, String> map) {
    com.aliyun.sls.android.producer.Log log = new com.aliyun.sls.android.producer.Log();
    log.putContent("device_id", xxx.getDeviceID());
    log.putContent("version_name", xxx.getClientVersion());
    log.putContent("phone_model", xxx.getSystemModel());
    log.putContent("phone_brand", xxx.getDeviceBrand());
    log.putContent("phone_manufacturer", xxx.getDeviceManufacturer());
    log.putContent("system_version", xxx.getSystemVersion());
    log.putContent("cpu_info", xxx.getCpuInfo());
    log.putContent("cpu_core_num", xxx.getCpuCoreNum());
    ...
    if (this.xxx.getUserInfo() != null) {
      log.putContent("user_id", xxx.getUserId());
      log.putContent("yid", xxx.getYId());
      log.putContent("age", xxx.getAge());
      ...
    }
  }
}
```

Figure 9: Cloud Log Data Breach

authentication. Such exposure allows malicious actors to forge identities and gain control over user accounts, significantly heightening the risk of data breaches. Additionally, the real-time data collection and automatic upload mechanisms of cloud log services exacerbate the potential for data breaches, particularly in social apps where impersonation could lead to serious personal and relational damage. It highlights the necessity of adhering to data protection regulations like GDPR or CCPA during personal information collection through cloud logs.

5.4.3 *Data Pollution Attacks.*

Even cloud credentials with limited permissions can cause severe damage if they allow write operations. In our study of a children's app, com.Anonymized3, the developer used Tencent Cloud storage services to manage multimedia content required for app functionality and uploaded personal information using temporary credentials. Although permissions were minimally assigned, the lack of proper bucket segmentation allowed malicious files to overwrite necessary app data. This vulnerability poses significant risks to children's mental health, as exposure to inappropriate or harmful content can adversely affect their psychological development and behavior.

5.4.4 *Remote Code Execution on Cloud Servers.*

As mobile apps increasingly rely on cloud servers, new security vulnerabilities emerge, particularly with exposed cloud credentials. In the case of com.Anonymized4, a utility app for managing phone files and cleaning photos, it hardcoded root user cloud credentials in its resources, granting complete access to cloud resources. This misconfiguration exposed cloud server APIs to potential misuse, including malicious activities or data breaches. The app used Alibaba Cloud APIs for managing photos and backups, with dangerous permissions like ecs:StopInstance and ecs:DeleteInstance, allowing attackers to shut down or delete cloud instances. Data leakage risks were linked to ecs:RunCommand, enabling remote code execution (RCE). Ethically, we did not attempt these actions on live servers.

5.5 Lessons

We distributed questionnaires to 16 mobile application developers contacted through public channels and received 15 valid responses. These developers primarily came from companies or teams engaged in mobile application development. As the purpose of this survey was to gather developers' awareness of cloud credential security practices and was constrained by the principles of voluntary participation and anonymity requirements, we did not collect specific demographic data about the developers (e.g., age, geographic location, specific company names, etc.) to ensure their privacy and encourage candid responses.

This survey aimed to explore developers' awareness, practices, and challenges regarding cloud service credential management. The questionnaire covered the following 11 questions:Which cloud service provider do you use? Are you aware of and have you used temporary credentials (STS tokens)? Do you use long-term credentials or temporary credentials in your mobile applications? Do you store cloud credentials in the client-side app or on the server-side? Are you aware of and have you implemented permission policies for cloud credentials? Do you check whether cloud credentials have been over-permissioned? Are you aware of how to handle leaked cloud credentials? Do you regularly check and update your cloud credentials? Have you provided training or guidance on cloud credential security for your development team? Do you have strategies to manage cloud credential differences across regions? Do you configure different levels of cloud credential permissions for different roles or departments?

Through these questions, we sought to understand the current state of cloud credential management from the developers' perspective to supplement our findings from system detection.

