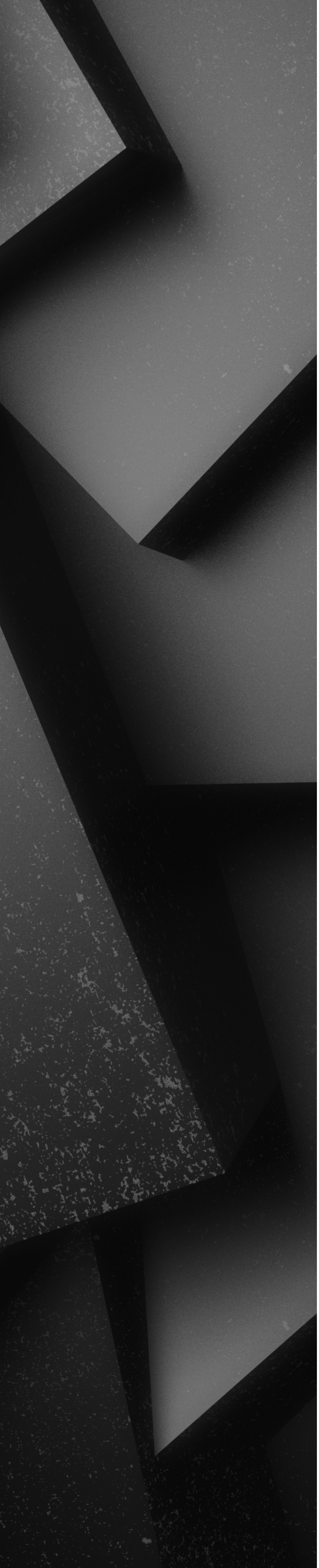
# Vulnerabilities and threats in mobile applications

2019

ptsecurity.com

## Contents



## Introduction

In 2018, mobile apps were downloaded onto user devices over 205 billion times. Data by Marketing Land indicates that 57 percent of total digital media time is spent on smartphones and tablets. More often than not, our daily lives depend on apps for instant messaging, online banking, business functions, and mobile account management. According to Juniper Research, the number of people using mobile banking apps is approaching two billion—around 40 percent of the world's adult population.

Developers pay painstaking attention to software design in order to give us a smooth and convenient experience. People gladly install mobile apps and provide personal information, but rarely stop to think about the security implications.

Positive Technologies experts regularly perform security analysis of mobile applications. This report summarizes the findings of their work performing security assessment of mobile apps for iOS and Android in 2018.

# Executive summary

- High-risk vulnerabilities were found in 38 percent of mobile applications for iOS and in 43 percent of Android applications.
- Most security issues are found on both platforms. Insecure data storage is the most common issue, found in 76 percent of mobile applications. Passwords, financial information, personal data, and correspondence are at risk.
- Hackers seldom need physical access to a smartphone to steal data: 89 percent of vulnerabilities can be exploited using malware.
- Most cases are caused by weaknesses in security mechanisms (74% and 57% for iOS and Android apps, respectively, and 42% for server-side components). Because such vulnerabilities creep in during the design stage, fixing them requires significant changes to code.
- Risks do not necessarily result from any one particular vulnerability on the client or server side. In many cases, they are the product of several seemingly small deficiencies in various parts of the mobile application. Taken together, these oversights can add up to serious consequences, including financial losses for users and reputational damage to the developer.
- Many cyberattacks rely on user inattention. Escalated privileges or side-loaded software can pave the way for a damaging attack.

# How mobile applications work

Mobile applications are at the epicenter of current development trends. Most of these applications have a client–server architecture. The client runs on the operating system, which is most frequently Android or iOS. This client is downloaded to the device from the app distribution platforms, where developers publish their wares. As perceived from the user's point of view, the client installed on the smartphone is the mobile application. This is what the user interacts with to make purchases, pay bills, or read emails. But in fact, there is also another component: the server, which is hosted by the developer. Often this role is performed by the same software that is responsible for generating and processing content on the site. In other words, most often the server-side component is a web application that interacts with the mobile client over the Internet by means of a special application programming interface (API). So in reality we can regard the server as the more important component. It is where information is stored and processed. The server is also responsible for synchronizing user data between devices.

Modern mobile OSs come with various security mechanisms. By default, an installed app can access only files in its own sandbox directories, and user rights do not allow editing system files. Nevertheless, errors made by developers in designing and writing code for mobile applications cause gaps in protection and can be abused by attackers.

