The Betrayal At Cloud City: An Empirical Analysis Of Cloud-Based Mobile Backends

Omar Alrawi, *Georgia Institute of Technology;* **Chaoshun Zuo,** *Ohio State University;* **Ruian Duan and Ranjita Pai Kasturi,** *Georgia Institute of Technology;* **Zhiqiang Lin,** *Ohio State University;* **Brendan Saltaformaggio,** *Georgia Institute of Technology*

https://www.usenix.org/conference/usenixsecurity19/presentation/alrawi

This paper is included in the Proceedings of the 28th USENIX Security Symposium.

August 14–16, 2019 • Santa Clara, CA, USA

978-1-939133-06-9

Open access to the Proceedings of the 28th USENIX Security Symposium is sponsored by USENIX.

The Betrayal At Cloud City: An Empirical Analysis Of Cloud-Based Mobile Backends

Omar Alrawi*
Georgia Institute of Technology

Chaoshun Zuo*
The Ohio State University

Ruian Duan
Georgia Institute of Technology

Ranjita Pai Kasturi
Georgia Institute of Technology

Zhiqiang Lin
The Ohio State University

Brendan Saltaformaggio
Georgia Institute of Technology

Abstract

Cloud backends provide essential features to the mobile app ecosystem, such as content delivery, ad networks, analytics, and more. Unfortunately, app developers often disregard or have no control over prudent security practices when choosing or managing these services. Our preliminary study of the top 5,000 Google Play Store free apps identified 983 instances of N-day and 655 instances of 0-day vulnerabilities spanning across the software layers (OS, software services, communication, and web apps) of cloud backends. The mobile apps using these cloud backends represent between 1M and 500M installs each and can potentially affect hundreds of thousands of users. Further, due to the widespread use of third-party SDKs, app developers are often unaware of the backends affecting their apps and where to report vulnerabilities. This paper presents SkyWalker, a pipeline to automatically vet the backends that mobile apps contact and provide actionable remediation. For an input APK, SkyWalker extracts an enumeration of backend URLs, uses remote vetting techniques to identify software vulnerabilities and responsible parties, and reports mitigation strategies to the app developer. Our findings suggest that developers and cloud providers do not have a clear understanding of responsibilities and liabilities in regards to mobile app backends that leave many vulnerabilities exposed.

1 Introduction

Cloud-based mobile backends provide a wide array of features, such as ad networks, analytics, content delivery, and much more. These features are supported by multiple layers of software and multiple parties including content delivery networks (CDNs), hosting providers, and cloud providers who offer virtual/physical hardware, provisioned operating systems, and managed platforms. Due to the inherent complexity of cloud-based backends, deploying and maintaining them securely is challenging. Consequently, mobile app developers often disregard prudent security practices when choosing cloud infrastructure, building, or renting these backends.

Recent backend breaches of the British Airways [1] app and Air Canada [2] app demonstrate how wide-spread these incidents are. More recently, the hijacking of the Fortnite mobile game [3] showed how incrementally-downloaded content from mobile backends can allow an attacker to install additional mobile apps without the user's consent. Additional cases [4] involving the exposure of 43TB of enterprise customer names, email addresses, phone numbers, PIN reset tokens, device information, and password lengths was due to *insecure mobile backends* and not the developer's mobile app code.

Even for security-conscious developers, it is not clear what backends their mobile app will interact with because of third-party libraries. Third-party libraries do not expose their backends to developers, instead, they offer an application program interface (API) that developers use. Many of these vulnerabilities can be identified ahead of time if developers have the right tools and resources to evaluate the security of their backends. Further, identifying vulnerable software layers and the responsible party can expedite remediation and therefore lower the risk of exposure.

To deal with the complexities in cloud infrastructure, the research community surveyed [5] and proposed several taxonomies [6], ontologies [7], assessment models [8], and threat classifications [9]. Unfortunately, these approaches provide few practical recommendations for mobile app developers. Recent works on server-side vulnerability discovery of mobile apps [10]–[12] have shown that a lack of security awareness among app developers is a growing problem. Yet, these works only scratch the surface by examining only the software service layer of mobile backends.

A systematic study is needed to identify the most pressing issues facing mobile backends. Moreover, to conduct such a study, the analysis must be reproducible, transparent, and easy to interpret for developers. The study should be done on a representative mobile app ecosystem to provide real in-

*Authors contributed equally.

sight into the backend vulnerability landscape. Finally, the study should offer practical steps to guide and inform app developers on the security of their mobile backends.

To this end, this paper presents the design and implementation of SkyWalker, an analysis pipeline to study mobile backends. Using SkyWalker, we conducted an empirical analysis of the top 5,000 free mobile apps in the Google Play store from August 2018. Based on this study, we uncovered 655 0-day instances and 983 N-day instances affecting thousands of apps. We used Google Play Store metadata to measure the impact of our findings and estimate the number of affected users. We propose mitigation strategies for different types of vulnerabilities and guidelines for developers to follow. Lastly, we offer the SkyWalker analysis pipeline as a free public web-service to help developers identify what backends their mobile apps interact with, the security state of the backends, and recommendations to address any detected issues.

Our empirical study found 983 N-day instances of 52 vulnerabilities affecting hypervisors, operating systems, databases, mail servers, DNS servers, web servers, scripting language interpreters, and others. We found 655 0-day instances of SQL injection (SQLi), cross-site-scripting (XSS), and external XML entity (XXE). These affected thousands of mobile apps, with some apps having over 50M+ installs and more than 332,000 reviews. We present two case studies to demonstrate the vulnerabilities affecting a specific developer and vulnerabilities affecting a platform that is used by many developers.

We found these backends to be geographically distributed across the globe and hosted on 6,869 different networks. We notified all affected parties about the findings, and were careful to follow ethical and legal guidelines when conducting this study, additional details are in Section 8. We propose mitigation strategies for developers to follow based on the issues found and the types of backends. We conclude with recommendations for deploying and maintaining secure backends.

2 A Motivating Example

Mobile apps use cloud-based backend services to support extensive functions like ads, telemetry, content delivery, and analytics. Unfortunately, a mobile app developer who wants to audit the backends their app uses will quickly find that this is harder than it seems. The first thing the developer must do is simply enumerate those mobile backends. Consider the *Crime City Real Police Driver* (*com.vg.crazypoliceduty*) app, a mobile game with over 10M+ installs and 126,257 reviews. The mobile app uses several third-party SDK libraries including Amazon In-App Billing, SupersonicAds, Google AdMob, Unity3D, Nuance Speech Recognition Kit, and Xamarin Mono. The developer may not be aware of many of the backends that are invoked from imported native or Java libraries, i.e., the Unity3D backends. In most cases, the developer will first have to employ static binary analysis tools or dynamically instrument the app to track multiple levels of SDK inclusion. SkyWalker automatically identified 13 unique backends from this app's APK (shown in Table 1) and mapped them to the modules they were found in, i.e., library backends versus developer backends.

Party	Vendor	Backend	Purpose
Hybrid	Vasco Games	androidha.vascogames.com	Game Content
Third	Unity3D	api.uca.cloud.unity3d.com	Telemetry
		cdn-highwinds.unityads.unity3d.com	Ads
		config.uca.cloud.unity3d.com	Telemetry
		impact.applifier.com	Telemetry
	Sizmek	bs.serving-sys.com	Ads
		secure-ds.serving-sys.com	Ads
	Moat	px.moatads.com	Analytics
		z.moatads.com	Ads
	Google	googleads.g.doubleclick.net	Ads
		pagead2.googlesyndication.com	Ads
		tpc.googlesyndication.com	Ads
		www.google-analytics.com	Analytics

Table 1: Backends identified for *Crime City Real Police Drive* and their purpose. The red cells indicate vulnerable backends.

Backends have layers of software (components) that support the web application software (*AS*), including an operating system (*OS*), software services (*SS*), and communication services (*CS*). The developer now has to fingerprint the backends to inventory the software layers and identify the software type, version, and its purpose. Using this information, the developer can then check to see if any of their software is outdated or affected by a known vulnerability [13], a laborious and time-consuming task. SkyWalker identified that the game content backend runs *Debian 6* for the *OS*; *OpenSSH 6.5p1*, *Apache httpd 2.2.22*, *PHP/5.4.4-14*, and *Apache-Coyote/1.1* for the *SS*; and uses the *HTTP* protocol for *CS*. SkyWalker's search of the national vulnerability database (NVD) [13] and correlation with the fingerprint results showed multiple common vulnerability exposure (CVE) entries affecting PHP 5.4.4-14. Further, the Debian version running on the backend is no longer supported and does not receive any updates from the vendor.

In addition to these issues, the developer's *AS* can contain bugs that must be audited. The developer can check the *AS* by auditing the parameters passed to each API and testing for SQLi, XSS, XXE, or any other applicable vulnerabilities from OWASP's top 10 common issues [14]. This task requires secure programming experience and security domain expertise to identify bugs in the source code. SkyWalker found that the game content backend interface is vulnerable to SQLi for some parameters passed by the mobile app, which is due to the *AS* not properly sanitizing the input.

The developer must now remediate or mitigate these risks, but each backend layer may be operated by different entities that provide hardware and software as a service. Therefore,

before fixing any issues, they must figure out what party is responsible for each component. SkyWalker fingerprinted the *Crime City Real Police Driver* game content backend, *androidha.vascogames.com*, as being hosted on a Google Compute Engine Flexible Environment instance (which provides virtual hardware, operating system, and PHP). We refer to this type of backend model as *hybrid* since Google is partially responsible for the virtual environment and the developer is responsible for the *AS* and *CS*.