5.5.1 *Root Causes Behind Developer Practices.*

The survey results show that among the 15 interviewed developers, 60% (9) admitted to reducing security measures due to project time constraints, with 46.7% (7) engaging in high-risk practices such as hardcoding credentials. Additionally, only 40% (6) accurately understood the security advantages of temporary credentials, with adoption rates varying by cloud provider (Alibaba Cloud 53.3%, AWS 33.3%). Regarding permission management, 66.7% (10) considered IAM configuration overly complex, with key challenges including ambiguous permission boundaries (53.3%), difficulties in debugging policy syntax (46.7%), and cross-service permission conflicts (40%).

According to our findings, the main challenges developers face can be summarized as follows:

Time and Resource Constraints: Developers often face tight development cycles and limited resources, especially in the early stages of mobile app development. To release the app as quickly as possible, developers tend to overlook necessary security measures. For example, hardcoding cloud credentials or simplifying IAM permission configurations have become choices to speed up the development process. While these practices help enhance development efficiency, they significantly increase the risk of security vulnerabilities.

Lack of Awareness of Security Measures: Many developers do not fully understand the security advantages of temporary credentials. Although temporary credentials effectively reduce the risk of credential leakage, many developers still do not commonly adopt this mechanism. One reason for this is that the configuration of temporary credentials is relatively complex, and developers lack sufficient guidance and tools to correctly implement them. Some developers prefer to use long-term credentials or directly hardcode credentials, as this approach, though insecure, is more convenient for debugging and testing, especially in the early stages of development.

Complexity of IAM Permission Configuration: Developers often find configuring IAM permissions confusing, especially when it comes to fine-grained permission management in cloud platforms. Despite the cloud service providers offering detailed documentation and guidelines, the complexity of permission configurations makes it difficult for developers to ensure that each permission setting adheres to the principle of least privilege.

5.5.2 *Mitigation Strategies.*

To help developers overcome these challenges and improve their security practices, we propose the following mitigation strategies:

Fine-grained permission division: Developers should implement fine-grained permission partitioning for each user or role, adhering to the principle of least privilege. For instance, different permission sets should be assigned to distinct user groups, ensuring team members only receive necessary access rights based on their specific functions.Concurrently, cloud platforms should provide clear documentation on granular permissions, including interactive permission matrices and policy simulators, along with detailed configuration examples—such as risk-level annotations for permissions and comparative explanations of commonly confused permissions—to help developers accurately understand the scope and risks of each permission item.Furthermore, both parties should collaborate to establish a dynamic feedback mechanism, enabling continuous optimization of the permission management system.

Simplify Cloud Service Configuration Processes: To ease the burden on developers when configuring temporary credentials and IAM permissions, cloud service providers should offer more user-friendly tools and SDKs that automate the process of generating temporary credentials and configuring permissions. By simplifying the IAM permission configuration process, developers can more intuitively understand and apply best security practices, reducing the likelihood of human errors.

Enhance Security Awareness Training: Regular training and security guidance should be provided to enhance developers' understanding of cloud credential security, the advantages of temporary credentials, and IAM permission configuration. This can be achieved through online learning platforms, case studies, and practical demonstrations, helping developers master the basics of secure configurations and placing greater emphasis on credential protection.

Conduct Developer Research and Collaboration: To address the specific difficulties developers encounter when configuring IAM permissions, further user research, including interviews and surveys, is recommended to gather more detailed feedback. Based on this feedback, cloud service providers can design more user-friendly permission configuration wizards or automation tools to help developers configure cloud services more efficiently and securely.

These mitigation strategies will not only help developers reduce security risks encountered during the use of cloud services but also raise their awareness of cloud security issues, enabling them to better tackle new challenges that may arise in the future.

