Ostorlab

MOBILE BANKING APP SECURITY RESEARCH 2025

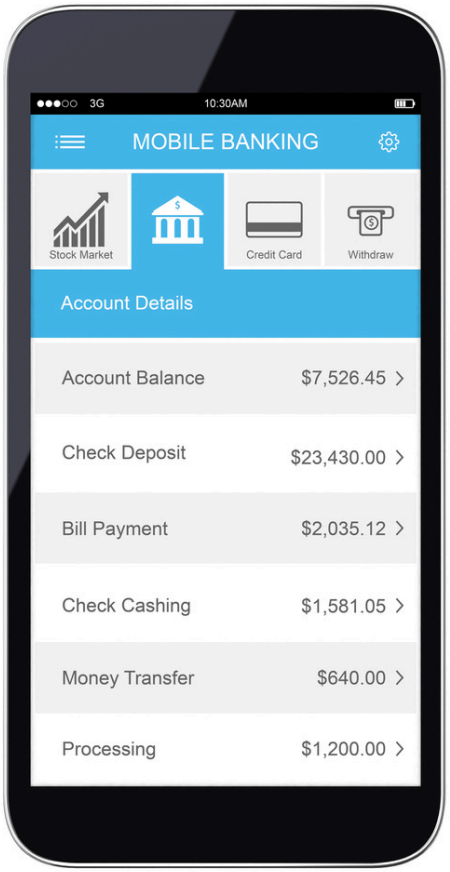
2025

TABLE OF CONTENTS

Executive Summary

Mobile banking is integral to modern finance but remains a significant cyber risk vector. Despite industry improvements like biometric adoption and frequent updates, most apps display real-world, exploitable vulnerabilities: hardcoded secrets, outdated dependencies, insecure configurations, and excessive permissions. Accelerating threats, notably the AI-driven ToxicPanda Trojan, demonstrate that traditional defenses are no longer sufficient. A holistic, proactive security strategy is now urgent to protect user trust and institutional integrity.

Introduction



Facts:

- Nearly 1 in 10 apps hadn't been updated in two years.
- Over 50% of apps contain hardcoded cloud credentials, exposing backend systems and personal data.
- Most traditional banking apps still fail core security standards (e.g., OWASP MASVS).

Mobile banking has become the beating heart of everyday finance, offering convenience, speed, and reach to millions worldwide. Yet, beneath this seamless experience runs a persistent undercurrent of security challenges.
Research in 2024 revealed a troubling reality: a significant portion of banking apps in North America had not seen updates in two years or more, exposing users and financial institutions to known vulnerabilities. The assumption that established banks ensure digital safety does not always hold true.

Surveys spanning hundreds of mobile banking apps found that nearly all failed to meet key industry standards for security, often harboring hardcoded credentials and weak backend protections. These oversights are not merely technical—they carry real danger. Mobile banking apps guard more than just funds; they handle identities, personal histories, and sensitive transactions. A single flaw can open the door to data breaches, financial theft, and a loss of trust that is difficult to restore. In this landscape, the commitment to rigorous app security, from foundational coding to ongoing oversight, has never been more critical.

Applications Overview

Most apps are still actively maintained, with 94.7% of iOS apps and 88.8% of Android apps having had a version released in the last six months, while only 1% of the applications having their latest version over a year old. However, frequent updates do not necessarily reflect security awareness. Many may focus on adding features or UX changes rather than addressing security flaws. The real concern lies in whether these updates actively improve the app's resilience to evolving cyber threats.

Many banking apps have been around for over 10 years. On iOS, about 25% were released between 2008 and 2011, and another 22% between 2011 and 2014. On Android, the numbers are similar — around 27% came out between 2010 and 2013, and 25% between 2013 and 2016. This shows that many apps have been in the market for a long time, which is good for stability. But it also creates a big security challenge: **maintaining old applications. Old codebases that weren't built with today's security in mind need regular updates to stay safe from modern cyber threats.**

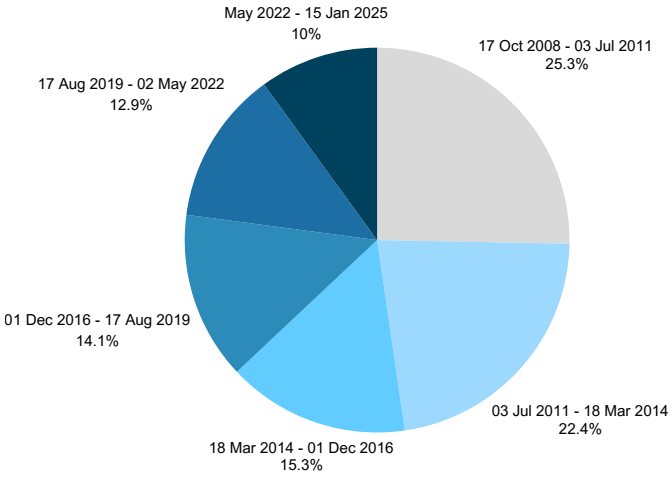


***Figure 1**: iOS application release dates*

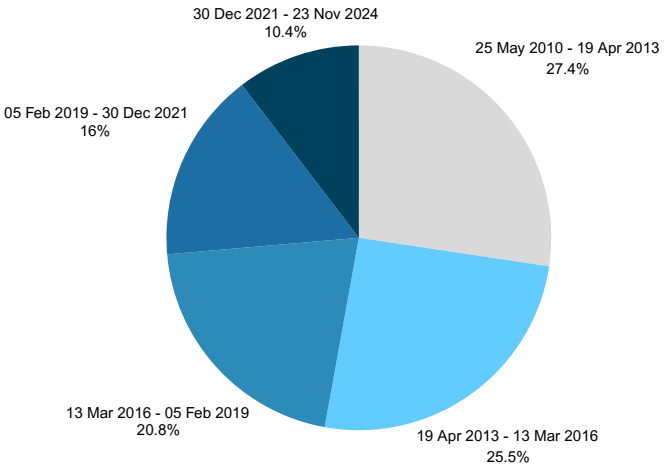


***Figure 2:** Android application release dates*

On iOS, the largest share of apps (43.3%) falls into the 5-164 MB range, with a significant proportion (33.3%) in the 164-336 MB category. Android apps show a tendency towards smaller sizes,

These numbers show that banking apps are stable and still getting updates. But keeping old apps safe is hard, especially if they weren't built with today's security in mind.

with the most common ranges being 74-111 MB (30.6%) and 13-74 MB (29.6%). The size distribution shows that iOS apps tend to be larger, while Android apps are generally smaller.

As apps get bigger and more complex, making sure they stay secure is more important than ever.