The developer must come up with a remediation strategy to address these problems. Google advertises that they patch any vulnerable software affecting the *OS* and *SS*, but this is only applicable to non-deprecated versions. In the case of *Crime City Real Police Driver* app, the developer is responsible for all the software layers since the *OS* and *SS* versions are deprecated. The developer must upgrade to a supported *OS*, apply patches to the PHP interpreter (*SS*), patch the *AS* source code against SQLi, and support *HTTPS* for secure *CS*.

The Unity3D, Sizmek, and Moat backends shown in Table 1 are called *third-party*, since the developer has no control over them. This evaluation must also be carried out on third-party backends to identify additional vulnerabilities (potentially affecting all apps which use those shared services). SkyWalker found that the *Crime City Real Police Driver* app uses the *config.uca.cloud.unity3d.com* backend, which contains an XXE vulnerability, and the *bs.servingsys.com* backend that contains an XSS vulnerability. Ideally, the developer could report those vulnerabilities to the platform through a bug bounty program or migrate their app to backends that are not vulnerable.

This manual assessment procedure is very involved and requires extensive security domain knowledge, which many app developers may not have. Instead, SkyWalker gives all mobile app developers the ability to identify the backends invoked by their app, assess their software layers, and suggest remediation strategies to improve the security for their mobile app backends.

3 Background

This section defines an abstraction to model mobile backends for our empirical study. We also define our labeling for backends and create a mapping between responsible stakeholders and resources. We outline how we count vulnerabilities and define them in the context of this work.

3.1 Mobile App Backend Model

We follow the standard definition for mobile backends used by industry leaders [15]–[18], which encompass many cloud features, such as storage, user management, notifications, and APIs for various services, regardless of who maintains/owns them. We breakdown mobile backends into a stack representation that consists of five layers:

- **Hardware** (*HW*) refers to the physical or virtual hardware that hosts the backend.
- **Operating System** (*OS*) refers to the OS running on the hardware, i.e., Linux or Windows.
- **Software Services** (*SS*) refers to software services running in the OS, i.e., database service, web service, etc.
- **Application Software** (*AS*) refers to the custom application interface used by mobile apps to interact with the running services.
- **Communication Services** (*CS*) refers to the communication channel supported between the mobile app and the mobile backend.

Our approach does not consider the hardware layer because 1) we would need root-level access on the backend to evaluate the hardware and 2) mobile app developers have no direct way of addressing hardware vulnerabilities, i.e., manufacturers must issue firmware updates or replace the hardware. It is important to note that this work does not consider the mobile app security, instead we leverage the mobile app to study the backend.

We differentiate between mobile backends by ownership, which provides a granular mapping between stakeholders and resources. We define four labels for the mobile backends with respect to the app developer:

- **First-Party** (B_{1st}) refers to backends that are fully managed by the mobile app developers (i.e., full control over the backend).
- **Third-Party** (B_{3rd}) refers to backends that are fully managed by third-parties (i.e., no control over the backend).
- **Hybrid** (B_{hyb}) refers to backends that are co-managed by third-parties and developers such as cloud infrastructure (i.e., some control over the mobile backend).
- **Unknown** (B_{ukn}) refers to backends that ownership could not be established with high confidence.

In our model, there are two primary stakeholders, the app developers (*D*) and the cloud service providers (*SP*). There are additional stakeholders, like app users and internet service providers (ISP), but they do not have direct remediation oversight. We define a mapping between backends layers, labels, and ownership, shown in Table 2.

The final piece of the model is the mitigation component that maps vulnerable backends to the proper mitigation strategies. There are five mitigation strategies for developers:

- **Upgrade** (*u*) the software to vendor supported versions.
- **Patch** (*p*) vulnerable software with a vendor patch.

Label	*HW*	*OS*	*SS*	*AS*	*CS*
First-Party (B_{1st})	○	○	○	○	○
Third-Party (B_{3rd})	●	●	●	●	●
Hybrid (B_{hyb})	●	◑	◑	○	○

Table 2: Backend labels (first-party - B_{1st}, third-party B_{3rd}, and hybrid - B_{hyb}) and cloud layers (hardware - *HW*, operating system - *OS*, software services - *SS*, application software - *AS*, and communication services - *CS*) mapping to stakeholders (developers - ○, service providers - ●, and shared - ◑)

- **Block** (*b*) incoming internet traffic to exposed services.
- **Report** (*r*) the vulnerability to the responsible party.
- **Migrate** (*m*) the backend to secure infrastructure.

In many cases, the developer may not have control or authority to fix the issues but still has the option to report it (*r*) or change service provider (*m*).

3.2 Counting Vulnerabilities

This work considers vulnerabilities which are software bugs that exist in the backend software stack, including the operating system (*OS*), services (*SS*), application (*AS*), and communication (*CS*). We consider *N-Day* vulnerabilities to be those vulnerabilities which have an associated common vulnerabilities and exposure (CVE) number assigned by the national institute of standards and technology (NIST) and indexed in the national vulnerability database (NVD) [13]. In our findings, we count *N-Day* vulnerabilities by class and instance, where class refers to the CVE number of a particular vulnerability and an instance refers to the vulnerability affecting a specific interface or software component on a mobile backend. For example, Apache Struts vulnerability *CVE-2017-5638* that affects Apache Struts 2.3.*x* before 2.3.32 and 2.5.*x* before 2.5.10.1 is counted as a single vulnerability (class), but it can affect multiple backends that run different versions of Apache Struts (instances).

Some software versions are affected by multiple CVEs, in this case, we **do not** count every CVE as an instance. We generally assume patching the latest CVE should address all previous unpatched CVEs. We only consider the latest CVE affecting the vulnerable software and count it once. Further, a vulnerability instance is a tuple of the backend's domain name, IP address, and the vulnerable software version. As for *0-Day* vulnerabilities, they are associated with the software application (*AS*) running on the backend. This work looks at three classes of *0-Day* vulnerabilities, SQLi, XSS, and XXE and counts each instance per API interface endpoint on the mobile backend. The defined model, labels, mitigations, mappings, and vulnerabilities are the basis for our methodology, which we describe next.

4 Methodology

In this section, we provide an overview of our assessment and details about implementing SkyWalker. Figure 1 is an overview of SkyWalker's internal components. We divide the implementation into four phases, namely binary analysis, labeling, fingerprinting, and vulnerability analysis. Each phase provides input to the next phase, starting from an input app APK to the final vulnerability/mitigation report.

4.1 Binary Analysis

SkyWalker leverages our prior work, Smartgen [19], to perform the binary analysis and extract query messages from an APK binary. SkyWalker dynamically executes the code paths to the network functions and extracts the native usage of the backend APIs. The native usage of an API includes the URI path and their parameter types/values.

4.2 Backend Labels

Backend labeling assigns one of the four labels defined in our model. The labels are used to map the responsible parties and the mitigation strategies needed (excluding unknown), shown in Table 2. Moreover, the labels are used to identify where the most common issues are found. To perform the labeling, we curate three unique lists using the ipcat [20] datacenter dataset. The first list is called *CP* and contains cloud providers, content delivery networks (CDNs), and mobile platform cloud services. The second list, *Colo*, contains a list of collocation centers. The third list is a list of *SDK* libraries that we extracted using LibScout [21] (Table 3), which help SkyWalker identify third-party backends. OSSPolice [22] provides a more comprehensive list, including native libraries used by the mobile app, but our binary analysis technique only instruments Java code, therefore, we limit the third-party SDK identification to LibScout.

To perform the labeling we generate a tuple for each extracted backend B that contains the effective-second level domain d, IP address ip, a boolean flag lib indicating if the backend belongs to an SDK library, and the developer or vendor name v. We define a function $owner()$ that parses WHOIS, MaxMind [23], and ASN records to extract ownership information. The $owner()$ function uses text tokenization, normalization, and aliasing to consolidate varying records.

SkyWalker uses Algorithm 1 to assign labels to each backend. Algorithm 1 takes as input a list of backends, β, containing tuples $B = \{d, ip, lib, v\}$ and returns a list of labeled backends β'. The algorithm uses the *CP* and *Colo* list to check membership for the domains and IPs to determine the appropriate label. The first check is to determine the origin of the backend (was it extracted from an SDK library?) then

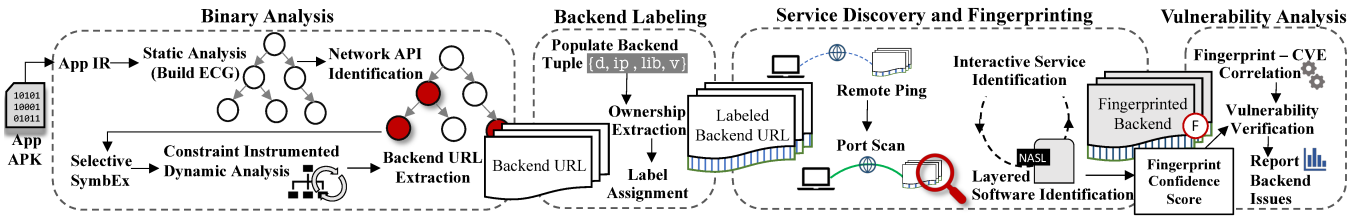

Figure 1: SkyWalker Overview. Phase 1 (Binary Analysis) extracts backend URLs through a dynamic binary instrumentation technique. Phase 2 labels backends into first-party, third-party, and hybrid. Phase 3 discovers and fingerprints the backend services to collect cloud layer information. Phase 4 (vulnerability analysis) uses the fingerprints and correlates them with public vulnerabilities to identify vulnerable backends.