5.6 Vulnerability Disclosure

Our vulnerability disclosure process utilized a multi-channel and phased approach to maximize the coverage and rectification of vulnerabilities. We initially engaged developers directly using the contact details provided on application marketplaces. For those who did not respond, we escalated our outreach to organizations such as CERT[10], the China National Vulnerability Database (CNVD)[27], and the China Application Vulnerability Disclosure Platform (CAPPVD) [25].

Adhering to responsible disclosure protocols, we will refrain from publicly releasing any details about unresolved vulnerabilities until they are adequately addressed by the developers. We are encouraged by the response from some developers who have acknowledged the vulnerabilities we reported and have commenced remedial actions. Updates on these developments, along with developer feedback and letters of appreciation, are regularly posted on our blog, fostering an ongoing dialogue on improving app security.

6 DISCUSSION

In this study, we introduce Cloudet, a semi-automated detection system designed to identify cloud credential leakage risks in mobile applications. To validate Cloudet's effectiveness and performance, we conducted a comprehensive evaluation and compared it with existing state-of-the-art methods. The evaluation covered key performance indicators, including detection accuracy, operational efficiency, and system stability.

6.1 Operational Efficiency Comparison

To assess Cloudet's operational efficiency, we benchmarked it against two representative existing tools: PrivRuler[37] and LeakScope[42].

Our experiment involved 100 randomly selected APK files containing cloud SDKs, with installation package sizes ranging from a few hundred KB to over 100 MB, simulating real-world applications of varying scales.

The results show that Cloudet processed each application in an average of 28 minutes, while PrivRuler and LeakScope took an average of 18 minutes and 20 minutes, respectively. Further analysis by application size revealed that for apps smaller than 10 MB, Cloudet's average processing time was 8 minutes, compared to 5 minutes for PrivRuler and 6 minutes for LeakScope. For larger applications exceeding 50 MB, Cloudet averaged about 65 minutes, while PrivRuler took 45 minutes and LeakScope reached 51 minutes. This difference is primarily due to Cloudet's integration of an additional server-side parameter identification mechanism during analysis, which enhances detection depth and breadth. Therefore, Cloudet achieves more comprehensive risk coverage while maintaining efficiency.

6.2 Comparison with State-of-the-Art Methods

In terms of credential detection, both PrivRuler and LeakScope primarily focus on long-term credentials directly embedded in application code. PrivRuler supports AWS, Google Cloud, and Azure, while LeakScope targets AWS, Azure, and Alibaba Cloud.

In contrast, Cloudet's design is broader, covering not only the identification of long-term credentials but also specifically strengthening its ability to detect temporary credentials. Using the same 100 application samples:PrivRuler detected 12 root credentials and 35 standard credentials. LeakScope detected 14 root credentials and 32 standard credentials. Cloudet identified a total of 13 root credentials, 33 standard credentials, and 23 temporary credentials.

It's noteworthy that PrivRuler and LeakScope, due to the limitations of their static analysis strategies, failed to effectively identify any temporary credentials. This demonstrates Cloudet's advantage in the range of credential types covered.

6.3 False Positive and False Negative Rates

To evaluate system accuracy, we engaged two experienced security researchers to independently manually verify the detection results using a standardized verification process and toolchain.

Out of a total of 945 detected cloud credentials, 912 were confirmed as valid after manual review (accounting for 96.5%), while the remaining 33 were identified as false positives (a false positive rate of 3.5%).

Additionally, through an in-depth audit of 100 randomly selected APK files, we discovered 15 real credentials that were not detected, corresponding to a false negative rate of approximately 1.8%. Further analysis revealed that false positives primarily stemmed from:12 instances of strings stored using non-standard encryption methods. 9 instances of obfuscated data within .so files. The remaining false positives were composed of pseudo-credentials found in resource files or log information.