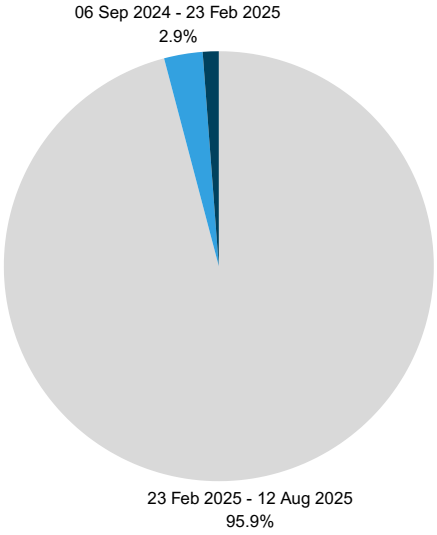


Figure 3: *iOS application current release date*

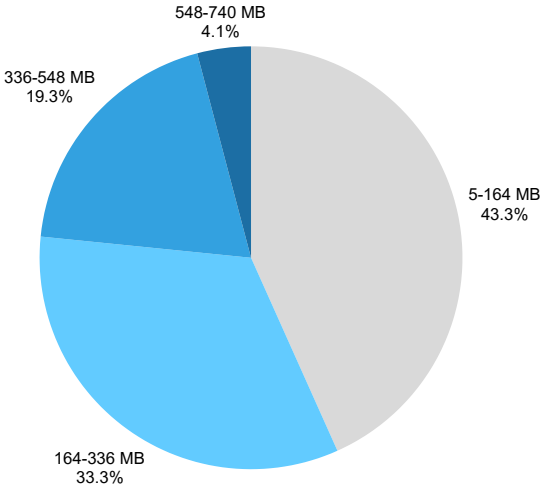


Figure 5: *iOS application size distribution*

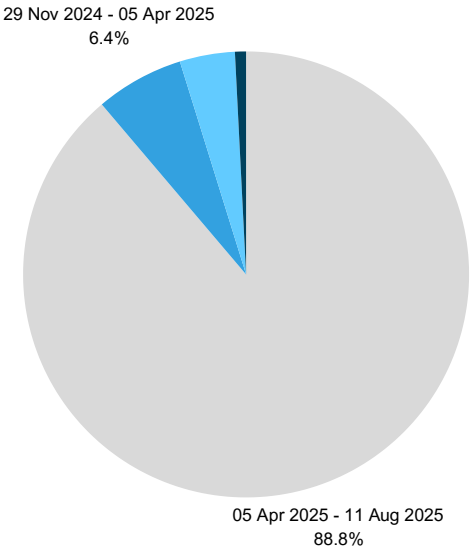


Figure 4: *Android current release date*

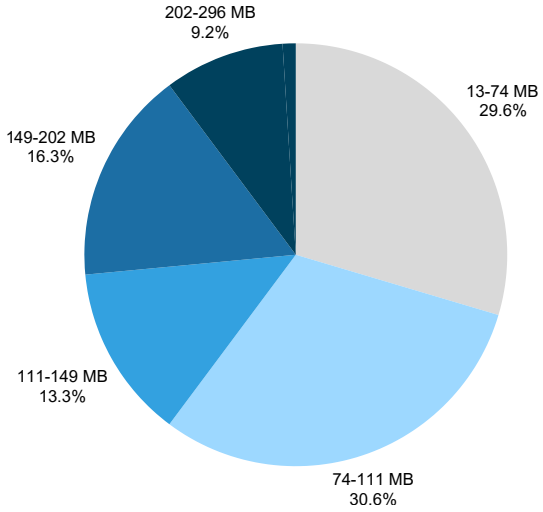


Figure 6: *Android application size distribution*

Technical Architecture

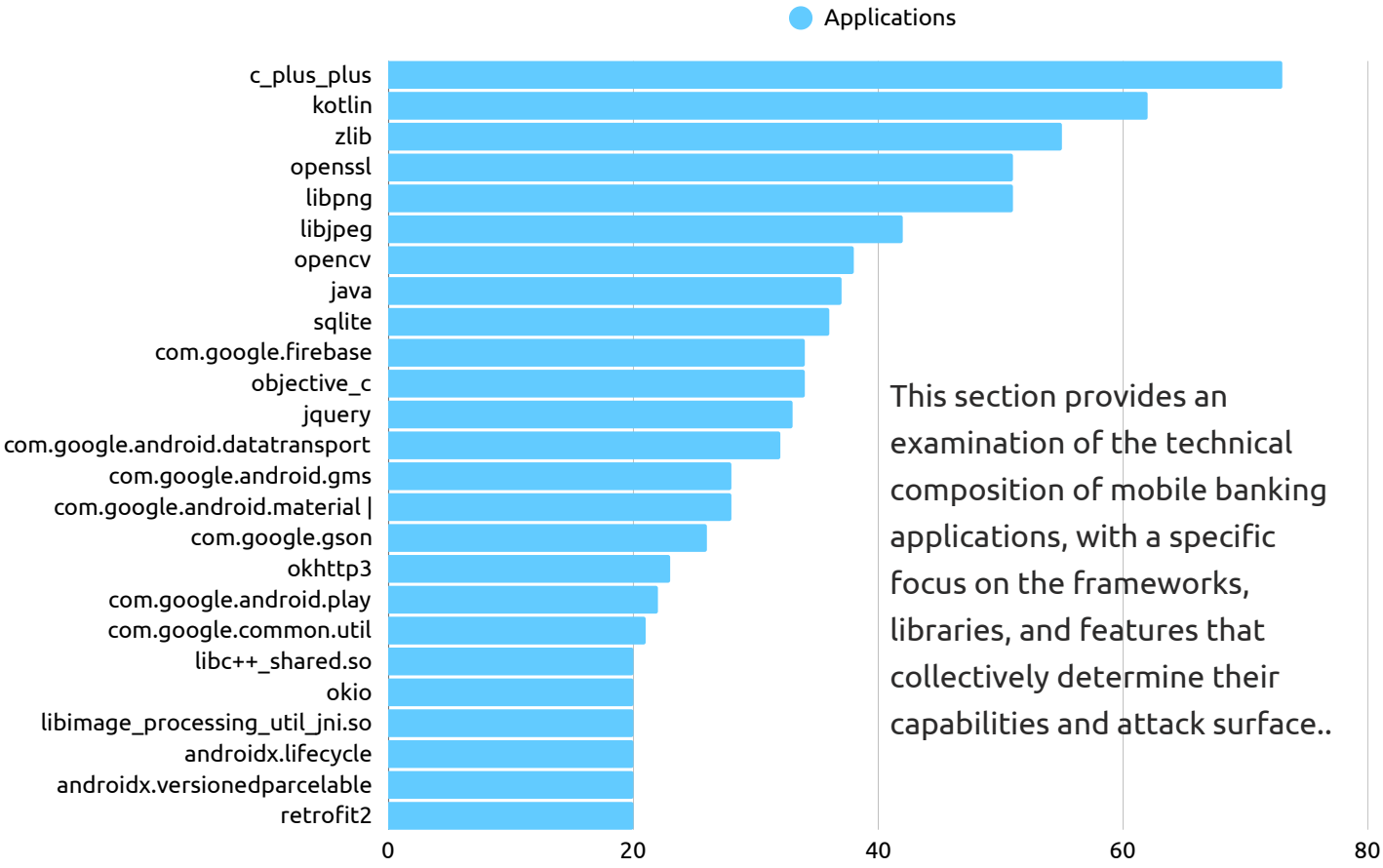


Figure 7: *Number of applications by fingerprint*

Tech Stacks & Libraries

The scans show a clear use of common programming languages, native code, and well-known libraries across mobile apps.
In terms of languages, Kotlin was used in over 61% of Android apps, while Objective-C appeared in over 65% of iOS apps. Native C/C++ code was found in more than 72% of apps.

For libraries, zlib was seen in almost 55% of apps for data compression. OpenSSL was found in over 51% of apps, helping secure network communication. SQLite, used for local data storage, was present in over 35% of apps. As for backend services, Firebase (com.google.firebase) appeared in 34% of apps.

Features and Capabilities

Banking apps rely heavily on native mobile features to provide a secure and smooth user experience. Biometric authentication, used in over 65% of apps, is a key feature for secure login and identity checks. Analytics libraries appear in over 45% of apps, showing a clear focus on tracking user behavior. Few apps include Advertising libraries, and a mere 1% possess user-tracking capabilities. This suggests that most applications prioritize user privacy, avoiding tracking practices. Interestingly, the rare presence of Advertising libraries raises questions, as monetization through ads is generally irrelevant in the financial sector, while the appearance of Health-related features is also unusual within the context of banking applications.

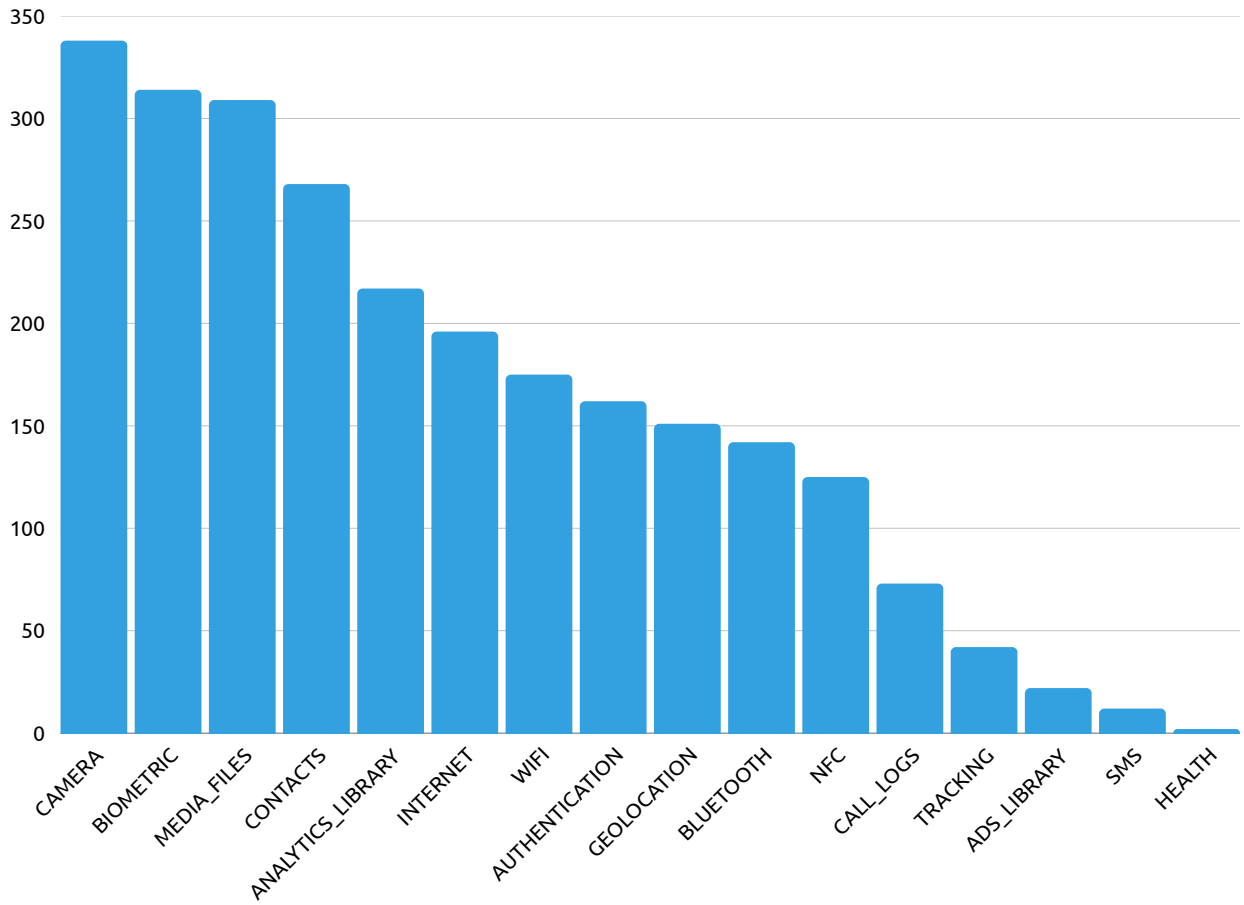


Figure 8: *Distribution applications by feature*

Backend Infrastructure

The scans also showed some interesting differences in how Android and iOS apps use backends. On Android, about 62% of apps connect to 19 or fewer backends, and around 23% use between 19 and 38. On the other hand, most iOS apps—about 78%—connect to just 2 or fewer backends, and another 16% use between 2 and 5.