Algorithm 1: Assigning Labels to Backends

Input: β = List of backend tuple $B = \{d, ip, lib, v\}$
Output: β' = Ownership labeled backend list
SDK: List of backend domains found in the SDK libraries;
CP: List of cloud and hosting providers (domains, net prefix, and ASNs);
Colo: List of collocation providers (domains, net prefix, and ASNs);
for $\forall B \in \beta$ **do**
  if $B.lib \vee B.d \in SDK$ **then**
    `// Backend from Java lib`
    $B.label \leftarrow$ "third-party";
    continue
  end
  if $owner(B.d) \neq v \wedge owner(B.d) \notin CP$ **then**
    `// Backend domain not owned by developer or CP`
    $B.label \leftarrow$ "third-party";
    continue
  end
  if $B.ip \in CP$ **then**
    `// Backend IP hosted by cloud provider`
    $B.label \leftarrow$ "hybrid";
    continue
  end
  if $B.ip \in Colo$ **then**
    `// Backend IP hosted by collocation center`
    $B.label \leftarrow$ "first-party";
    continue
  end
  $B.label \leftarrow$ "unknown";
end

assigns "third-party" label if *lib*'s value is true or the backend domain belongs to the list of SDK backends.

If none of the previous statements are true about the domain, then SkyWalker checks the *IP* membership against the *CP* and *Colo* list. If the IP address belongs to a network on the *CP* list SkyWalker assigns "hybrid" label. If the IP address belongs to a network on the *Colo* list SkyWalker assigns "first-party" label. Otherwise, SkyWalker assigns an "unknown" label since it cannot be determined. It is important to note that SkyWalker's labeling approach relies on LibScout [21] to identify third-party backends based on the SDK libraries. SkyWalker performs an additional check before setting the *lib* flag to exclude *SDK* libraries built by the same vendor (Google, Facebook, etc.).

4.3 Service Discovery and Fingerprinting

Service discovery identifies internet-facing services on backends and fingerprinting identifies the software type, version, and configuration of each service. Our approach is a multi-tier approach that starts by remotely pinging the backend, then port scanning it, then interacting with the discovered service, and finally collecting service configurations. For instance, the scan first checks to see if the host is reachable, then it scans for all ports to identify available services, then it tries to connect to the service to collect its banner, and finally, if the services use TLS/SSL, it would collect their configurations and supported ciphers. For each step, our scanner is configured to be non-intrusive, throttled (slow scan speed and a light load on the remote server), and conservative (using techniques that yield low to no false positives).

First, SkyWalker groups all IP addresses into their network prefixes and in a random order picks a prefix and a random IP from the selected prefix to scan. Prefixes are grouped by the autonomous system number (ASN) for each network. If a network spans multiple ASNs, SkyWalker keeps each ASN as a separate prefix to distribute the scanning uniformly across different IP segments. SkyWalker does a TCP ping against common service ports (FTP, SSH, HTTP/S, IMAP, SMTP, RDP, etc.) by sending out a SYN packet followed by

Third-Party SDKs

ACRA	CleverTap	InMobi	Supersonic
AMoAd	Crashlytics	JSch	Syrup
AdColony	Crittercism	Joda-Time	Tapjoy
AdFalcon	Dagger	MdotM	Tremor Video
Adrally	EventBus	Millennial Media	Twitter4J
Amazon	ExoPlayer	Mixpanel	Urban-Airship
Android	Facebook	MoPub	Vungle
Apache	Firebase	New-Relic	WeChat
AppBrain	Flurry	OkHttp	flickrj
AppFlood	Fresco	Parse	heyZap
AppsFlyer	Fyber	Paypal	ironSource
BeaconsInSpace	Google	Picasso	jsoup
Bolts	Gson	Pollfish	roboguice
Brightroll	Guava	Retrofit	scribe
Butter-Knife	Guice	Segment	smaato
Chartboost	HockeyApp	Stetho	vkontakte

Table 3: A list of third-party SDKs extracted by LibScout from the top 5,000 apps, which is used to curate third-party backends.

a RST packet. TCP ping scans are more reliable in detecting the availability of the remote server (backend) because they are not filtered by firewalls like ICMP scans.

Once SkyWalker establishes the host is reachable, SkyWalker conducts a TCP SYN scan (SYN-SYN/ACK-RST) across *all* ports. This process identifies candidate ports on the target backend that will be used for a more thorough scan (TCP connect). To be efficient, SkyWalker uses the list of ports identified in the TCP SYN scan to conduct a TCP connect scan (SYN-SYN/ACK-ACK) i.e., establish a complete connection. Based on the port/service identified, SkyWalker interactively grabs the banner, the header response, and any available configuration. The retrieved information varies per service type, for example, HTTP will have header information unlike SSH, nonetheless, both help fingerprint the host. Moreover, SkyWalker looks for TLS/SSL connections on all candidate ports because many services like HTTP and IMAP can run over TLS/SSL. Finally, to obtain the backend IP address fronted by CDNs, SkyWalker looks up the IP address in a manually curated CDN list and uses passive DNS to find historical records that existed just before the current records. When SkyWalker cannot locate such record, the backend is excluded from fingerprinting.

Once SkyWalker discovers all the services running on a backend, SkyWalker uses the result to fingerprint the backend. The fingerprint identifies the *OS*, *SS*, and *CS* type (Linux, Windows; PHP, .NET, Python, Perl; FTP, SFTP, HTTP, HTTPS, SSH, IMAP, etc.), version, and configuration information if available. The fingerprinting uses open source and commercial Nessus Attack Scripting Language (NASL) scripts to identify the different layers of software on the backend. For example, to identify the *OS*, the NASL script inspects the banner string, analyzes the SSL certificate, checks additional running services (SMB, RDP, SSH), performs structured ICMP pings, inspects HTTP headers, and uses TCP/IP fingerprinting algorithms [24]. Based on these signals a confidence score is provided based on matching a set of pre-profiled *OS*es. For example, if 90% of the signals match a *Windows Server 2008 R2 Service Pack 1* profile, we consider the *OS* layer for that backend in the vulnerability analysis. Any confidence level below 90% or ambiguity between the same *OS* but different versions will not be considered for the vulnerability analysis phase.

Web Applications. Web apps (*AS*) are generally tailored per mobile app, unlike *OS*, *SS*, and *CS* layers. The binary phase performs in-context analysis for each API interface on the backend, which provides API information used for fingerprinting. We reference the OWASP's top 10 vulnerability issues [14] that can be passively tested within the ethical and legal bounds discussed in Section 8. Specifically, SkyWalker uses side-channel SQLi through time delay, reflective XSS, and XXE callback to identify *candidate* issues in web apps. It is important to note that other vulnerabilities such as authentication bypass, broken access control, and sensitive data exposure present a high risk that can violate legal obligations. Adding a module to SkyWalker to support additional vulnerabilities is trivial and can be easily implemented.

For each backend interface, a number of parameters (p) are associated with each request. SkyWalker tests each interface p times to check every parameter for SQLi and XSS. The XXE check is performed on all interfaces because some *AS* can accept JSON or XML requests. As mentioned earlier, the scan is slow and randomly done to avoid congestion and degradation of service on production backends. SkyWalker creates two queues, a job queue and a processing queue. SkyWalker generates p requests for a given backend interface and stores them in the job queue. The job queue contains all backend requests, which are shuffled and loaded into the processing queue in batches (128 requests per batch). Batches that contain requests with the same domain or IP address are removed and replaced by non-overlapping domains and IP address requests. There are 32 workers that ingest from the processing queue and store the results for vulnerability analysis.

4.4 Vulnerability Analysis

The vulnerability analysis is two parts, N-day analysis, and 0-day analysis. For the N-day analysis, SkyWalker correlates CVE entries with results from the fingerprinting to identify possible issues. The confidence level of the fingerprint results is also used to verify each vulnerability. SkyWalker uses NASL scripts that take the output of the service discovery, *OS* identification, *SS* identification, and *CS* identification as input and match them against known vulnerabilities (CVEs). The NASL results are considered if they have 90% confidence level or higher for *OS* detection, which provides high accuracy for vulnerability matching. Note, that the confidence level is calculated based on pre-profiled *OS*es by matching the fingerprint signals (collected from all layers) to the profile signals.

We manually verified all 983 N-days and found them to be all true positives. The zero false positive results are due to the Nessus configuration, which allows us to tune how the scans are done and how they should be reported. For example, we configure Nessus to perform the scan types described above, consider *OS* type and version detection of 90% or higher and consider *SS* that have banner information with version numbers. On the other hand, when we used UDP scanning techniques and consider generic service banner information we find over $6,500$ candidate N-day instances with a large false positive rate. In theory, the backend can be configured to lie about the banner information, which would make it hard for us to verify.

For the 0-day analysis, SkyWalker carefully triggers the candidate vulnerability to verify the findings. For each vulnerable parameter, SkyWalker generates a pair of request messages, the original message and the vulnerable message.