7 Related Works

7.1 App Cloud and Other Vulnerabilities Detection

Previous research has extensively studied the issue of cloud credential leakage in mobile applications. Zuo et al.[42] analyzed and found that the misuse of root credentials is a major cause of data leakage in the cloud. Wang et al. focused on the excessive misuse of cloud credentials, revealing that regular user credentials with excessive permissions are another factor contributing to data leakage in the cloud. Furthermore, Wang et al.[37] , in their study of 11,891 applications using cloud services, discovered that 2,474 applications (20.8%) failed to extract cloud credentials, indicating that these applications might employ dynamic credential acquisition methods, such as temporary access credentials. Our study firstly conduct a large-scale investigation into the accessibility and security of temporary credentials in the context of dynamic credential acquisition.

Additionally, mobile applications have been widely exposed to the issue of different kinds of hardcoded credentials. Zhou et al.[41] successfully extracted and analyzed credential leakage problems in mobile applications using the data flow analysis tool CredMiner. Wen et al.[38] summarized credential misuse patterns in apps, particularly focusing on SDK credential abuse issues. These patterns include embedding credentials directly into application code, storing credentials using outdated or insecure methods, and insufficient verification and access control mechanisms. Shi et al.[36] examined credential leakage issues in mobile payment applications and how leaked payment credentials can be exploited.

As mobile backends have evolved, the focus of many large-scale studies has shifted towards mobile apps. In 2019, Zuo et al.[20] conducted a thorough survey of mobile backend systems, identifying 983 N-day and 655 0-day vulnerabilities across the top 5,000 free apps in the Google Play Store. This research underscored a general lack of clarity among developers and cloud providers regarding their responsibilities for securing mobile backends, leading to numerous security gaps. Shangcheng Shi's[35] work on insecure Single Sign-On (SSO) implementations found that 72% of over 500 apps reviewed had incorrect SSO implementations. Yue Zhang and colleagues[40] further explored how early APK versions can serve as attack vectors, compromising the security of later app versions, with about 34% of 1,500 apps showing vulnerabilities, including high-profile apps like Facebook.

7.2 Cloud Security Vulnerabilities Detection

Cloud security remains a critical area of focus within the IT field, gaining prominence with the rapid advancement and adoption of cloud computing technologies.

Xiao et al.[39] systematically reviewed the major security and privacy challenges in cloud computing, analyzing existing solutions and their limitations. Armbrust et al.[21] discussed in detail the technological driving factors, potential security issues, and challenges faced by cloud computing. Kaufman et al.[33] analyzed the application of encryption technology, access control, and data integrity verification in protecting cloud data security, and pointed out the limitations of these technologies in terms of performance and scalability. Continella et al.[28] investigated the use of Amazon S3 by generating and analyzing 240,461 bucket names, identifying 191 websites vulnerable to attacks. This study highlighted the inherent risks associated with cloud storage buckets. Jack Cable et al.[23] conducted an extensive study on the naming conventions of storage buckets and performed a detailed analysis of the types of sensitive information stored therein. The above work also helps us analyze security vulnerabilities in the cloud.

8 Conclusion and Discussion

We propose Cloudet, a semi-automated system for comprehensively identifying various cloud credentials in mobile applications and detecting cloud vulnerability risks. Through Cloudet, we conducted extensive analyses of applications using nine types of mBaaS cloud services and cloud infrastructure services, including cloud servers and databases, from three major cloud providers. This enabled us to address two critical security concerns.

For Q1, What is the current usage rate of these security measures across various apps, and are they widely adopted and valued by developers?we found that more than half of the cloud credentials are insecurely hard-coded. Furthermore, non-hardcoded temporary credentials often fail to adhere to the principle of least privilege. This indicates that the usage rate of IAM security services by cloud service providers in current mobile applications needs significant improvement.

For Q2, Do the existing security measures continue to pose potential security risks, and do existing security measures cover all potential security risks of cloud backend services in mobile app? we demonstrated the ability to bypass and attack the recommended temporary access credential mechanism, verifying that it can lead to severe cloud access vulnerabilities. This shows that the recommended mechanism still poses significant security risks.