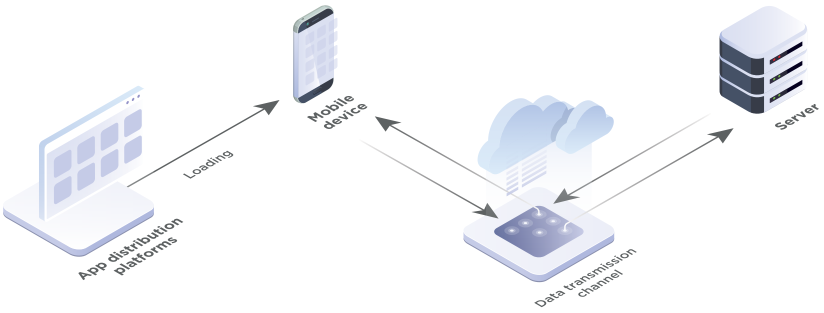
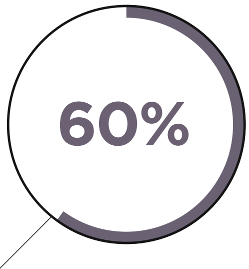


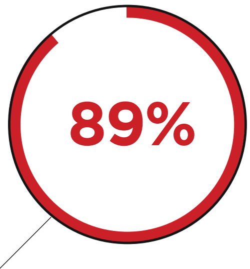
Figure 1. Client–server interaction in a mobile application

Comprehensive security checks of a mobile application include a search for vulnerabilities in the client and server, as well as data transmission between them. In this report, we will cover all three aspects. We will also talk about threats to users, including threats arising from interaction between the client and server sides of mobile applications. Methodology and the source dataset are described at the end of the report.

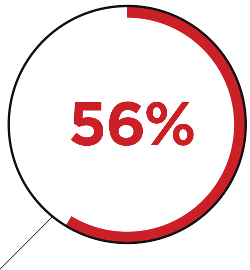
## Client-side vulnerabilities



of vulnerabilities are on the client side



of vulnerabilities can be exploited without physical access



of vulnerabilities can be exploited without administrator rights (jailbreak or root)

Android applications tend to contain critical vulnerabilities slightly more often than those written for iOS (43% vs. 38%). But this difference is not significant, and the overall security level of mobile application clients for Android and iOS is roughly the same. About a third of all vulnerabilities on the client side for both platforms are high-risk ones.

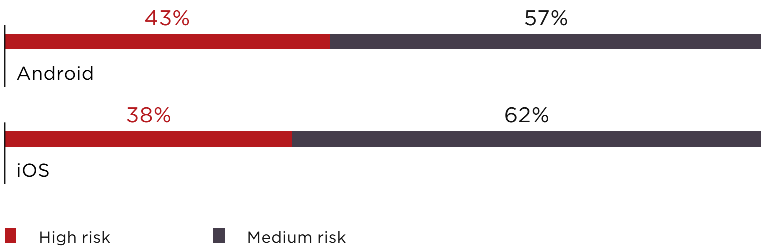


Figure 2. Maximum risk level of vulnerabilities (percentage of client-side components)

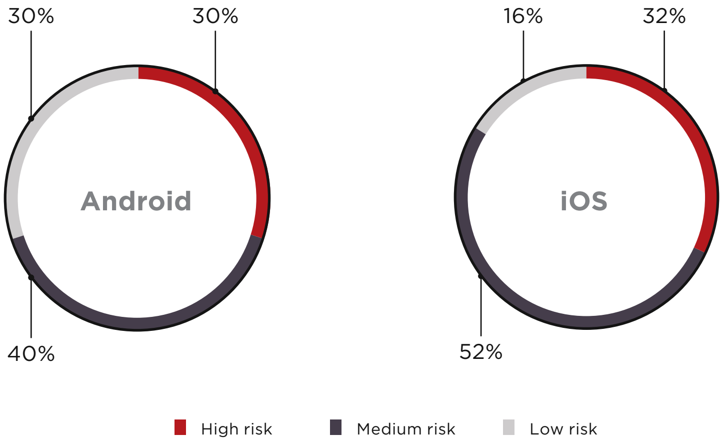


Figure 3. Vulnerabilities by severity

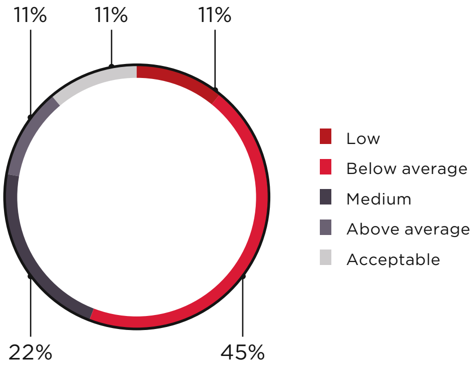


Figure 4. Security of client-side components (percentage of mobile applications)