For *SQLi*, SkyWalker baselines the original request message several times throughout the week and at different times of the day. Then SkyWalker performs the same measurement on the vulnerable message in the same week but in non-overlapping time intervals by triggering the vulnerable parameter through an SQLi sleep injection. SkyWalker calculates the response time deviation based on the sleep parameter passed in the SQL statement and the average response time of the message pairs. If the deviation is equal to the time delay parameter in the SQL statement, SkyWalker concludes that the interface and parameter pair is vulnerable.

Similarly for *XSS*, SkyWalker triggers the vulnerable parameter and includes JavaScript code to creates a new *div* element with a unique *name* attribute. SkyWalker checks the returned content by parsing the document object model (DOM) to find the *div* element containing the unique *name* attribute. If the *div* element with the set *name* attribute exists SkyWalker concludes that the interface and parameter are vulnerable. Note that SkyWalker matches the returned content with parameters sent to ensure that the *XSS* candidate vulnerability is of type 2 (reflected). For *XXE*, SkyWalker generates a request message that contains an HTTP callback request to a server we operate. The request message is passed to the backend, which will parse the specially crafted XML document. If the parser is vulnerable to XXE, SkyWalker will log an HTTP request from the backend under analysis, which indicates the interface is vulnerable. In addition, we manually reviewed the request/return pairs for all 655 0-day instances and found no false positives.

4.5 Open Access for Developers

One of our primary goals for this work is to empower app developers with open access to SkyWalker via a free-to-use web-service. The service currently supports Android mobile apps but can be extended to support other mobile platforms, e.g., Apple iOS. The web-interface takes as input a link to an Android app in the Google Play store or a direct APK upload. SkyWalker then performs binary analysis to extract the backends, label them based on our curated dataset, fingerprint them, and identify vulnerabilities that affect them. In addition to the analysis, the output report provides guidelines on how to mitigate the identified issues using the strategies discussed earlier (upgrade, patch, block, report, and migrate).

SkyWalker summarizes vulnerability findings across all observed SDK and Java library backends, which developers can turn to *proactively* to make an informed decision when choosing third-party libraries to include in their future apps. It is important to note that attackers can abuse this system to attack mobile app backends. Therefore we require the developers to disclose their affiliation with the target app before the analysis results are provided. Once a user is manually vetted, they can only submit apps that they develop. We do not consider third-party *SDK*s in this process. The SkyWalker service can be found at: https://MobileBackend.vet.

5 Assessment Findings

5.1 Experiment Setup

Environmnet. We use a local workstation running Ubuntu 14.04 with 24GB memory and 16 x 2.393GHz Intel Xeon CPUs and four Nexus phones to run and instrument the mobile apps. We use an Amazon Web Service (AWS) Elastic Compute (EC2) instance with a reserved IP address to conduct the fingerprinting and run a web server with information about our study along with an email address for backend hosts to contact us if they want to opt-out.

Tools and Data Sets. For the binary analysis tool implementation, we relied on Soot [25], FlowDroid [26], Z3-str [27], and Xposed [28] with custom code written in Java (7,000 lines of code) and Python (900 lines of code). For our backend labeling implementation, we relied on Team Cymru IP-to-ASN [29], MaxMind Geolocation [23], Alexa ranking [30], ipcat list [20], and Domaintools WHOIS [31] with custom code written in Python (480 lines of code). For fingerprinting, we relied on the Nessus scanner and commercial plugins [32], sqlmap [33], and Acunetix [34]. We used Nessus plugins and custom Python code (1010 lines of code) to perform the vulnerability analysis. For internet measurements, we utilized honeypot scanning activity from Greynoise [35].

5.2 Software Vulnerability Details

Table 4 shows the distribution of 0-day and N-day instances across the software layers. We categorize the apps using the Google Play store groups and present the number of vulnerabilities and backend labels. Overall, we analyzed 4,980 apps with cloud-based backends and successfully extracted backends for 4,740 mobile apps. The remaining 240 mobile apps crashed and did not complete the full binary analysis.

Interestingly, the *OS* component reports the least vulnerabilities, while the *AS* component reports the most vulnerabilities, across all mobile app categories. Recall from Section 3.2, vulnerabilities affecting *AS* components are all considered *0-day*. The *OS*, *SS*, and *SC* components account for *N-day* vulnerabilities. Although the number of apps is not uniform across the categories, we use the *raw* vulnerability count for ranking. For *0-day* vulnerabilities, the top three mobile app categories are tools, entertainment, and games. For *N-day* vulnerabilities, the top three mobile app categories are entertainment, tools, and games.

Ownership. Table 4 presents the labels for the backends used by mobile apps. The most common label is *hybrid*, where 3,336 backends use hybrid infrastructure. The second

Category	# Mob. Apps	Vulnerabilties					Labels				
		# *OS*	# *SS*	# *AS*	# *CS*	**Total**	# B_{1st}	# B_{3rd}	# B_{hyb}	# B_{ukn}	**Total**
Books & Reference	332	15	49	55	71	190	365	653	501	354	1,873
Business	145	5	22	10	37	74	93	258	150	113	614
Entertainment	1,177	36	108	158	170	472	746	913	942	783	3,384
Games	1,283	34	81	147	106	368	290	804	651	444	2,189
Lifestyle	363	20	50	79	72	221	262	665	311	237	1,475
Misc	199	6	21	45	46	118	76	422	163	105	766
Tools	792	19	84	184	115	402	729	796	812	464	2,801
Video & Audio	689	24	46	89	98	257	267	648	434	357	1,706
Total	4,980	121	356	655	506	1,638	2,492	1,089	3,336	2,506	9,423

Table 4: An overview of the vulnerable mobile apps per genre along with the *raw* counts of vulnerabilities and labels.

Party	Vulnerable Component				Total
	OS	*SS*	*AS*	*CS*	
B_{1st}	37	87	155	211	490
B_{3rd}	6	21	200	42	269
B_{hyb}	47	150	154	184	535
B_{ukn}	55	135	146	173	509

Table 5: Count of *apps* affected by vulnerabilities per cloud layer and their corresponding labels.

Comp.	Vulnerability (Top 3)	#Apps
OS	Expired Lifecycle for Linux OS (various)	124
	Windows Server RCE (MS15-034)	64
	Expired Lifecycle for Windows Server	9
SS	Vulnerable PHP Version	357
	Expired Lifecycle for Web Server (various)	181
	Vulnerable Apache Version	76
AS	XSS (various)	262
	SQLi (various)	160
	XXE (various)	86
CS	Support for Vulnerable SSL Version 2 and 3	997
	OpenSSH Bypass (CVE-2015-5600)	16
	Vulnerable OpenSSL (various)	15

Table 6: The top three vulnerabilities found per cloud layer along with the number of affected mobile apps.

largest is *first-party* with 2,492 backends followed by *third-party* with 1,089 backends. There are 2,506 backends that we were not able to label due to ambiguities, but we labeled approximately 73% of all backends we encountered.

More important is providing remediation guidance to the responsible party. Table 5 shows the mapping between the backend labels and the vulnerable apps. We cannot say much about the *Unknown* category since the vulnerabilities may belong to either *first-party* or *hybrid* categories. We observe that for first-party backends, the highest number of vulnerable apps are found in *AS* and *CS* components with 155 and 211 instances, respectively. Similarly, the hybrid backends have 154 0-day vulnerability instances and 184 N-day instances. In general, we observe that the components that app developers are responsible for (*AS* and *CS* in the B_{1st} and B_{hyb}) have more vulnerabilities.

Operating System (*OS*). The *OS* component issues can be summarized into two categories: legacy unsupported *OS* or unpatched *OS*. The difference is that the legacy *OS* are no longer supported by the vendor, hence vulnerabilities will not be addressed. We see from Table 6 that both *Linux* (various flavors) and *Windows* backends use expired lifecycle versions and 133 apps use these backends. The second most common issue is the *Windows Server* vulnerability *MS15-034* affecting 64 apps, which have patches by the vendor. Overall, the top three *OS* vulnerabilities listed in Table 6 affect 197 mobile apps.

We found the MS15-034 vulnerability affecting hybrid backends (B_{hyb}) that run on *Amazon AWS*, *Akamai*, *OVH*, *Go Daddy*, *Digital Ocean*, and other smaller hosting providers. Further, some of the backends appear on CDN networks, like *Akamai*, *Fastly*, and *CloudFlare*, that offer "EdgeComputing" services [36] which provide web app accelerator services. This insight shows that some developers who deploy vanilla versions of Windows Server *OS* are not maintaining them. In Table 5 the first-party *OS* component has 37 vulnerable backends, which is much higher than third-party backends (6). App developers who run and maintain their own backends (B_{1st}) have to be mindful of these bugs, which in some cases require provisioning new backends with newer OSes causing incompatibilities with existing services (*SS*) and applications (*AS*). SkyWalker can inform the developer of these issues and report mitigation strategies.

Software Services (*SS*). SkyWalker identified multiple vulnerabilities affecting a range of PHP versions, which can be used to cause denial of service (*CVE-2017-6004*), disclose memory content (*CVE-2017-7890*), disclose sensitive information (*CVE-2016-1903*), and execute arbitrary code (*CVE-2017-11145*). Backends of 357 mobile apps affected by PHP vulnerabilities, significantly higher than the other two vulnerabilities. Further, even though some mobile app backends had no *0-day* vulnerabilities, an attacker can still craft special requests to trigger deep bugs within the interpreter to compromise the backend. Although this might be a difficult task, recent advancement in vulnerability fuzzing [37] can uncover these deep bugs.