In summary, our findings suggest substantial room for improvement in the current security services of cloud IAM. More secure methods for obtaining cloud credentials and more robust, user-friendly mechanisms for formulating cloud credential policies should be carefully designed and implemented.

References

[1] 2014. Soot. https://github.com/soot-oss/soot.
[2] 2022. Turkish Based Airline's Sensitive EFB Data Leaked. https://www.safetydetectives.com/news/pegasus-leak-report/#review-1.
[3] 2023. Customer information leakage caused by incorrect cloud environment settings. https://global.toyota/jp/newsroom/corporate/39174380.html.
[4] 2023. Wechat historical versions. https://wechat.cn.uptodown.com/android/versions.
[5] 2024. 360 market. https://m.app.so.com/.
[6] 2024. Alibaba Cloud. https://www.alibabacloud.com.
[7] 2024. Alibaba Cloud. Ram policy. https://help.aliyun.com/zh/oss/user-guide/ram-policy-overview.
[8] 2024. Assumerole in AWS. https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_temp_control-access_assumerole.html.
[9] 2024. AWS. https://aws.amazon.com.
[10] 2024. CERT. https://www.kb.cert.org/.
[11] 2024. Google Play. http://googleplay.com/.
[12] 2024. Overview of IAM users. https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html.
[13] 2024. Overview of ram users. https://help.aliyun.com/zh/ram/user-guide/overview-of-ram-users.
[14] 2024. S3 API. https://docs.aws.amazon.com/AmazonS3/latest/API/API_Operations_Amazon_Simple_Storage_Service.html.
[15] 2024. scrapy. https://github.com/scrapy/scrapy.
[16] 2024. Security best practices in CAM. https://cloud.tencent.com/document/product/598/10592.
[17] 2024. Security best practices in IAM. https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html.
[18] 2024. Security best practices in RAM. https://help.aliyun.com/zh/openapi/accesskey-security-solution.
[19] 2024. Tencent Cloud. https://cloud.tencent.com.
[20] Omar Alrawi, Chaoshun Zuo, Ruian Duan, Ranjita Pai Kasturi, Zhiqiang Lin, and Brendan Saltaformaggio. 2019. The betrayal at cloud city: An empirical analysis of Cloud-Based mobile backends. In *28th USENIX Security Symposium (USENIX Security 19)*. 551–566.
[21] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy H Katz, Andrew Konwinski, Gunho Lee, David A Patterson, Ariel Rabkin, Ion Stoica, et al. 2009. Above the Clouds: A Berkeley View of Cloud Computing. *Technical Report UCB/EECS-2009-28, EECS Department, University of California* (2009).
[22] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. 2014. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. *ACM sigplan notices* 49, 6 (2014), 259–269.
[23] Jack Cable, Drew Gregory, Liz Izhikevich, and Zakir Durumeric. 2021. Stratosphere: Finding Vulnerable Cloud Storage Buckets. In *Proceedings of the 24th International Symposium on Research in Attacks, Intrusions and Defenses*. 399–411.
[24] Yinzhi Cao, Yanick Fratantonio, Antonio Bianchi, Manuel Egele, Christopher Kruegel, Giovanni Vigna, and Yan Chen. 2015. EdgeMiner: Automatically Detecting Implicit Control Flow Transitions through the Android Framework.. In *NDSS*.
[25] CAPPVD. 2024. CAPPVD. https://www.cappvd.org.cn/.
[26] Yuanchao Chen, Yuwei Li, Yuliang Lu, Zulie Pan, Yuan Chen Shouling Ji, Yu Chen, Yang Li, and Yi Shen. 2025. Understanding the Security Risks of Websites Using Cloud Storage for Direct User File Uploads. *IEEE Transactions on Information Forensics and Security* (2025).
[27] CNVD. 2024. CNVD. https://www.cnvd.org.cn.