**38% Android**
**22% iOS**

Percentage of applications with insecure interprocess communication

Insecure interprocess communication (IPC) is a common critical vulnerability allowing an attacker to remotely access data processed in a vulnerable mobile application. Let us review the workings of IPC in greater detail.

Android provides Intent message objects as a way for application components to communicate with each other. If these messages are broadcasted, any sensitive data in them can be compromised by malware that has registered a BroadcastReceiver instance.

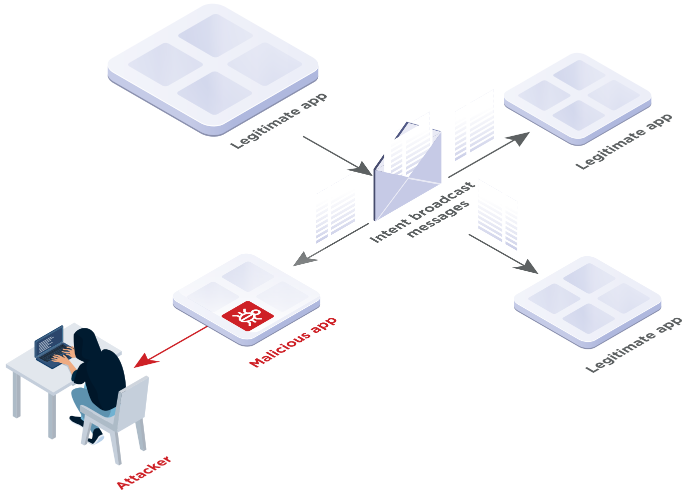
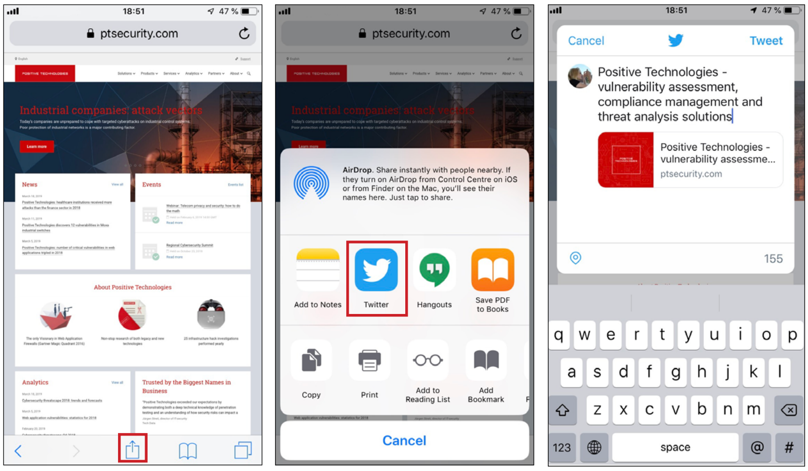


Figure 5. Insecure interprocess communication on Android

## Recommendations for developers

Use LocalBroadcastManager to send and receive broadcast messages not intended for third-party applications

Interprocess communication is generally forbidden for iOS applications. However, there are times when it is necessary. In iOS 8, Apple introduced App Extensions. With them, apps can share their functionality with other apps on the same device. For instance, social networking apps can provide quick in-browser sharing of content.



AirDrop. Share instantly with people nearby. If they turn on AirDrop from Control Centre on iOS or from Finder on the Mac, you'll see their names here. Just tap to share.

Positive Technologies - vulnerability assessment, compliance management and threat analysis solutions

Positive Technologies - vulnerability assessme...
ptsecurity.com

**Host App**
(in this example, Safari browser)

**Containing App**
(in this example, Twitter)

**App Extension**
(in this example, extension for Twitter)

Figure 6. Example of an app extension for Twitter

Deep linking is a common way for developers to implement communication between an app extension and its containing app. In this case, the app is called by a specific URL scheme registered in the system. During installation, the containing app registers itself as the handler for schemes listed in Info.plist. Such schemes are not tied to an application. So if the device contains a malicious app that also handles the same URL scheme, there is no telling which application will win out. This opens up opportunities for attackers to stage phishing attacks and steal user credentials.