The second most common *SS* vulnerability was unsupported versions of Apache web server (1.3.x and 2.0.x), Tomcat server (8.0.x), and Microsoft IIS web server (5.0). Similar to unsupported *OS*, web server vendors will not issue security patches for unsupported software, which affects backends of 181 mobile apps. For Apache web server versions less than 2.2.15, they are affected by several denial of service bugs (*CVE-2010-0408, CVE-2010-0434*) and TLS injection bug (*CVE-2009-3555*) affecting 76 mobile apps. Additionally, Apache servers that use Apache Struts versions 2.3.5 - 2.3.31 or 2.5.10.1 and lower are vulnerable to *CVE-2017-5638*, which allows remote code execution. The same Apache Struts vulnerability was reportedly used against Equifax's hack [38]. In total, the top three *SS* vulnerabilities affect 614 mobile apps.

Applications (*AS*). Table 7 has a breakdown of the number of mobile apps, their number of install categories, and the *instances* of *0-day* bugs affecting them. Although *XSS* is the largest category with 503 instances followed by *SQLi* (215) and *XXE* (46), we note that not all of the bugs have the same impact and some affect the same backend. For instance, an *SQLi* can be limited to an isolated instance of the app (e.g., a container), which would limit the attack to disclosing information from the application database or modifying preexisting records. Moreover, *XSS* vulnerabilities often have less impact than *SQLi* and *XXE*.

XXE vulnerabilities affect web apps that use XML for their API communication. The fundamental flaw that enables XXE vulnerabilities to exist is a faulty implementation of the XML parser. Based on our measurement, we found 1 *XXE* instance in the top 100M, 5 in the top 50M, 15 in the top 10M, 9 in the top 5M, and 17 in the top 1M. Table 7 shows the concentration of vulnerabilities found in lower ranking apps. For example: 1 *XXE* and 3 *XSS* vulnerabilities in the top 132 mobile apps; 4 *SQLi*, 10 *XSS*, and 5 *XXE* vulnerabilities in the next 131 mobile apps (though still representing over 50M+ installs each). However, *AS* vulnerabilities are not confined to lower ranking apps but do affect higher ranking apps.

# Installs	# Apps	# SQLi	# XSS	# XXE
1B	5	0	0	0
500M	11	0	0	0
100M	116	0	3	1
50M	131	4	10	5
10M	1,049	25	85	15
5M	1,047	54	89	9
1M	2,621	132	316	17

Table 7: The number of *0-day* vulnerabilities found per install category.

Table 8 shows the *AS* layer implementation language and associated vulnerabilities. *AS* implemented in *PHP* have the most *0-days* instances (284) affecting 108 different backends, followed by *ASP.NET* with 33 *0-day* instances affecting 13 different backends. We note that this trend does not mean causation. *PHP* is the most popular language used for web application development [39], hence it is expected to represent more vulnerabilities by being more popular. Furthermore, we found 9 *0-day* instances in *PERL*, 8 in *JavaScript (NodeJS)*, 5 in *JSP*, and the rest of the 316 could not be determined.

Language	# Backends	# 0-Days
PHP	108	284
ASP.NET	13	33
PERL	4	9
JS	4	8
JSP	2	5
Unknown	72	316

Table 8: The number of identified languages associated with *0-day* vulnerable backends.

Communication (*CS*). All mobile apps rely on the HTTP/HTTPS protocol for communication with their backends. The binary analysis phase extracted a total of 17,725 request messages from the 4,740 mobile apps. The request messages are split into HTTP (8,118) request messages and HTTPS (9,607) request messages. There are 446 mobile apps that only use HTTP communication and another set of 147 mobile apps that only use HTTPS communication. The remaining set of 4,147 apps mix between HTTP and HTTPS communication.

Despite using HTTPS, over 20% of the backends (1,012) have issues with TLS/SSL configuration (e.g., insecure session renegotiation and resumption) or unpatched software versions (e.g., SSL version 2 and 3). These flaws can be exploited by an attacker to carry out a MITM attack by downgrading the protocol negotiation using the POODLE [40] attack. Additionally, the OpenSSH Bypass vulnerability exposes the backend to compromise via SSH credential guessing or secret key leak. The mobile apps using these vulnerable backends do not use the *SSH* service and to remediate one can turn off, patch, or block the incoming internet traffic to it.

Those backends which only use HTTP expose users to eavesdropping and MITM attacks because it does not offer integrity or confidentiality. We manually inspected the request messages sent from 3,253 apps that use HTTP and found personally identifiable information (PII) such as name, gender, birth year, user ID, password, username, and country. Additionally, we found device information like MAC, IMEI, SDK version, make/model, SSID, Wifi signal, cell signal, screen resolution, carrier, root access, IP Address, and coordinate location. Combining this information, a network attacker can identify individuals and attribute behavior profiles to them. Furthermore, 6 apps we investigated perform a password reset over HTTP. Interestingly, the Apple iOS App Store enforces strict use of HTTPS through their *App Trans-*

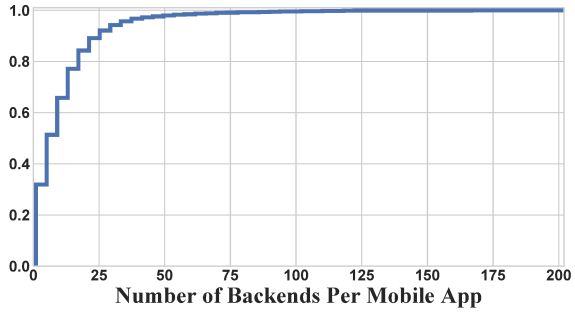

Figure 2: The figure shows the CDF of the number of backends per mobile app.

port Security [41] model. We recommend that the Android platform adopt the same restriction.

5.3 Impact on Mobile Application Users

The overall impact for each vulnerability varies based on the severity, the mobile app to backend usage, and the adversary capability/visibility. Although it is important to understand the impact of each vulnerability, it is not trivial to quantify the impact of each vulnerable backend on mobile apps. For N-day vulnerabilities, an attacker can perform an internet-wide scan to identify and attempt to compromise these backends. Even once identified, these N-days span many different components (*OS*, *SS*, and *CS*) that have varying impacts on the backend from basic information disclosure to a full system compromise. For 0-day vulnerabilities the attack impact varies based on the exploit type (*SQLi*, *XSS*, or *XXE*) and how the backend infrastructure is set up. Moreover, how the mobile app uses the backend directly impacts the severity of the vulnerability. For example, if a mobile app uses app slicing [3] or downloads additional libraries from the mobile backend, an attacker who compromises the backend can modify the content and attain code execution on the mobile device.

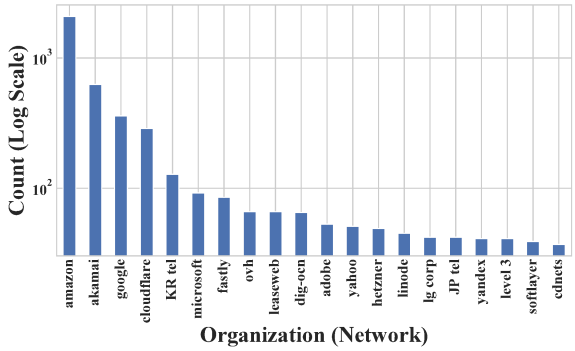

Figure 3: The figure shows the distribution of backends across internet networks.

In general, an attacker has a larger attack surface for apps that have many backends. Figure 2 shows a CDF of backends per mobile app. We can see that the majority of the 5,000 apps studied have between one and 25 different backends and in the worst case they have up to 203 different backends. We also observe that these backends reside in diverse networks as shown in Figure 3, which means the infrastructure set up for the backends will be different affecting the impact of the vulnerability.

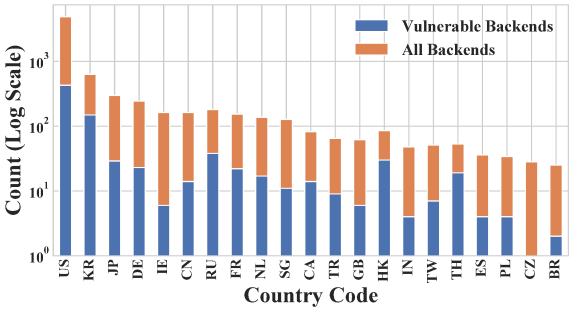

Figure 4: The figure shows the distribution of all mobile backends and vulnerable backends across the world.

Finally, the geographical distribution of the backends, shown in Figure 4, affect the impact on mobile apps. Many mobile apps deploy multiple backends that are geographically distributed to provide faster content for different user segments. In some cases, the different backends may not be fully synchronized in terms of the latest software patches for *OS*, *SS*, *AS*, and *CS* layers, which results in a vulnerable backend affecting only a segment of users for a particular mobile app. Directly quantifying the impact of each vulnerability is an involved task and depends on many variables such as the severity of the vulnerability, the mobile app to backend usage, the adversary capability, and other nuance factors (number of backends per app, network distribution, and geographical distribution). We plan to perform a comprehensive analysis to understand this impact as future work.