This suggests that iOS apps tend to rely on fewer backend services, which may reduce complexity. In contrast, Android apps often use more backends, which may require more careful security management.

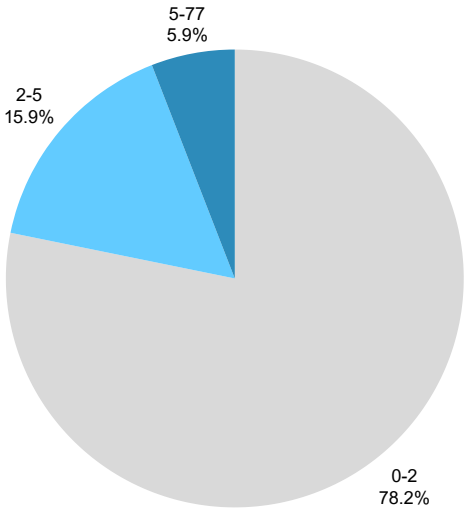


Figure 9: *iOS Number of backends by application*

A look at the distribution of backends by location shows that, although most banking apps are not from the US, they almost all communicate with servers based there. This heavy reliance on US-based infrastructure creates a single point of failure, any breach or outage affecting major US providers could impact multiple banks. It also means that customer data falls under US regulations.

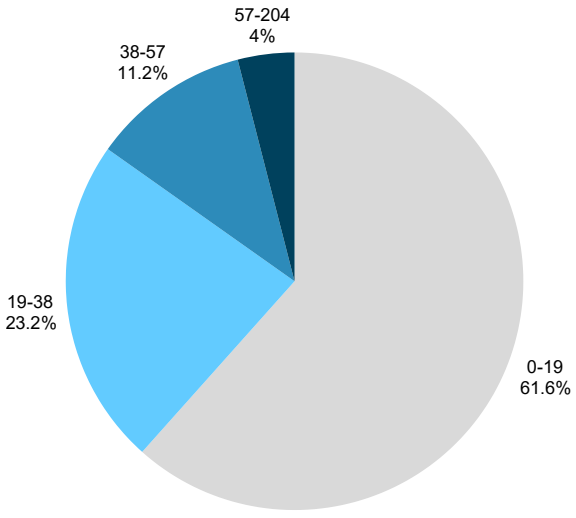


Figure 10: *Android Number of backends by application*

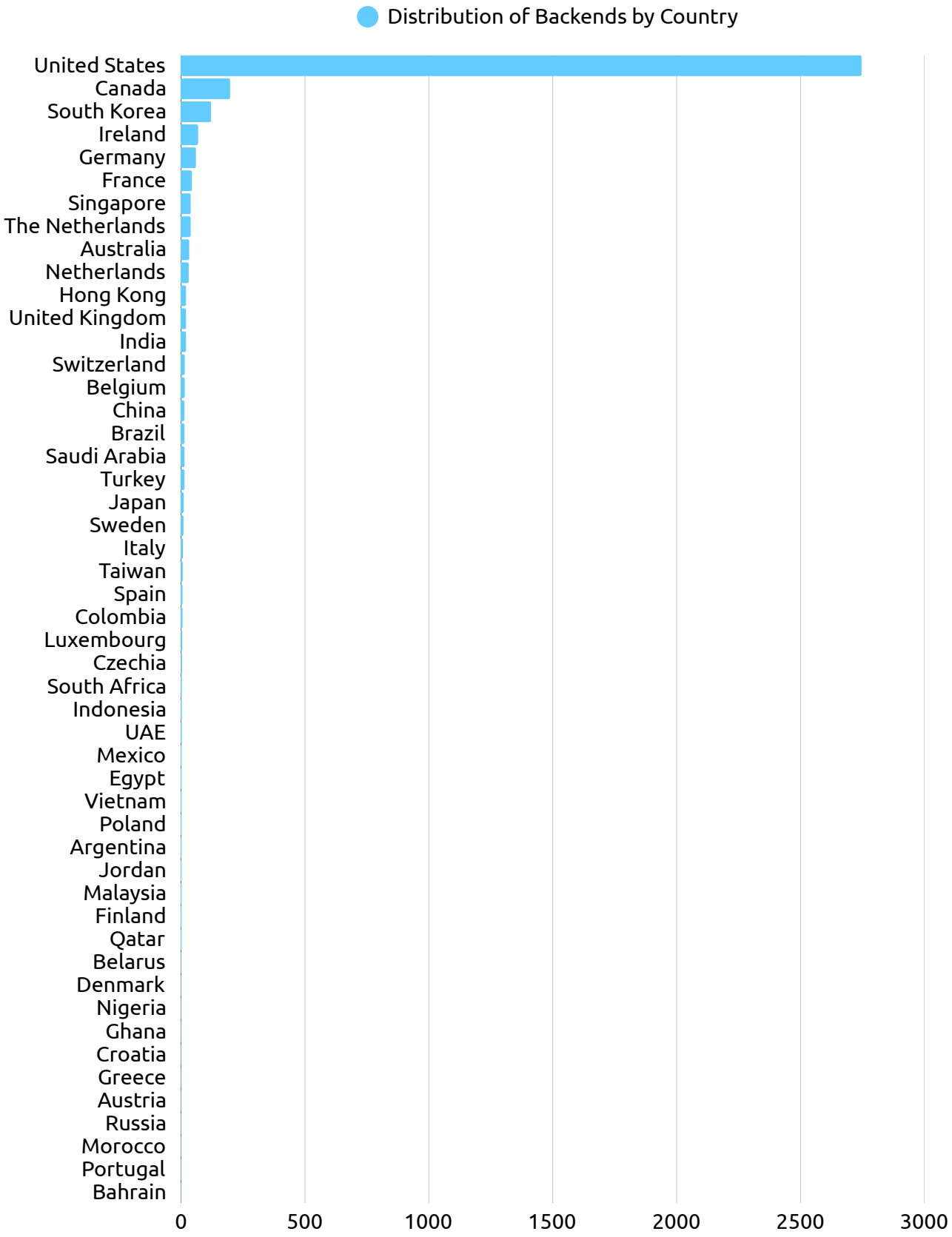


Figure 11: *Distribution of backends by country*

Banking App Attack Surface

Biometric-related permissions such as **USE_FINGERPRINT** and **USE_BIOMETRIC** are prominently used by nearly 80% of applications, reflecting the industry's shift toward stronger, user-friendly authentication methods. Similarly, **CAMERA** and **ACCESS_FINE_LOCATION** are widely used, supporting features like mobile check deposits, QR code scanning, and fraud prevention based on location patterns. Other permissions requested by some Android apps are difficult to justify, such as **FOREGROUND_SERVICE_CAMERA, FOREGROUND_SERVICE_MICROPHONE,** and **DISABLE_KEYGUARD,** which could pose potential risks to user safety and confidentiality.

The security of a mobile application is directly related to its attack surface, which is defined by the number and nature of its entry points and permissions. The analysis of the banking app ecosystem reveals the following key findings regarding potential vulnerabilities. The most widely requested permissions among Android banking apps are consistent with core app functionality. **ACCESS_NETWORK_STATE** and **INTERNET** are nearly universal, as they are essential for enabling network connectivity and online banking operations. **POST_NOTIFICATIONS** ranks high, likely due to its role in real-time alerts for transactions, login attempts, and account changes. **WAKE_LOCK** is also common.