## Recommendations for developers

If you need to use links for interaction between components, use universal links

Insecure interprocess communication arises during design of communication interfaces between app components, and is classified as an error in implementation of security mechanisms. Errors in security mechanisms were the cause of 74 percent of vulnerabilities in iOS applications and 57 percent of vulnerabilities in Android applications.

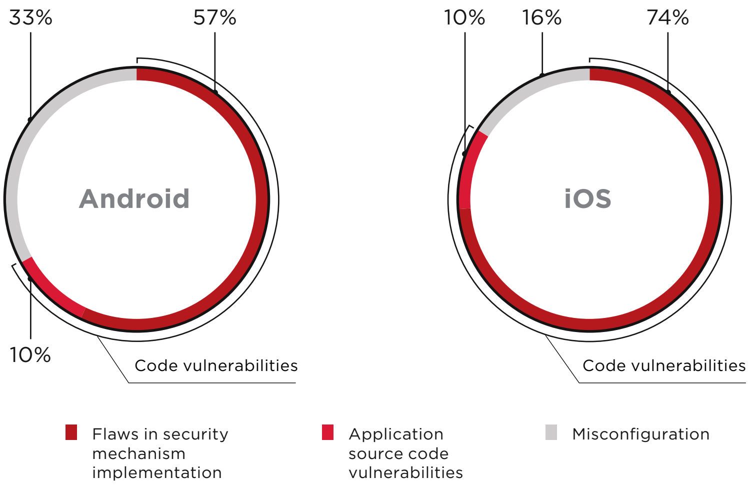


Figure 7. Vulnerabilities by type

**The developer of the AI.type virtual keyboard, for example, has been collecting sensitive data from mobile devices. This fact was discovered after the leak of a database containing information on 31 million users**

In 2018, when analyzing mobile applications for iOS, we encountered the failure by developers to restrict use of custom keyboard extensions. Since iOS 8, Apple has allowed the use of third-party keyboards (Android already had and continues to support them). It should be noted that iOS places more stringent restrictions on keyboard use than does Android. But if the user allows network interaction, Apple cannot control what the keyboard developers do with keystroke data.

## Recommendations for developers

To disable use of third-party keyboards within an application, implement the shouldAllowExtensionPointIdentifier method within the application's UIApplicationDelegate

If the application accepts input of sensitive data such as financial information, implement a custom keyboard. This will secure the app from attacks that manipulate the system keyboard

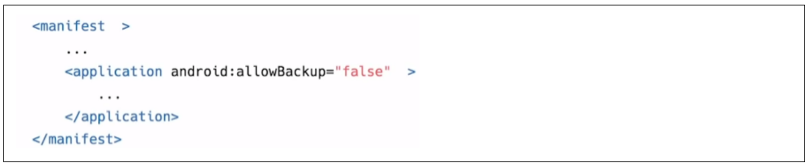
**25%**

**of Android applications** enable backups by setting android:allowBackup to "true"

One third of vulnerabilities in Android mobile applications stem from configuration flaws. For example, our experts when analyzing AndroidManifest.xml often discover the android:allowBackup attribute set to "true". This allows creating a backup copy of application data when the device is connected to a computer. This flaw can be used by an attacker to obtain application data even on a non-rooted device.

## Recommendations for developers

Disable app from being backed up by setting the android:allowBackup directive to "false"



```
<manifest  >
    ...
    <application android:allowBackup="false"  >
        ...
    </application>
</manifest>
```