5.4 Vulnerability Disclosure, Bug Bounties, And In The Wild Threats

During our disclosure process, we identified two mobile platform vendors that have a bounty program, namely *Unity3D* [42] and *Simpli.fi* [43]. In addition, the top third-party platform providers, Google, Facebook, Crashlytics, and Flurry, all participate in or run their own bug bounty programs. Similarly, the cloud providers either run their own program or use a third-party bug bounty program like Bugcrowd [44] or Bounty Factory [45]. We submitted our vulnerability disclosures through their bounty management program (e.g., *HackerOne* [46]) and received confirmation of the bugs.

For smaller third-party and first-party developers, they did not have a formal way to contact them to report vulnerabilities. We followed a tiered approach in our notification by first notifying the app developer directly using the contact information in the Play Store. Our second attempt to report the vulnerability is by contacting the domain owner using the WHOIS information and following the mitigation strategy. Our third attempt to report the vulnerability is by contacting Google directly through their issue tracker portal. For parties that did not confirm or respond to our multiple attempts, we reported the vulnerabilities to US-CERT [47].

Component	# IP Scanners
Operating System (*OS*)	341,521
Services (*SS*)	445,908
Application (*AS*)	206,533

Table 9: Number of IPs observed scanning the internet for vulnerabilities reported by Greynoise.io [35] over a period of a year (Sept 2017 to Sept 2018).

The N-day vulnerabilities we found are discoverable and easy to exploit due to the availability of fast internet scanners like ZMap [48] and MASSCAN [49]. We argue that it is a matter of time until these vulnerabilities are found and exploited. Table 9 shows the number of active scans detected on the internet through Greynoise [35] honeypots over a period of one year (Sept 2017 to Sept 2018). There are 341,521 unique IPs scanning for *OS* related vulnerabilities, 445,908 unique IPs scanning for *SS* vulnerabilities, and 206,533 unique IPs scanning for *AS* vulnerabilities. Many of these scans target *N-day* vulnerabilities, while scans for *0-day* vulnerabilities cannot be accounted for. Nonetheless, past events demonstrate that attackers are prone to scan for and exploit 0-day vulnerable web apps like Wordpress [50], Drupel [51], and PHPMyAdmin [52] when publicly disclosed. Furthermore, a recent report [53] also pointed out that the number of vulnerabilities in web apps increased in 2018 and that support for PHP version 5.*x* and 7.*x* will end in 2019, which means we can anticipate more unpatched and exposed backends in the future.

6 Case Studies

6.1 Case Study 1: Vulnerable Web App

The mobile app "Dailyhunt" has more than 50M+ installs and is part of the "Books & Reference" category. The mobile app interacts with nine different backends as shown in Table 10. The backends are split into two labels, hybrid and third-party. The hybrid backends are hosted on Akamai's EdgeComputing [36] and run a custom web app to serve the mobile app. The hybrid backends are used for CDN, telemetry, and requesting app-specific data. Specifically, the

Label	Backend	Use	Vulns. *AS*	Vulns. *CS*
B_{hyb}	api-news.dailyhunt.in	App Data	0	1
	acq-news.dailyhunt.in	App Data & Telemetry	2	1
	bcdn.newshunt.com	CDN	0	1
	acdn.newshunt.com	CDN	0	1
B_{3rd}	fonts.gstatic.com	CDN	0	0
	e.crashlytics.com	Telemetry	0	0
	settings.crashlytics.com	Telemetry	0	0
	t.appsflyer.com	Ads	0	0
	api.appsflyer.com	Ads	0	0

Table 10: A list of backends and issues found for the mobile app Dailyhunt.

*api-news.** domain registers the device and requests content, where the *acq-news.** backend captures user behavior and offers promotion and the actual content is delivered by the two CDN domains *acdn.** and *bcdn.**.

We were not able to fingerprint the *OS* and the *SS* because the Akamai servers respond only to web app specific responses, i.e., minimal header and banner information. Nonetheless, we found two 0-day vulnerabilities in the *acq-news.** backend on the same API interface. Since this web application is specific to this mobile app, we looked for other apps published by the same developer. We found that the *eBooks by Dailyhunt* app (which has over 500K installs but does not rank in the top 5,000 apps) also uses the same vulnerable API interface. Additionally, the mobile apps use HTTP to communicate with the hybrid backends and HTTPS to communicate with third-party services.

As for the third-party services, we did not find any vulnerabilities. The third-party backends serve requests on port 443 (HTTPS). The *appsflyer.com* backend is a service for ad analytics that provides different functions using the same interface. The *t.appsflyer.com* backend is a telemetry endpoint for the ad network and the *api.appsflyer.com* backend authenticates and associates the app with its profile.

Takeaway. This case highlights several challenges to securing mobile app backends. First, backends are heterogeneous and differ across their software stack, topology setup, configuration, and custom application. Second, outsourcing cloud management and provisioning (e.g., to cloud providers and CDNs) benefits security but comes with a lack of visibility, limited per-app customization, and unclear incident liability. Third, vulnerabilities can exist (and be scanned for) in any software layer of the cloud and API interface on the web app, which makes them challenging to identify and fix. Unfortunately, app developers do not have the resources, time, or personnel to fulfill this task. Using SkyWalker, we aim to provide guidance to where the most pressing issues exist and map them to responsible parties as shown in Table 2.

App Name	# Reviews	# Installs
com.icegame.fruitlink	332,907	50M+
com.unbrained.wifipasswordgenerator	151,518	10M+
com.magdalm.wifimasterpassword	148,355	10M+
com.unbrained.wifipassgen.app	43,824	1M+
com.magdalm.freewifipassword	35,552	1M+
apps.ignisamerica.gamebooster	23,725	500K+
com.icegame.crazyfruit	23,631	1M+
com.magdalm.wifipasswordpro	22,113	1M+
apps.ignisamerica.bluelight	16,659	500K+
com.icegame.fruitsplash2	15,193	1M+

Table 11: A list of the top 10 mobile apps using the *appnext* platform.

Label	Backend	Usage	Vulns.		
			OS	*AS*	*CS*
B_{3rd}	admin.appnext.com	App Data	0	1	0
	global.appnext.com	App Data	0	0	0
	cdn.appnext.com	CDN	1	0	1
	cdn3.appnext.com	CDN	1	0	1

Table 12: List of backends and vulnerable layers found in the *appnext* platform.

6.2 Case Study 2: Vulnerable Platform

The *appnext* [54] platform integrates with mobile apps to ingest user behavior telemetry and provide predictive actions that users might perform. Developers use this to upsell subscription, ads, or recommend actions to app users. The *appnext* platform is used by 6 mobile apps from the top 5,000 free apps. We analyzed all apps by the same developers that are not in the top 5,000 and found 140 additional apps using the *appnext* platform. The top 10 most reviewed apps using the *appnext* platform can be found in Table 11. The top app has 332,907 reviews and over 50M+ installs. These numbers give us an indication of the the platform's significant popularity and daily use.

The *appnext* platform backends (shown in Table 12) are labelled as third-party, because the backends are found in an SDK library. We found two CDN domains that point to the same server IP, which are hosted on *Limelight Networks*, a CDN provider. This CDN backend is vulnerable to an *OS* integer overflow in the HTTP protocol stack (MS15-034) that can be remotely exploited. Further, the *CS* still offers SSLv2 and SSLv3, which are vulnerable to insecure padding scheme for *CBC* cipher. *appnext*'s *admin.** and *global.** domains run on Amazon AWS and provide app-specific data, like authentication, telemetry ingestion, predictive actions, and configuration. The infrastructures run Microsoft Windows Server 2008 R2 for the *OS*, Microsoft-IIS/7.5 for its web server (*SS*), and the *CS* uses *HTTPS*. The application (*AS*) backend is a custom web application that is written in *ASP* and uses the *ASP.NET* framework. The *AS* has a vulnerability that allows an attacker to run arbitrary SQL queries. We have notified the developers about these findings and awaiting remediation.

Takeaway. This case highlights multiple vulnerabilities, *0-day* and *N-day*, that affect three of the four software layers. This mobile platform collects sensitive information about user behavior, including PII and device information. Unfortunately, these backend vulnerabilities are inherited by multiple apps and developers, and the app developers cannot immediately remediate the vulnerabilities in third-party services. The mitigation strategy for the app developer is to report (*r*) these findings to *appnext* or migrate (*m*) their app to a different service. SkyWalker helps us label the backends, identify the vulnerability, and guide the developer to a clear action (report or migrate).

7 Mitigation

The goal of our empirical analysis was to bring attention to this overlooked problem in mobile backends, but also to provide guidance to app developers for building or choosing secure backends. In this section, we discuss the general mitigation strategies which SkyWalker recommends for app developers and to help improve the security posture of their app backends.

7.1 Remediation Strategies

App developers who rely on first-party backends have to **upgrade**, **patch**, and **block** as needed for each software layer on their backend. If they rely on third-party backends they can **report** the issue or **migrate** their backend to a more secure provider. Ambiguity arises when the backend is hosted by a cloud provider, a hybrid type backend. To resolve these issues we further generalize the hybrid backends into IaaS (cloud provider manages the virtual *HW*) and PaaS (cloud provider manages *HW*, *OS*, and *SS*).

	Hybrid				
Strategies	**HW**	**OS**	**SS**	**AS**	**CS**
Upgrade		✔	✔		✔✔
Patch		✔	✔	✔✔	✔✔
Block			✔		✔
Report	✔✔	✔	✔		
Migrate	✔✔	✔	✔		

Table 13: A mapping of mitigation strategies for developers hosting their hybrid backend on infrastructure (**IaaS**) or a platform (**PaaS**).