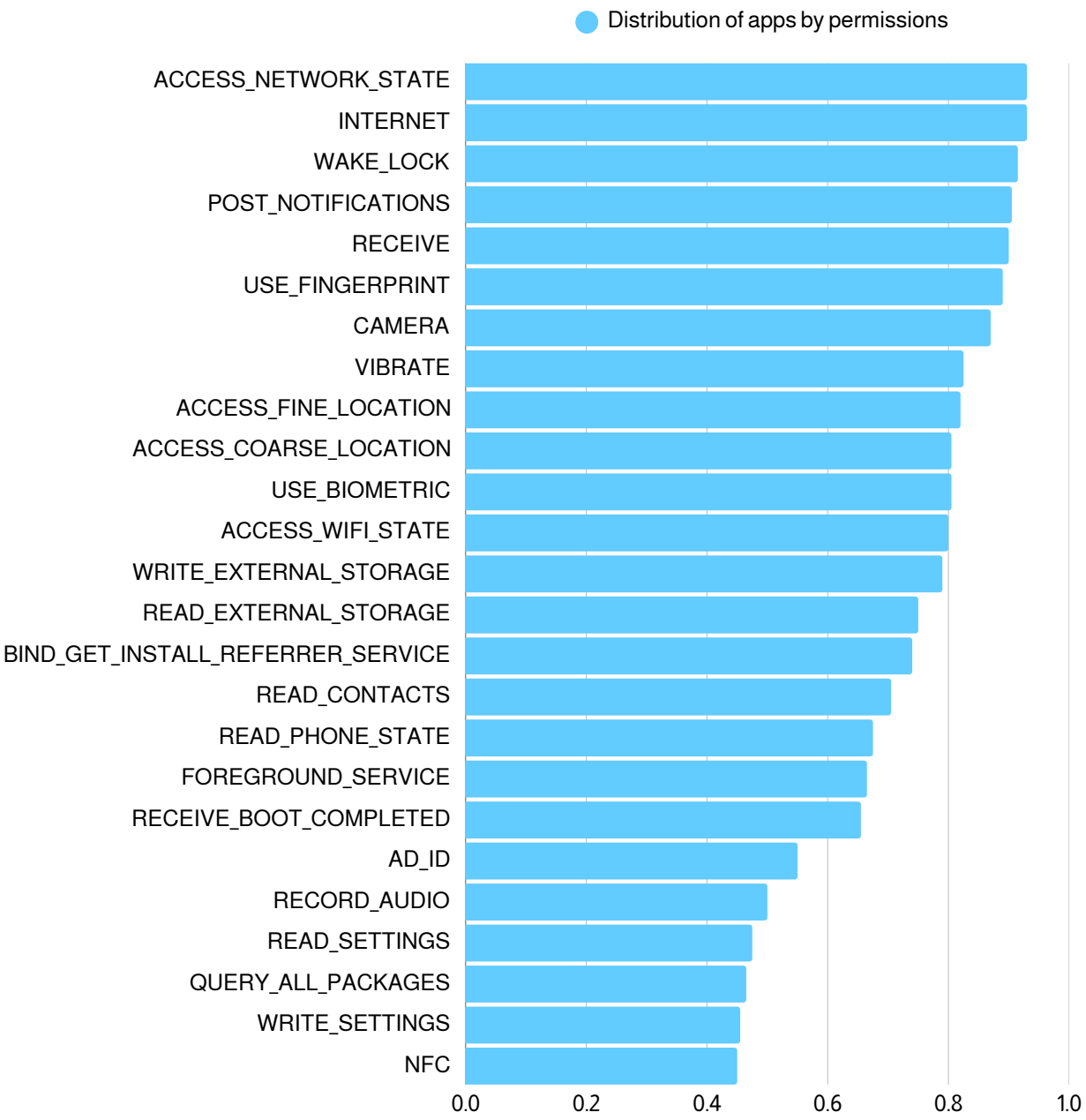


Figure 12: *Distribution of apps by permission*

For iOS, **Camera** is present in nearly 95% of apps, likely used for mobile check deposits, ID verification, and QR code scanning. **Face ID** was found in 80%, supporting biometric login and. **Location When In Use** is widely requested, suggesting its use in fraud detection. Permissions like **Bluetooth Always, User Tracking**, and **Speech Recognition** are seen in a smaller proportion of apps. These could enable features such as in-branch device connectivity, analytics, or accessibility enhancements, but they also raise potential privacy concerns and must be implemented transparently.

On iOS also, we found a small share of applications requesting unusual permissions with no clear banking relevance, such as Apple Music, Health Share, Health Update, Bluetooth Always, and Bluetooth Peripheral.

Such requests may indicate potential overreach or raise user privacy concerns. Banking applications should absolutely request the minimum amount of permissions required to function and keep customers secured. On iOS also, we found a small share of applications requesting unusual permissions with no clear banking relevance, such as Apple Music, Health Share, Health Update, Bluetooth Always, and Bluetooth Peripheral.

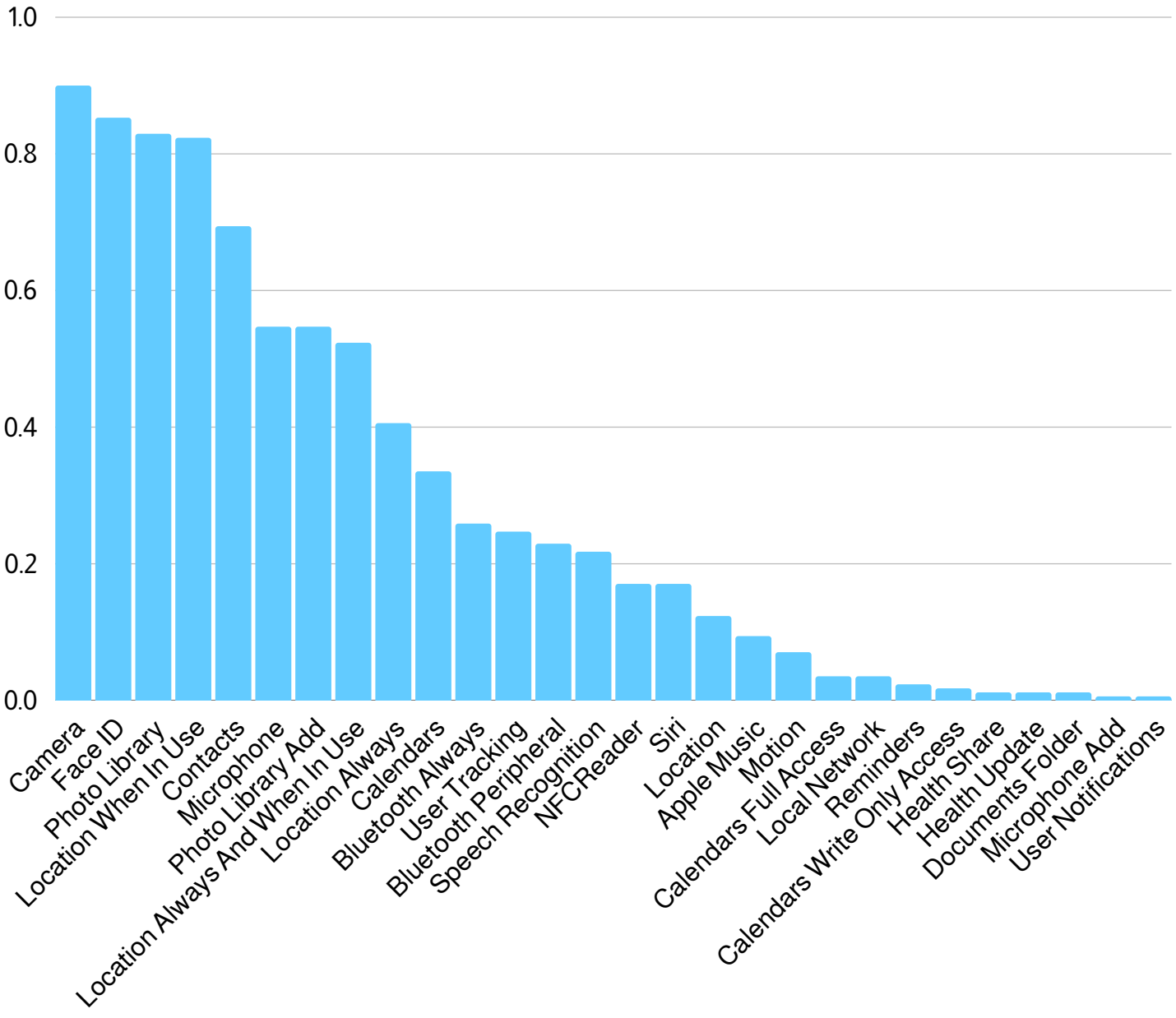


Figure 13: *iOS – Permissions by number of applications*

Exposure

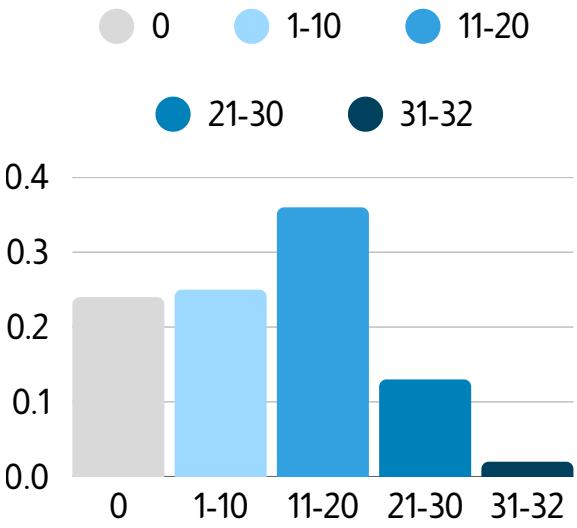


Figure 14

The analysis highlights a significant attack surface within applications. On Android, the data shows that most banking apps use many background services, with nearly 40% having 11 to 20 and 16% having 21 to 30. Only 13.6% have fewer than 10, while 24% don't use any. More services can offer extra features but also increase security risks if not properly protected. This indicates that the extensive use of background services significantly expands the attack surface of many Android banking apps, requiring stronger oversight and stricter security hardening to reduce exposure.