Figure 8. Disabling backups in AndroidManifest.xml

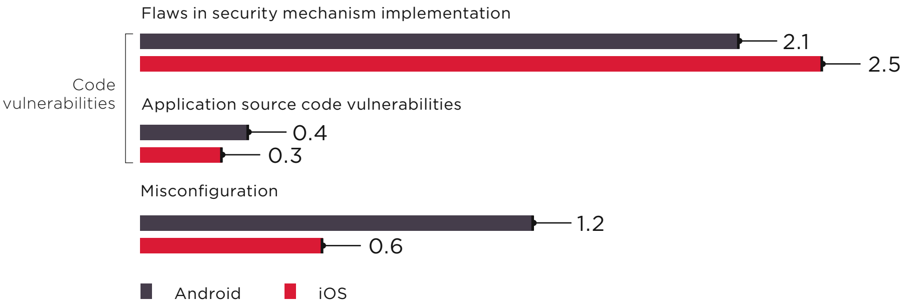


Figure 9. Average number of vulnerabilities per client application

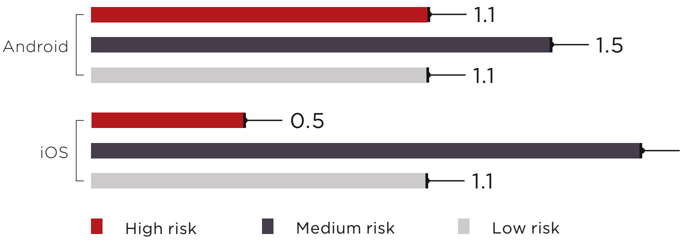


Figure 10. Average number of vulnerabilities per client application

In security assessment, our experts scour applications for the vulnerabilities that are most typical for each platform. At the same time, in most cases developers make similar errors in both Android and iOS apps. That is why in this document, we have provided combined vulnerability statistics without per-platform breakdowns.

Mobile devices store data such as geolocation, personal data, correspondence, credentials, and financial data, but secure storage of that data by mobile applications is often overlooked. Insecure Data Storage is second in the OWASP Mobile Top 10–2016 rating. This vulnerability was found in 76 percent of mobile applications.

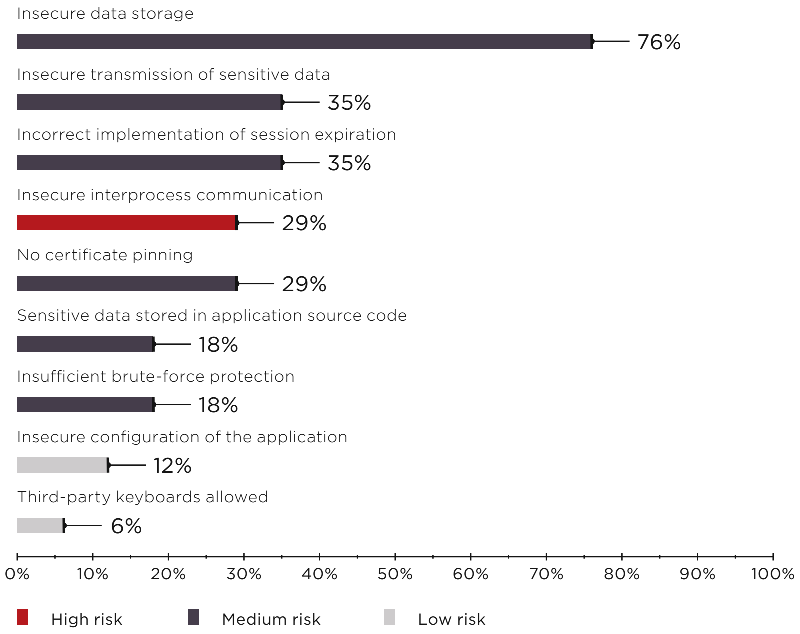


Figure 11. Mobile application vulnerabilities (percentage of client-side components)

Mobile devices allow viewing recently used applications and quickly switching between them. After the app moves to the background, the OS captures a snapshot of the app's current state for this purpose. Direct access to these snapshots is available only on rooted devices. It is important to make sure that snapshots do not contain sensitive data. For instance, if the owner was just using a mobile bank app, the snapshot could contain a card number. These snapshots could be stolen if the device is infected.

## Recommendations for developers

Use a special background image to mask sensitive data on the application screen

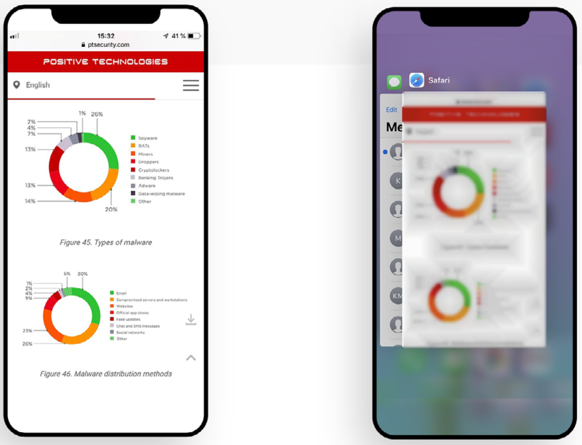

15:32
ptsecurity.com
POSITIVE TECHNOLOGIES
English
Figure 45. Types of malware
Figure 46. Malware distribution methods
Safari

Figure 12. Example of hiding application's contents

**41%**

**of mobile applications** check authentication data on the client side

Many mobile applications use a four- or six-digit PIN code for authentication. There are several ways of implementing PIN code verification when the user logs in. Performing this check on the client side is not secure: this would require that the PIN code be stored on the mobile device, which increases the risk of a leak. Authentication data is stored insecurely in 53 percent of mobile applications.