[28] Andrea Continella, Mario Polino, Marcello Pogliani, and Stefano Zanero. 2018. There's a Hole in That Bucket! A Large-Scale Analysis of Misconfigured S3 Buckets. In *Proceedings of the 34th Annual Computer Security Applications Conference*. 702–711.
[29] Murilo Góes de Almeida and Edna Dias Canedo. 2022. Authentication and authorization in microservices architecture: A systematic literature review. *Applied Sciences* 12, 6 (2022), 3023.
[30] Josué Alejandro Díaz-Rojas, Jorge Octavio Ocharán-Hernández, Juan Carlos Pérez-Arriaga, and Xavier Limón. 2021. Web API Security Vulnerabilities and Mitigation Mechanisms: A Systematic Mapping Study. In *2021 9th International Conference in Software Engineering Research and Innovation (CONISOFT)*. IEEE, 207–218.
[31] AWS Documentation. 2024. AWS Management Console. https://docs.aws.amazon.com/awsconsolehelpdocs/latest/gsg/learn-whats-new.html.
[32] Soufian El Yadmani, Olga Gadyatskaya, and Yury Zhauniarovich. 2024. The File That Contained the Keys Has Been Removed: An Empirical Analysis of Secret Leaks in Cloud Buckets and Responsible Disclosure Outcomes. In *2025 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 9–9.
[33] Lori M Kaufman. 2009. Data Security in the World of Cloud Computing. *IEEE Security & Privacy* 7, 4 (2009), 61–64.
[34] A Kanchana Rajaram, B Chitra Babu, et al. 2013. API Based Security Solutions for Communication Among Web Services. In *2013 Fifth International Conference on Advanced Computing (ICoAC)*. IEEE, 571–575.
[35] Shangcheng Shi, Xianbo Wang, and Wing Cheong Lau. 2019. MoSSOT: An automated blackbox tester for single sign-on vulnerabilities in mobile applications. In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*. 269–282.
[36] Shangcheng Shi, Xianbo Wang, Kyle Zeng, Ronghai Yang, and Wing Cheong Lau. 2021. An Empirical Study on Mobile Payment Credential Leaks and Their Exploits. In *Security and Privacy in Communication Networks: 17th EAI International Conference, SecureComm 2021, Virtual Event, September 6–9, 2021, Proceedings, Part II 17*. Springer, 79–98.
[37] Xueqiang Wang, Yuqiong Sun, Susanta Nanda, and XiaoFeng Wang. 2023. Credit Karma: Understanding Security Implications of Exposed Cloud Services Through Automated Capability Inference. In *32nd USENIX Security Symposium (USENIX Security 23)*. 6007–6024.
[38] Haohuang Wen, Juanru Li, Yuanyuan Zhang, and Dawu Gu. 2018. An Empirical Study of SDK Credential Misuse in iOS Apps. In *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 258–267.
[39] Zhifeng Xiao and Yang Xiao. 2012. Security and Privacy in Cloud Computing. *IEEE Communications Surveys & Tutorials* 15, 2 (2012), 843–859.
[40] Yue Zhang, Jian Weng, Jiasi Weng, Lin Hou, Anjia Yang, Ming Li, Yang Xiang, and Robert H Deng. 2019. Looking back! using early versions of android apps as attack vectors. *IEEE Transactions on Dependable and Secure Computing* 18, 2 (2019), 652–666.

[41] Yajin Zhou, Lei Wu, Zhi Wang, and Xuxian Jiang. 2015. Harvesting Developer Credentials in Android Apps. In *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*. 1–12.

[42] Chaoshun Zuo, Zhiqiang Lin, and Yinqian Zhang. 2019. Why Does Your Data Leak? Uncovering the Data Leakage in Cloud from Mobile Apps. In *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1296–1310.

9 Acknowledgments

This work was supported by the National key R&D projects(Project No.E3500921L6).