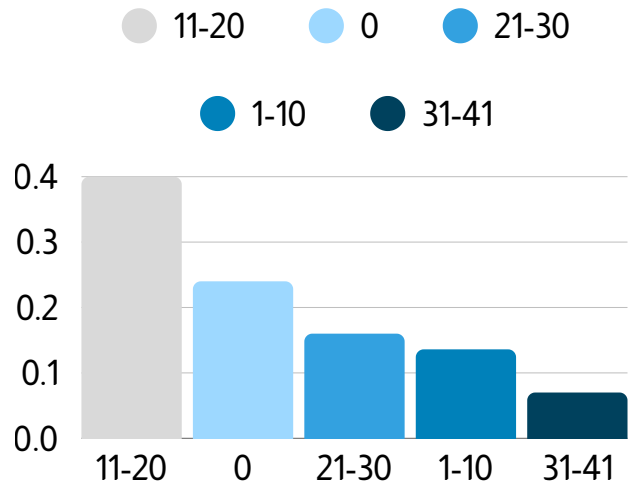


Figure 15

More services can offer extra features but also increase security risks if not properly protected. This highlights the need to carefully check and secure service components in banking apps.

Security Challenges and Vulnerabilities

About the Analysis:

The conducted scans combine static, dynamic, and behavioral analysis techniques to identify potential security weaknesses. Static analysis involves a deep inspection of an application's files and codes without executing the application itself. This technique is instrumental in identifying common security vulnerabilities such as:

- Configuration flaws, such as a debug mode in the Facebook SDK that could leak session information.
- Supply chain vulnerabilities, including outdated dependencies and critically dependency confusion leading to full application compromise.
- Errors like SQL injection or file path traversal.
- Validated hardcoded secrets and weak encryption practices.

Ostorlab's in-house static analysis engine plays a central role in this process. It is specifically designed to analyze mobile application code and leverages both heuristic techniques and a large database of rules for matching vulnerable patterns to detect a wide range of vulnerabilities.

In parallel, dynamic analysis provides a runtime perspective on the application's security. Using Ostorlab's MonkeyTester to run and drive the application in a controlled environment while observing its behavior. During dynamic analysis, the scanner can identify vulnerabilities that only manifest when the app is active, such as:
API hooking and automated interactions to enhance coverage. Backend interception to identify known vulnerabilities and fuzz-testing for potential unknown issues like SQL injection (SQLi) and XML External Entity (XXE) attacks.

With this, Ostorlab's scanner runs a holistic security assessment, covering both latent code-level issues and active runtime vulnerabilities that could be exploited in a live environment.

The scanning of banking apps reveals a concerning trend: The most common issue is hardcoded secrets within applications. This includes API keys, tokens, and credentials embedded directly in code or resource files. Such flaws expose banking infrastructure to unauthorized access, credential stuffing, and service abuse, especially when apps are reverse-engineered by attackers.

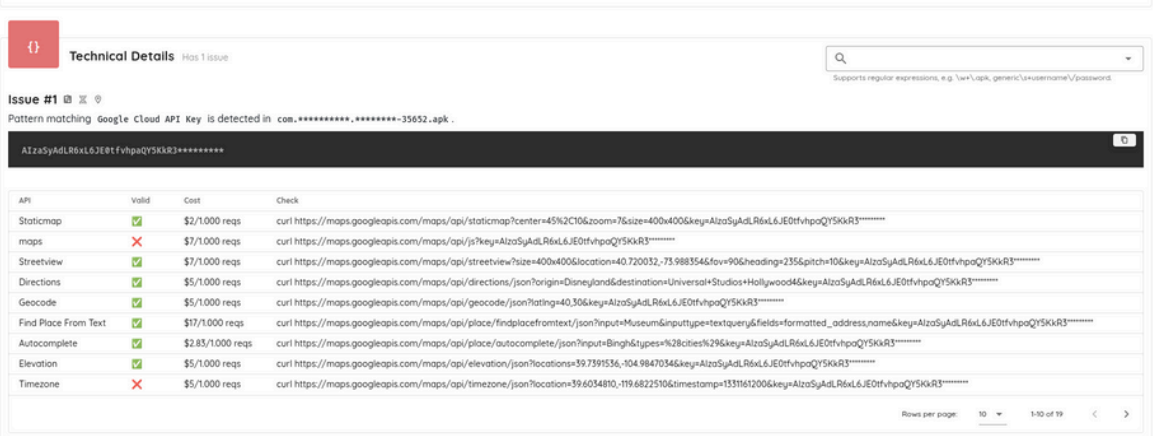


Figure 16: *Leaked Google Cloud API Key*

The second most found issue is the use of outdated vulnerable components. Over 46% apps rely on dependencies known to have critical flaws, highlighting the persistent risks posed by insecure supply chains.

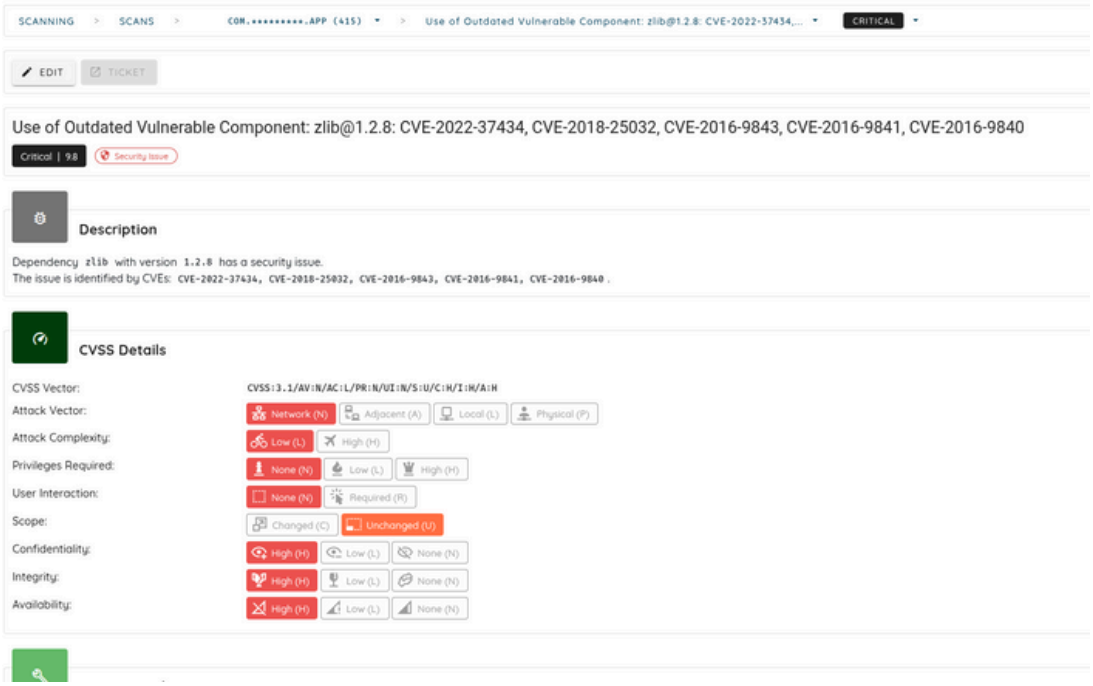


Figure 17: *zlib@1.2.8 vulnerable dependency*

Root detection checks were implemented in more than 40% of apps, suggesting that many applications still rely on basic device trust models.
Worryingly, biometric authentication bypass vulnerabilities were found in over 28% of apps, indicating misconfigured or insufficiently secured biometric flows.
Legacy issues like cleartext HTTP requests persist in roughly 20% apps, exposing user data in transit. This is very worrisome in financial applications, where transport-layer encryption is a fundamental requirement.

Less prevalent but high-impact issues include:
OAuth misconfigurations (around 15% of apps), enabling account takeover via malicious redirects or hijacked URI schemes.
Use of insecure cryptographic algorithms (e.g., ECB mode, MD5, static IVs) limited to a small number of apps.
The dynamic scans lacked authentication to the applications, so any potential post-authentication issues are not included in this report.