Table 13 provides developers with a guideline on how to mitigate vulnerable hybrid backends. For example: if the hybrid backend is using a cloud provider's platform offering, developers should report and/or migrate their backend if the vulnerabilities are found in *HW*, *OS*, *SS* and upgrade or patch if the vulnerabilities *CS* or *AS* related, respectively.

This matrix provides a starting point for app developers to explore their options, i.e., migrate or wait for a fix. In some cases, the offering from cloud providers includes *HW* and *OS* (as in the motivating example which uses Google Compute Engine Flexible Environment). In this case, developers have to make sure they use the latest *OS* images supported by their cloud provider.

7.2 Recommendations

The empirical analysis provides insight not only about insecure mobile backends, but also secure practices that developers can learn from. For developers who decide to build their own first-party backends, we recommend the following: First, developers should **delegate** as much of the backend functionality to reputable third-party backends and minimize the number of features and functions their backend needs to support. Second, developers should **dedicate** personnel to manage and maintain their backends including the routine maintenance of *OS*, *SS* and *CS*, and timely fixes of known vulnerabilities affecting their cloud backends and mobile apps using patching tools [55]–[57]. Third, developers should **develop** an audit plan and a mitigation plan and be familiar with it to execute during an incident or vulnerability disclosure. Finally, developers should utilize **defense** tools like web app firewalls (WAF), DDoS mitigation, and crawler/scanner blockers to protect from internet scanners, DDoS threats, and web app attacks (SQLi, authentication bypass, etc.). We identified over 730 backends using defense services, all of which had smaller footprints when fingerprinted and no vulnerabilities were detected.

8 Measurement Considerations

Ethical. Because our work does not require or implicate human subjects, no IRB approval was required by our institute. Our study identified a large number of *0-day* and *N-day* vulnerabilities in active mobile app backends through scanning and probing. Our techniques include service scans, banner grabs, and side-channel probes. We emphasize that no active exploitation, disruption, or sensitive data access was attempted against the mobile backends. Although there are no set guidelines for vulnerability measurements in the community, several previous works (e.g., [48], [58]–[60]) have set some precedent. Our measurements followed the best practices used in previous work using the following approach:

- **Good Internet Citizenship:** Similar to the work of Li et al. [58], we provided an opt-out page for our scanner IP that gives targets an option to be removed from the study. Further, we signal our benign intention by setting the user-agent string in the scans and provide a reverse DNS record for our IP to give targets additional information about our study. We were contacted by one app developer and requested that we remove their backends and related infrastructure from our study.
- **Non-Exploit Payloads:** Similar to the work of Durumeric et al. [59], our scanning and measurement techniques did not include any active exploits against the mobile app backends. We used side-channel measurements with time delay probes to infer vulnerabilities. The requests were carefully crafted to ensure that vulnerabilities are triggered for verification and not persistent or full system exploitation. Further, our scanning approach was throttled to ensure the availability of the backend is not affected by the additional load.
- **Responsible Disclosure:** Lastly, we notified affected mobile app developers and third-party mobile service providers through the appropriate channels. For developers and third-party service providers who did not respond to our communication, we reported the vulnerabilities through the US-CERT [47].

Legal. Similar to Ristenpart et al. [60], we operate within the legal bounds in conducting this study. In the US, the Computer Fraud and Abuse Act (CFAA) is the governing law that pertains to use and access of computer systems. The law states, in brief, that access to any computer system must be authorized, but does so in broad terms. The decision from the case of Moulton v VC3 (2000) sets a precedent that service discovery scanning does not cause damages or direct harm to target systems. Additionally, we assume any internet-facing service gives implicit permission to access the target computer system, in particular, we refer to how web crawlers and internet indexing services operate. As we outlined in our ethical section earlier, we provide subjects the option to opt-out, perform non-malicious measurement probes, and use responsible disclosure to notify affected parties.

9 Related Work

Cloud Security. Cloud security has been surveyed extensively [61]–[65]. Xiao et al. [9] performed a comprehensive analysis of the security issues in cloud services by surveying high-level *provider* and *tenant* issues for the cloud-based services in general. Singh et al. [66] presented a survey to identify common issues reported in third-party cloud services and summarize the work from the architecture framework, service and deployment, and cloud technologies perspective. Our work looks at "in-the-wild" deployment of cloud services from the *OS*, *SS*, *AS*, and *CS* perspectives to empirically study and uncover common issues in mobile backends.

Measurement Studies. Durumeric et al. [67] conducted a comprehensive internet-wide study of the HTTPS certificate ecosystem. Later, Durumeric et al. [59] carried out a similar internet-wide study for the Heartbleed vulnerability [68].

Perez-Botero et al. [69] presented an in-depth study characterizing hypervisor vulnerabilities in cloud services. Zuo et al. [19] proposed a system to identify mobile app URLs and examine their reputation with public blacklists to detect malicious apps. Our work differs from prior work by studying a range of vulnerabilities which may affect mobile app backends on the internet.

Empirical Backend Analysis. Zuo et al. [12] performed an assessment of mobile app backend services by investigating the cloud offerings of Google, Amazon, and Microsoft. Our work provides a wider analysis by going beyond just the third-party service backends and by examining a diverse set of cloud-based backends. Fernandes et al. [70] analyzed the top apps found in the Samsung SmartThings platform to identify permission issues. We follow a similar approach but focus on the mobile app integration with cloud services instead of IoT apps and cloud services. Alrawi et al. [71] presented a systematization security assessment of home-based IoT devices and their companion cloud and mobile apps. Our work encompasses a wider application, beyond only IoT mobile apps, and a more focused assessment by looking at the supporting backends provided by cloud services.

10 Conclusion

This paper presented SkyWalker, an analysis pipeline to study mobile app backends. We used SkyWalker to empirically analyze the top 5,000 mobile apps in the Google Play store and uncovered 655 0-days and 983 N-days instances affecting thousands of apps. Lastly, we offer SkyWalker as a public service to help app developers improve the security of their backends, give insight on what platforms are vulnerable, and guide developers to fix issues found in their backends: https://MobileBackend.vet.

Acknowledgement

We thank Manos Antonakakis, Yizheng Chen, Angelos Keromytis, Panagiotis Kintis, Chaz Lever, Frank Li, Xiaojing Liao, Yinqian Zhang, and the anonymous reviewers for their insightful comments. This work was partially supported by AFOSR under grant FA9550-14-1-0119, NSF awards 1834215, and 1834216.

References

[1] S. Ghosh, *British Airways customer data stolen from its website*, https://www.theguardian.com/business/2018/sep/06/british-airways-customer-data-stolen-from-its-website, 2018.

[2] Z. Whittaker, *Air Canada confirms mobile app data breach*, https://techcrunch.com/2018/08/29/air-canada-confirms-mobile-app-data-breach/, 2018.

[3] A. Martonik, *Epic's first Fortnite Installer allowed hackers to download and install anything on your Android phone silently*, https://www.androidcentral.com/epic-games-first-fortnite-installer-allowed-hackers-download-install-silently, 2018.

[4] K. Watkins, "HospitalGown: The Backend Exposure Putting Enterprise Data at Risk," Appthority, Tech. Rep., 2017.

[5] S. Subashini and v. Kavitha, "A survey on security issues in service delivery models of cloud computing," *Journal of Network and Computer Applications*, 2011.

[6] C. Höfer and G. Karagiannis, "Cloud computing services: Taxonomy and comparison," *Journal of Internet Services and Applications*, 2011.

[7] L. Youseff, M. Butrico, and D. Da Silva, "Toward a unified ontology of cloud computing," in *In Proc. IEEE Grid Computing Environments Workshop (GCE)*, 2008.

[8] D. Gonzales, J. M. Kaplan, E. Saltzman, Z. Winkelman, and D. Woods, "Cloud-trust-A security assessment model for infrastructure as a service (IaaS) clouds," *IEEE Transactions on Cloud Computing*, 2017.

[9] Z. Xiao and Y. Xiao, "Security and privacy in cloud computing," *IEEE Communications Surveys & Tutorials*, 2013.

[10] K. Watkins and S. M. Kywe, "Unsecured Firebase Databases: Exposing Sensitive Data via Thousands of Mobile Apps," Appthority, Tech. Rep., 2018.

[11] C. Zuo, Q. Zhao, and Z. Lin, "Authscope: Towards automatic discovery of vulnerable authorizations in online services," in *Proceedings of the 24th ACM Conference on Computer and Communications Security (CCS)*, Dallas, TX, Oct. 2017.

[12] C. Zuo, Z. Lin, and Y. Zhang, "Why does your data leak? uncovering the data leakage in cloud from mobile apps," in *Proceedings of the 40th Symposium on Security and Privacy (Oakland)*, San Francisco, CA, May 2019.

[13] National Institute of Standards and Technology, *NATIONAL VULNERABILITY DATABASE*, https://nvd.nist.gov, 2019.

[14] OWASP, *OWASP Top 10 - 2017: The Ten Most Critical Web Application Security Risks*, https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf, 2018.

[15] Kony, *Kony Fabric*, https://www.kony.com/products/fabric/, 2018.

[16] OutSystems, *Build Mobile Apps*, https://www.outsystems.com/platform/build-mobile-apps/, 2018.

[17] Apache, *Architectural overview of Apache Cordova platform*, https://cordova.apache.org, 2018.

[18] Backbase, *Backbase Enterprise Integration Framework*, https://backbase.com/platform/integration/, 2018.