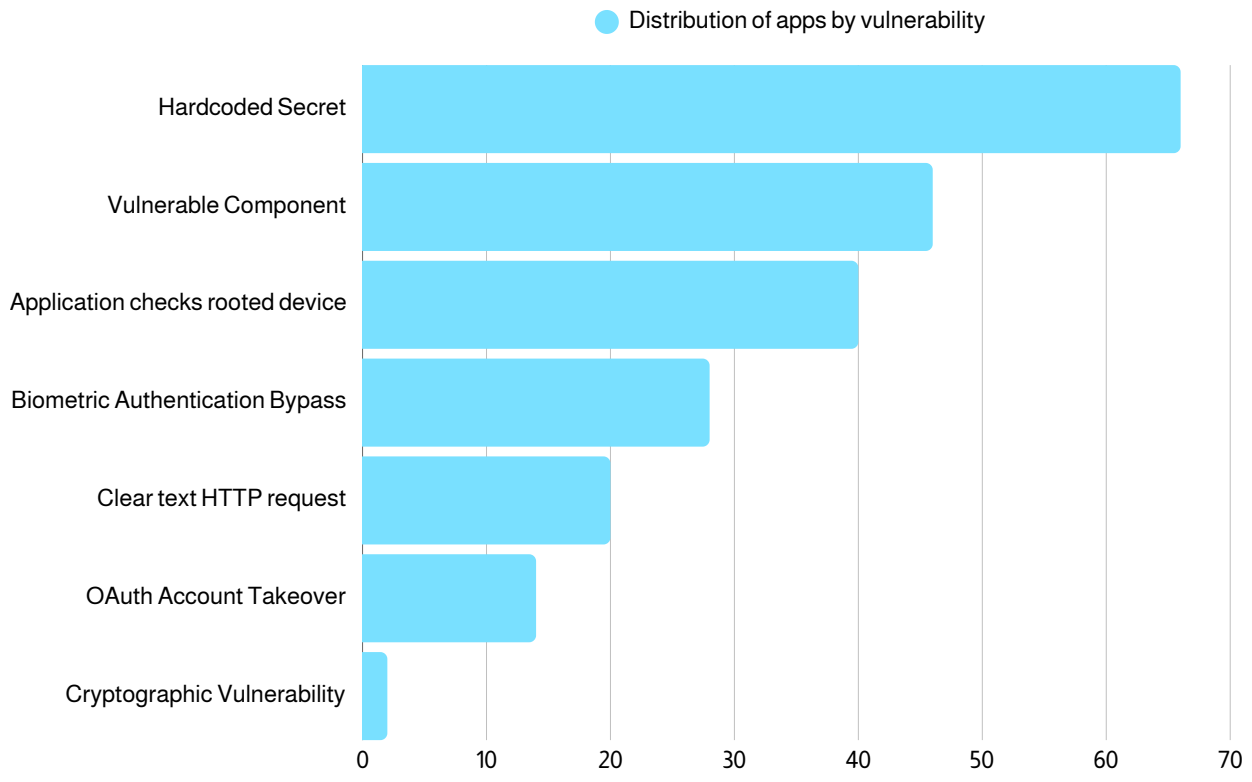


Figure 18

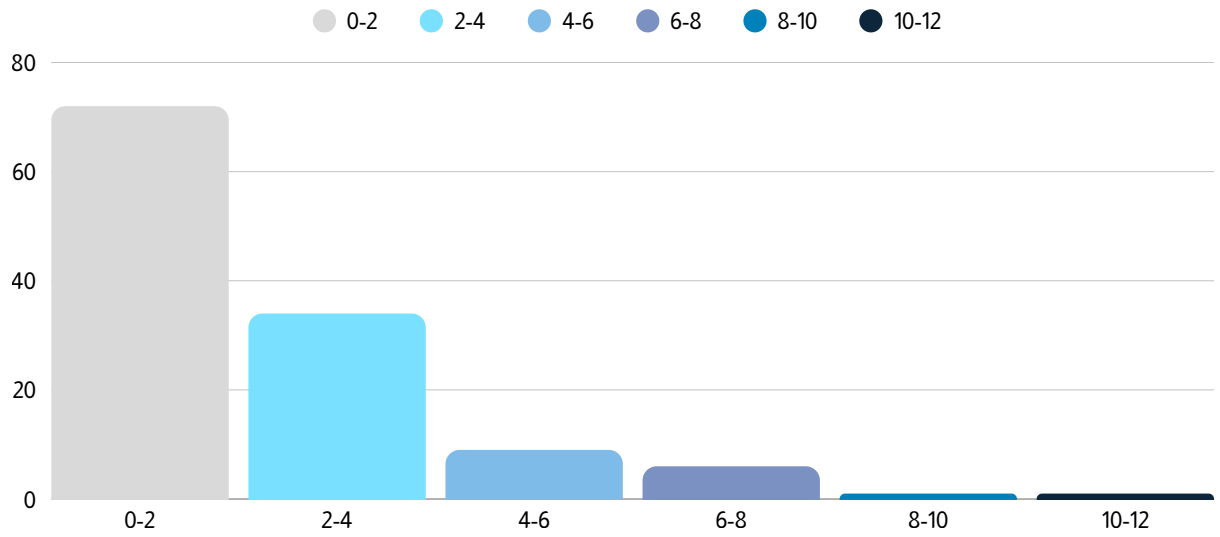


Figure 19: *Distribution of Critical Vulnerabilities - IOS*

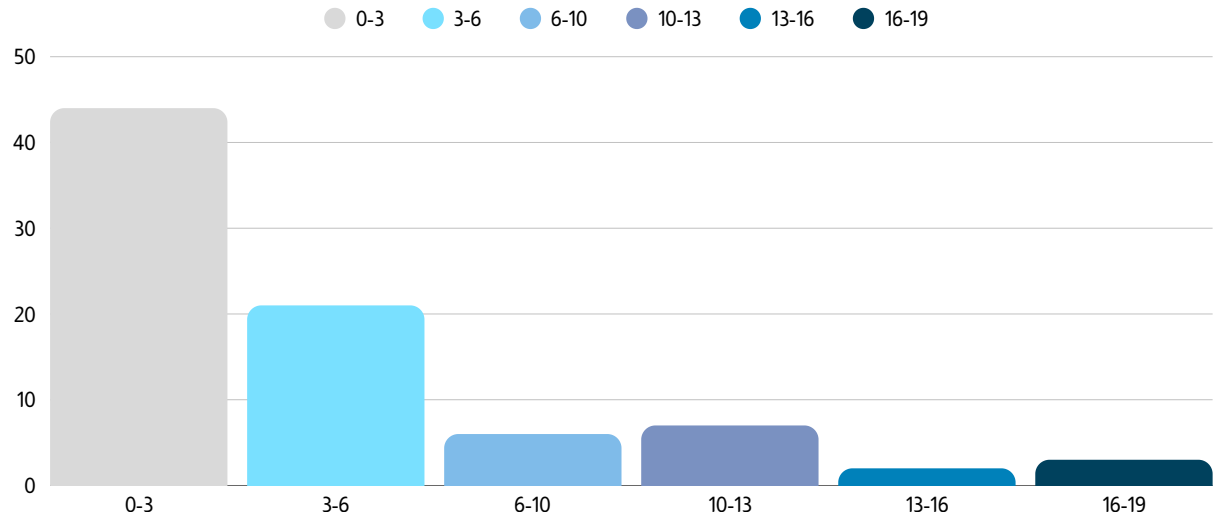


Figure 20: *Distribution of Critical Vulnerabilities - Android*

The majority of both iOS and Android banking apps fall within the 0–2 critical vulnerabilities range, indicating that many have a relatively low count of severe issues, though the presence of even a single critical flaw in a banking app remains a serious security concern. A small but concerning subset of apps containing 10 or more critical vulnerabilities, some reaching up to 19.

1. Application Surface Misconfigurations

Some banking apps exposed unnecessary attack surfaces through unsafe exported components, insecure register receivers, and implicit PendingIntents. These weaknesses may allow malicious applications installed on the same device to intercept sensitive tokens or initiate unauthorized actions within the banking app. These vulnerabilities represent a persistent entry point for on-device fraud and privilege escalation.

The static taint engine also found instances of SQL injection where unsanitized user-controlled input is passed into database queries, potentially enabling unauthorized data access or manipulation

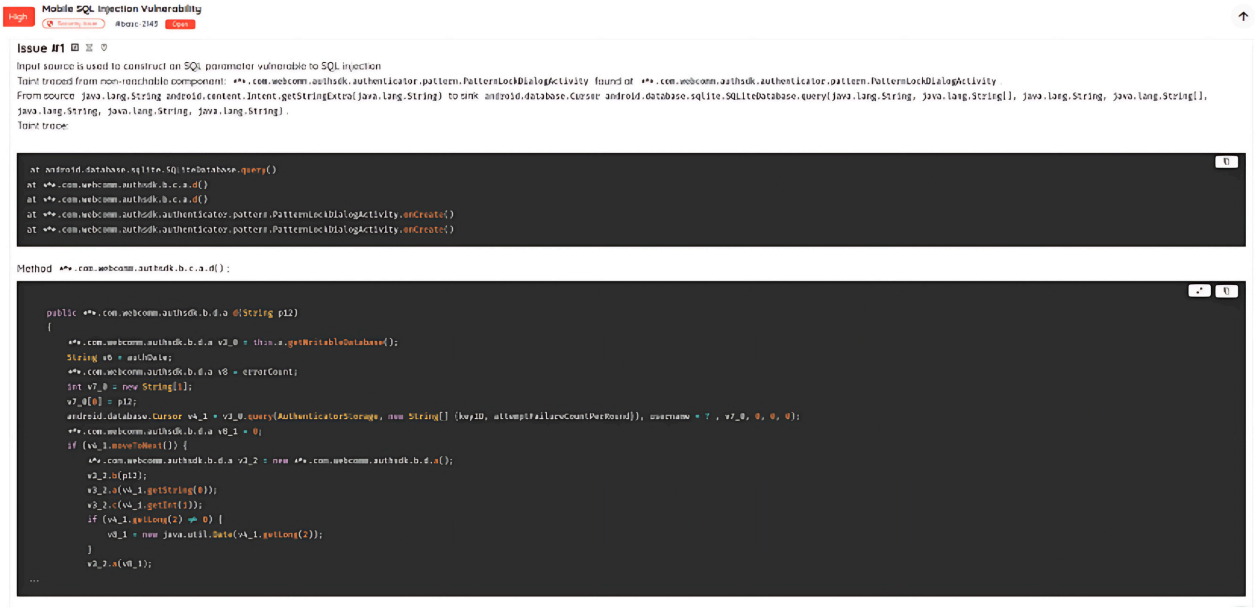


Figure 21 : *SQL injection vulnerability found through taint analysis*

2. Backend Vulnerabilities

The scanned banking apps revealed serious backend weaknesses, including **Remote Command Execution (RCE)** and **File Path Traversal**, which can grant attackers unauthorized access to servers and sensitive customer information. **Directory listing** and **publicly exposed Swagger or GraphQL endpoints** were found, enabling attackers to enumerate internal APIs and plan targeted exploits. In several cases, **credentials exposed in logs** posed an immediate risk of backend compromise, potentially allowing unauthorized access to transaction processing or account management systems.

3. Supply Chain Vulnerabilities

Analysis uncovered widespread reliance on **outdated and vulnerable dependencies,** exposing banking apps to known CVEs that could be exploited to compromise authentication modules or payment flows. Instances of **dependency confusion** presented an opportunity for attackers to introduce malicious code during the build process. Additionally, **insecure object serialization** and **prototype pollution** flaws identified in some apps could lead to remote code execution or privilege escalation, enabling unauthorized financial operations or customer data exfiltration.

4. Traffic & Encryption

Several apps **Vulnerabilities** were found using **cleartext HTTP requests** or susceptible to **HTTPS→HTTP downgrade attacks**, which could expose user credentials, one-time passwords (OTPs), and transaction details to interception. **Misconfigured TLS certificates** (expired, mismatched, or self-signed) were also observed, creating opportunities for Man-in-the-Middle attacks. Furthermore, weak encryption practices, including **insecure cipher suites and hashing algorithms**, weakened the confidentiality and integrity of sensitive banking data in transit.

5. Authentication & Session Management

The scans identified flaws such as **OAuth account takeover via custom scheme hijacking, intent redirection**, and **biometric authentication bypasses** that could lead to unauthorized access to customer accounts and fraudulent transactions. **Hardcoded keys and tokens** were found in several apps, while improper session handling increased the risk of hijacking active banking sessions. **Tapjacking vulnerabilities** could further trick users into unknowingly authorizing high-value transfers.

Emerging Trends in Mobile Banking App Security

Mobile banking is rapidly evolving, driven by advances in technology, shifts in customer expectations, and expanding digital ecosystems. While these innovations bring greater convenience and personalization, they also expand the threat landscape. Cyberattacks are evolving at a similar pace, often exploiting the same technologies that enable progress. From integrating financial services into non-banking platforms to adopting AI-driven fraud detection, banks must balance innovation with robust safeguards to protect user trust and meet regulatory demands.

Security Built into Development

Mobile banking apps now integrate security checks into every step of the development process (DevSecOps). This includes automated scans for vulnerabilities, following secure coding standards (e.g., OWASP Mobile Top 10), and using AI tools to find bugs early.

Biometrics & Blockchain Risks

Biometrics (like fingerprints or face scans) and blockchain can improve security but also introduce new problems. If biometric data is stolen, it cannot be changed, and blockchain systems can still have weaknesses in how they're built or integrated.

Rise of Neobanks and Digital-Only Financial Institutions

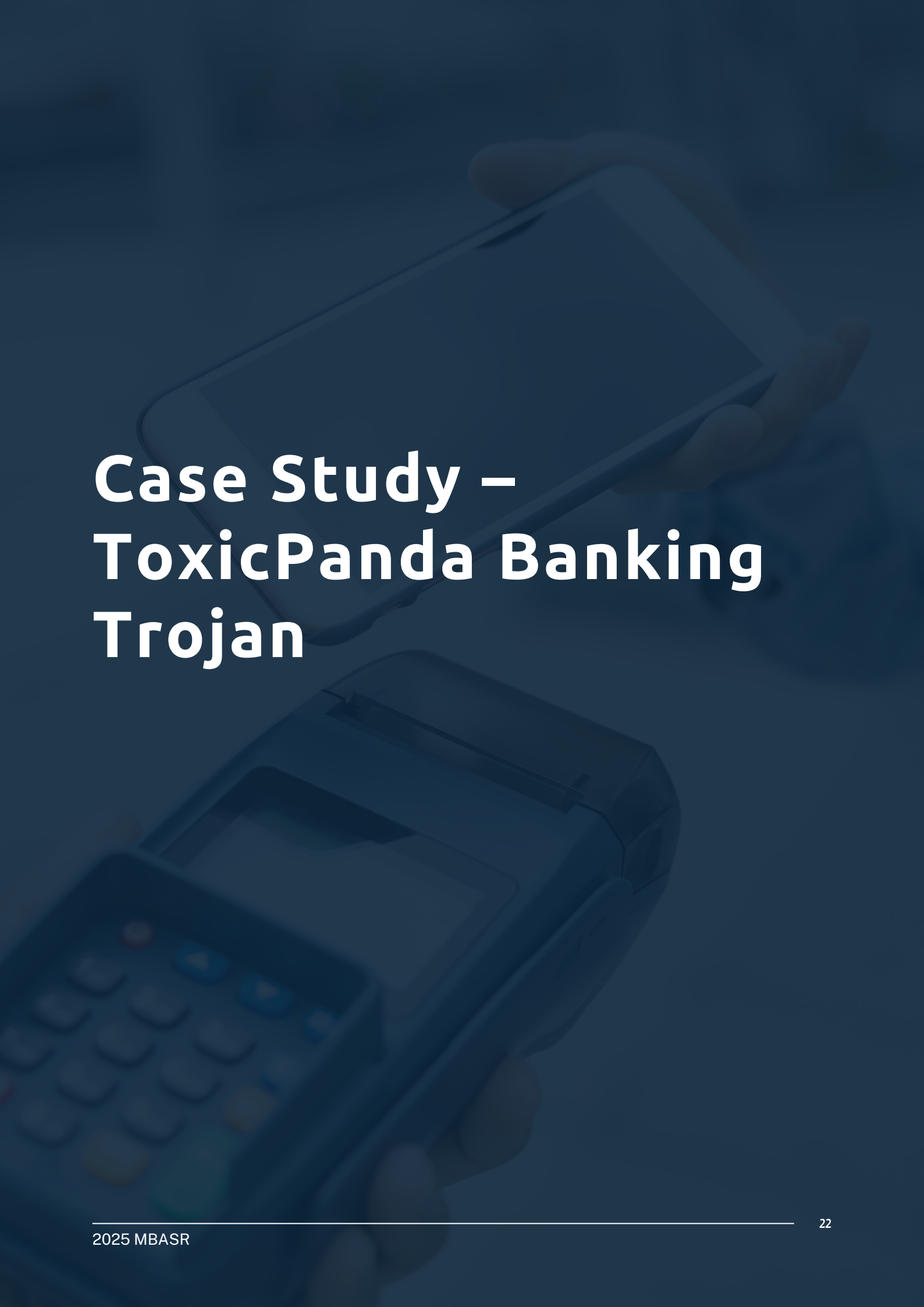
Neobanks are reshaping traditional banking by operating entirely online and challenging existing regulatory frameworks. However, their fully digital nature can introduce risks, including compliance gaps, potential impacts on financial stability, and erosion of customer trust in the event of technical failures or security breaches.

Banking Beyond Traditional Apps

Banking services are increasingly integrated into third-party platforms, whether through regulated open banking APIs or embedded directly into non-financial apps like e-commerce, ride-hailing, or social media. While this creates convenience and new service opportunities, it also widens the attack surface and increases data privacy risks. Both regulated and unregulated integrations require strong security controls, strict third-party vetting, and robust consent management to prevent breaches and misuse.

AI in Banking Apps

Artificial intelligence can improve banking services, but it also introduces risks, such as biased algorithms, unclear decision-making, and the possibility of AI systems being tricked or attacked. Criminals are now using AI to power more sophisticated attacks, including deepfake voices and videos to impersonate customers or bank staff, bypassing traditional verification methods.



Case Study – ToxicPanda Banking Trojan

Context

Over the past decade, mobile malware has evolved from crude scams and adware into sophisticated threats capable of performing credential theft, surveillance, remote access, and unauthorized financial operations. In this shifting threat landscape, banking trojans have emerged as one of the most damaging categories, targeting both users and financial institutions through credential interception, SMS hijacking, and app overlay attacks.

ToxicPanda is a newly discovered Android banking trojan identified by Cleafy in late 2024. Its main goal is to initiate **money transfers** from compromised devices via **account takeover** (ATO) using **On-Device fraud** (ODF), and by leveraging Android's Accessibility Services to bypass security controls, intercept SMS messages and OTPs, and manipulate banking apps in real time.