[19] C. Zuo and Z. Lin, "Smartgen: Exposing server urls of mobile apps with selective symbolic execution," in *Proceedings of the 26th International World Wide Web Conference (WWW)*, 2017.

[20] N. Galbreath, *Categorization of IP Addresses*, https://github.com/client9/ipcat, 2019.

[21] M. Backes, S. Bugiel, and E. Derr, "Reliable third-party library detection in android and its security applications," in *Proceedings of the 23rd ACM Conference on Computer and Communications Security (CCS)*, Vienna, Austria, Oct. 2016.

[22] R. Duan, A. Bijlani, M. Xu, T. Kim, and W. Lee, "Identifying open-source license violation and 1-day security risk at large scale," in *Proceedings of the 24th ACM Conference on Computer and Communications Security (CCS)*, Dallas, TX, Oct. 2017.

[23] MaxMind, *About MaxMind*, https://www.maxmind.com/en/company, 2018.

[24] R. Beverly, "A robust classifier for passive TCP/IP fingerprinting," in *Workshop on Passive and Active Network Measurement*, Springer, 2004.

[25] E. Bodden, *A framework for analyzing and transforming java and an- droid apps*, https://sable.github.io/soot/, 2018.

[26] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel, "Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps," in *Proceedings of the 2014 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, Edinburgh, UK, Jun. 2014.

[27] Y. Zheng, X. Zhang, and V. Ganesh, "Z3-str: A z3-based string solver for web application analysis," in *Proceedings of the 18th European Software Engineering Conference (ESEC) / 21st ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE)*, Saint Petersburg, Russia, Aug. 2013.

[28] X. Framework, *Xposed Module Repository*, https://repo.xposed.info/module/de.robv.android.xposed.installer, 2018.

[29] T. Cymru, *IP TO ASN MAPPING*, http://www.team-cymru.com/IP-ASN-mapping.html, 2018.

[30] Alexa, *Find Website Traffic, Statistics, and Analytics*, https://www.alexa.com/siteinfo, 2018.

[31] DomainTools, *About Us*, https://www.domaintools.com/company/, 2018.

[32] T. Security, *Nessus Professional*, https://www.tenable.com/products/nessus/nessus-professional, 2018.

[33] S. Project, *sqlmap: automatic SQL injection and database takeover tool*, http://sqlmap.org, 2018.

[34] Acunetix, *Audit Your Web Security with Acunetix Vulnerability Scanner*, https://www.acunetix.com/vulnerability-scanner/, 2018.

[35] Geynoise, *About*, https://greynoise.io/about/, 2018.

[36] E. Nygren, R. K. Sitaraman, and J. Sun, "The akamai network: A platform for high-performance internet applications," in *Proceedings of the ACM SIGOPS Operating System Review*, vol. 44, Jul. 2010.

[37] W. You, P. Zong, K. Chen, X. Wang, X. Liao, P. Bian, and B. Liang, "Semfuzz: Semantics-based automatic generation of proof-of-concept exploits," in *Proceedings of the 24th ACM Conference on Computer and Communications Security (CCS)*, Dallas, TX, Oct. 2017.

[38] D. Goodin, *Failure to patch two-month-old bug led to massive Equifax breach*, https://arstechnica.com/information-technology/2017/09/massive-equifax-breach-caused-by-failure-to-patch-two-month-old-bug/, 2018.

[39] X. Li and Y. Xue, "A survey on server-side approaches to securing web applications," *ACM Computing Surveys (CSUR)*, 2014.

[40] B. Möller, T. Duong, and K. Kotowicz, "This poodle bites: Exploiting the ssl 3.0 fallback," *Security Advisory*, 2014.

[41] *App Transport Security*, https://forums.developer.apple.com/thread/6767, 2015.

[42] Unity3D, *Imagine, build and succeed with Unity*, https://unity3d.com, 2018.

[43] Simpli.fi, *About Us*, https://www.simpli.fi/about-us/, 2018.

[44] bugcrowd, *THE BUGCROWD DIFFERENCE*, https://www.bugcrowd.com/who-we-are/the-bugcrowd-difference/, 2018.

[45] B. Factory, *CREATE MY BUG BOUNTY PROGRAM*, https://bountyfactory.io/en/mybugbounty.html, 2018.

[46] HackerOne, *About HackerOne*, https://www.hackerone.com/about, 2018.

[47] US-CERT, *About Us*, https://www.us-cert.gov/about-us, 2018.

[48] Z. Durumeric, E. Wustrow, and J. A. Halderman, "Zmap: Fast internet-wide scanning and its security applications.," in *Proceedings of the 22th USENIX Security Symposium (Security)*, Washington, DC, Aug. 2013.

[49] R. D. Graham, *MASSCAN*, https://github.com/robertdavidgraham/masscan, 2018.

[50] M. Veenstra, *Privilege Escalation Flaw In WP GDPR Compliance Plugin Exploited In The Wild*, https://www.wordfence.com/blog/2018/11/privilege-escalation-flaw-in-wp-gdpr-compliance-plugin-exploited-in-the-wild/, 2018.

[51] J. Mattsson, *Drupal core - Highly critical - Remote Code Execution - SA-CORE-2018-002*, https://www.drupal.org/sa-core-2018-002, 2018.

[52] C. Point, *Web servers PHPMyAdmin Misconfiguration Code Injection*, https://www.checkpoint.com/defense/advisories/public/2014/cpai-17-mar1.html, 2018.

[53] N. Avital, *The State of Web Application Vulnerabilities in 2018*, https://www.imperva.com/blog/the-state-of-web-application-vulnerabilities-in-2018/, 2019.

[54] Appnext, *The Appnext Discovery Platform*, https://www.appnext.com/platform/, 2018.

[55] SecurityFTW - cs-suite, *Cloud Security Suite - One stop tool for auditing the security posture of AWS/GCP/Azure infrastructure.* https://github.com/SecurityFTW/cs-suite, 2018.

[56] J. Arnold and M. F. Kaashoek, "Ksplice: Automatic rebootless kernel updates," in *Proceedings of the 4th European Conference on Computer Systems (EuroSys)*, Nuremberg, Germany, Mar. 2009.

[57] R. Duan, A. Bijlani, Y. Ji, O. Alrawi, Y. Xiong, M. Ike, B. Saltaformaggio, and W. Lee, "Automating patching of vulnerable open-source software versions in application binaries," in *Proceedings of the 2019 Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, Feb. 2019.

[58] F. Li, Z. Durumeric, J. Czyz, M. Karami, M. Bailey, D. McCoy, S. Savage, and V. Paxson, "You've got vulnerability: Exploring effective vulnerability notifications," in *Proceedings of the 25th USENIX Security Symposium (Security)*, Austin, TX, Aug. 2016.

[59] Z. Durumeric, F. Li, J. Kasten, J. Amann, J. Beekman, M. Payer, N. Weaver, D. Adrian, V. Paxson, M. Bailey, *et al.*, "The matter of heartbleed," in *Proceedings of the 14th Internet Measurement Conference (IMC)*, 2014.

[60] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds," in *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS)*, Chicago, Illinois, Nov. 2009.

[61] S. Subashini and V. Kavitha, "A survey on security issues in service delivery models of cloud computing," *Journal of network and computer applications*, vol. 34, no. 1, pp. 1–11, 2011.

[62] M. Almorsy, J. Grundy, and I. Müller, "An analysis of the cloud computing security problem," *arXiv preprint arXiv:1609.01107*, 2016.

[63] Y. Sun, G. Petracca, X. Ge, and T. Jaeger, "Pileus: Protecting user resources from vulnerable cloud services," in *Proceedings of the 32th Annual Computer Security Applications Conference (ACSAC)*, 2016.

[64] S. Iqbal, M. L. M. Kiah, B. Dhaghighi, M. Hussain, S. Khan, M. K. Khan, and K.-K. R. Choo, "On cloud security attacks: A taxonomy and intrusion detection and prevention as a service," *Journal of Network and Computer Applications*, vol. 74, pp. 98–120, 2016.

[65] N. V. Juliadotter and K.-K. R. Choo, "Cloud attack and risk assessment taxonomy," *IEEE Cloud Computing*, vol. 2, no. 1, pp. 14–20, 2015.

[66] A. Singh and K. Chatterjee, "Cloud security issues and challenges: A survey," *Journal of Network and Computer Applications*, 2017.

[67] Z. Durumeric, J. Kasten, M. Bailey, and J. A. Halderman, "Analysis of the https certificate ecosystem," in *Proceedings of the 13th Internet Measurement Conference (IMC)*, 2013.

[68] Codenomicon and Google, *The Heartbleed Bug*, https://heartbleed.com/, 2017.

[69] D. Perez-Botero, J. Szefer, and R. B. Lee, "Characterizing hypervisor vulnerabilities in cloud computing servers," in *Proceedings of the 20th ACM Conference on Computer and Communications Security (CCS)*, Berlin, Germany, Oct. 2013.

[70] E. Fernandes, J. Jung, and A. Prakash, "Security analysis of emerging smart home applications," in *Proceedings of the 37th Symposium on Security and Privacy (Oakland)*, San Jose, CA, May 2016.

[71] O. Alrawi, C. Lever, M. Antonakakis, and F. Monrose, "Sok: Security evaluation of home-based iot deployments," in *Proceedings of the 40th Symposium on Security and Privacy (Oakland)*, San Francisco, CA, May 2019.