The malware was first observed targeting users in Europe and Latin America, with 87.4 infected devices reported across the provided countries. Approximately 65.0% of these were in Italy, followed by 21.4% in Portugal. While the remaining share was distributed across various countries. [Source: Cleafy]

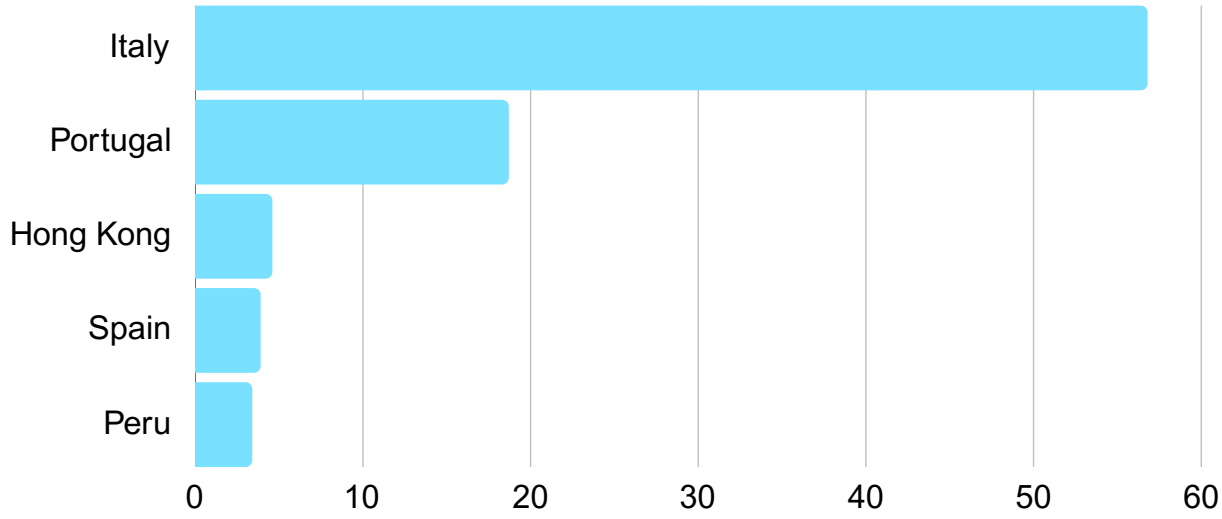


Figure 22: *Victims' geographic distribution*

How Toxic Panda Works

ToxicPanda's primary goal is on-device fraud (ODF), and it achieves this through a combination of phishing overlays and credential harvesting.

ToxicPanda disguises itself as a trusted legitimate application such as Google Chrome. Upon installation, the malicious application requests the user to enable Android's accessibility services. This gives the malicious app a free pass to control the entire device. Once accessibility privileges are granted, It can:

- Read and interact with screen contents
- Simulate user input (clicks, swipes)
- Automatically grant itself additional permissions without user interaction
- Silencing notifications
- Perform overlay attacks

Next thing, the application updates the settings, gives itself necessary permissions and downloads resources for its phishing overlays from the threat actors' servers.
These overlays are identical to the legitimate banking apps. The malware maintains a local mapping of targeted banking apps and associated phishing templates. When the user opens a banking app, the malware renders an identical WebView allowing it to capture user credentials which are exfiltrated to the TAs servers.

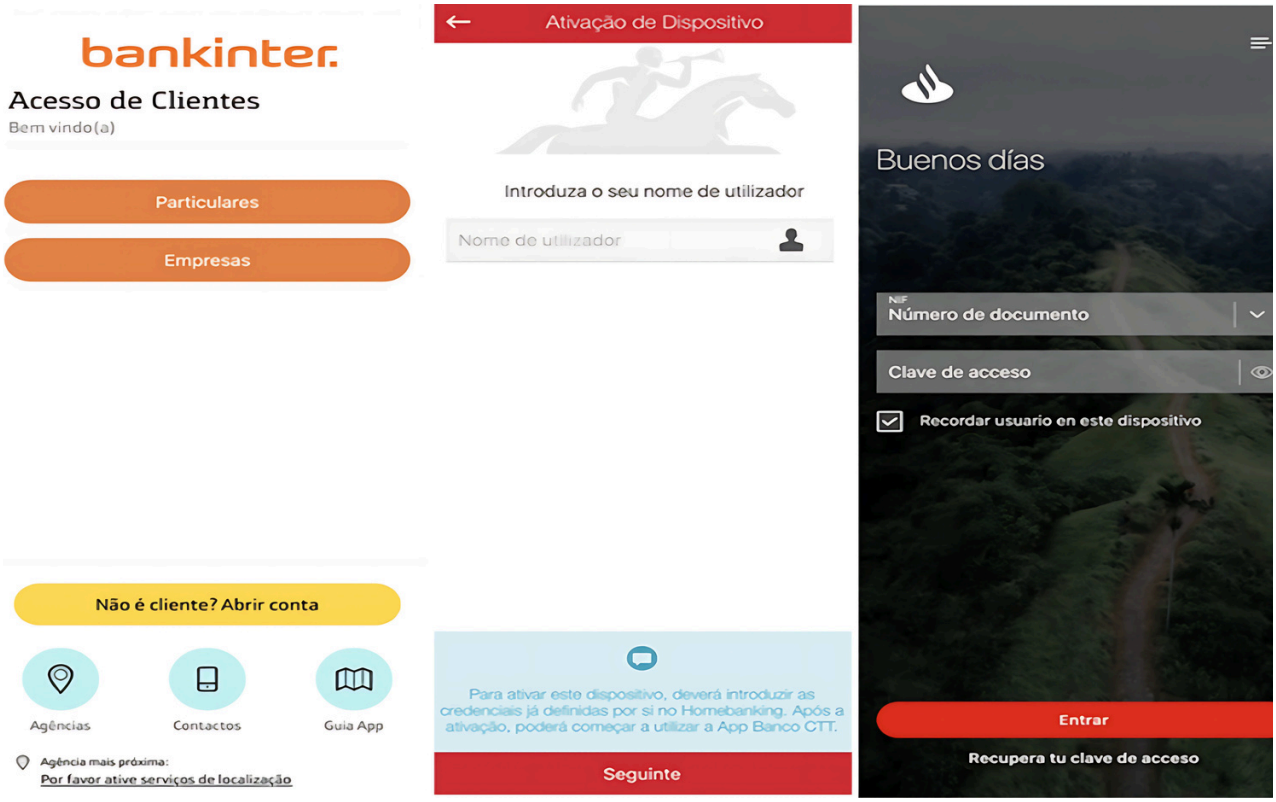


Figure 23: *Example of phishing UI's identical to the legitimate apps' [Source: bitsight]*

Example of phishing UI's identical to the legitimate apps' [Source: bitsight]

ToxicPanda also grants attackers' remote access to the infected device, enabling them control of the device from any location. Combined with credential theft, OTP and message interceptions, plus Remote control, threat actors can initiate banking transactions on behalf of the user without them noticing.

ToxoPanda also employs some persistence actions to remain active at all time: It re-registers itself on **BOOT_COMPLETED** to auto-start after device reboot, and actively monitors the Accessibility Service status and prompts the user to re-enable it if disabled,

Command-and-Control Infrastructure & Command Set

ToxicPanda's command-and-control (C2) system is the core of its remote operations. Once the malware is installed on a victim's device, it connects to the attacker's infrastructure to receive instructions, download updates, and send stolen data.

Primary C2 servers are hardcoded into the malware, but they're hidden using encryption and obfuscation so they're not easy to spot by analysts. But if these servers were to be blocked or taken down, the malware falls back to a **Domain Generation Algorithm (DGA)** to create new domains on the fly. Attackers can register new domains from the same pattern and the malicious app stays connected.

Once connected, the infected device receives commands in JSON format. These are lightweight, easy to parse, and customizable. ToxicPanda supports a growing list of commands, which can be updated as needed. Some of the main ones include:

- **updatePageRule:** Tells the malware which apps to watch for and which phishing overlay to show.
- **setDomain:** Dynamically changes the C2 endpoint.
- **openLayer:** Triggers a WebView-based overlay for a specific banking app to phish credentials.
- **get_sms:** Extracts and forwards all incoming SMS messages (useful for intercepting OTPs).
- **get_contacts:** Dumps the user's contact list.
- **push_log:** Sends logs of user interactions, like inputs or accessed apps.
- **startInject / stopInject:** Controls when and which injection or overlay attack should run.

These commands give attackers full remote control over the infected device. They can launch phishing screens, steal messages, harvest credentials, and even silently interact with banking apps to perform fraud. All without alerting the user.

Conclusion

Mobile banking stands as both the dynamic heart of modern finance and its front line of cyber risk.

Ostorlab's analysis of over 500 mobile banking apps paints a picture of a rapidly advancing digital ecosystem that is both deeply entrenched in daily financial life and increasingly under threat.

While banks have made notable strides in deploying secure, feature-rich apps, the data shows that many still rely on outdated libraries, weak encryption practices, and exposed attack surfaces.

The vulnerabilities uncovered aren't theoretical—they're real risks affecting millions of users across platforms.Yet amid the concern, there are signs of progress. Frequent app updates, growing adoption of biometric authentication, hardware-backed security, and on-device threat detection are all steps in the right direction.

The banking industry is clearly investing in stronger defenses, but sophistication on the attacker side is growing just as fast—powered now by AI, deepfakes, and advanced malware.

What's clear is this: security can no longer be an afterthought or a compliance checkbox. It must be baked into the design, development, and deployment of every banking application. With regulatory pressure mounting and user trust on the line, the institutions that succeed will be those that embrace a proactive, holistic approach to mobile security.

Ostorlab

2025