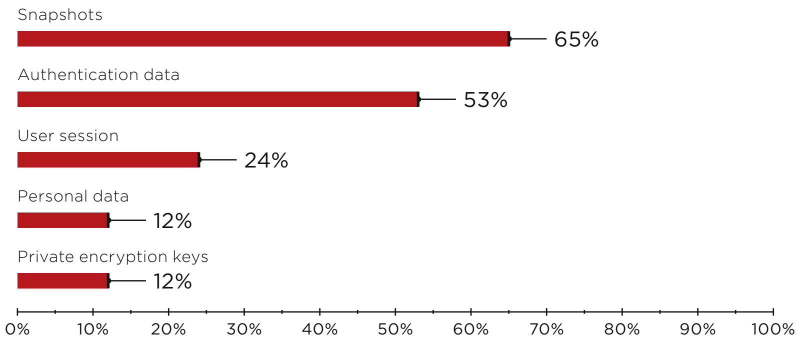


Figure 13. Top five leaks in client-side components (percentage of vulnerable applications)

PIN codes and passwords should be verified on the server, by passing credentials as hashes. Hash functions require a salt (set of random characters) to increase security. Often our experts find the salt and other sensitive data in the source code, which reduces application security. A good alternative to storing the salt in the source code is generating it dynamically when the user logs on, based on the data the user enters. However, this method is secure only if the data has high entropy.

## Recommendations for developers

Modern devices tend to use biometrics (Touch ID or Face ID) for authentication in applications. In this case, the PIN code is stored on the device. Local storage of sensitive data is acceptable only in special directories with encryption. Android has a key vault called Keystore;
iOS has Keychain

## Server-side vulnerabilities

As noted already, the server component of a mobile application is, in essence, a web application. Web application vulnerabilities have been analyzed in our previous report. However, here we will take a closer look at vulnerabilities in the server components of mobile applications.

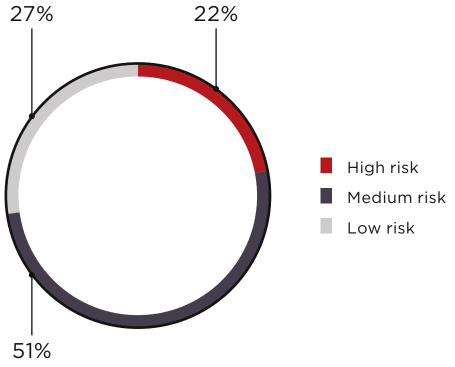


Figure 14. Vulnerabilities by severity

**In August 2018 hackers stole personal data for 20,000 users of the Air Canada mobile app**

According to McAfee, the amount of malware for mobile devices keeps growing. Every quarter 1.5 to 2 million new malware variants are discovered. As of the end of 2018, there were over 30 million malware variants in total. Constant growth in the amount and variety of malware for mobile devices has fueled the popularity of attacks on client-side components. Server vulnerabilities are no longer the main threat to mobile applications. Back in 2012, Weak Server Side Controls ranked second in the OWASP Mobile Top 10 rating. In 2016, server-side vulnerabilities did not even make the list of the top 10 most common threats. However, risks related to server flaws still remain, and major data leaks due to server vulnerabilities continue to occur. Our study shows that the server side is just as vulnerable as the client side: 43 percent of server-side components have a security level that is "low" or "extremely poor," and 33 percent contain critical vulnerabilities.

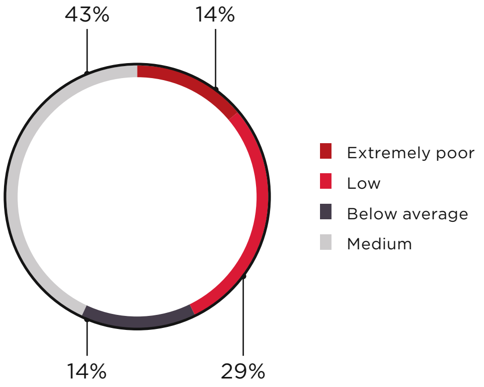


Figure 15. Security of server-side components (percentage of systems)

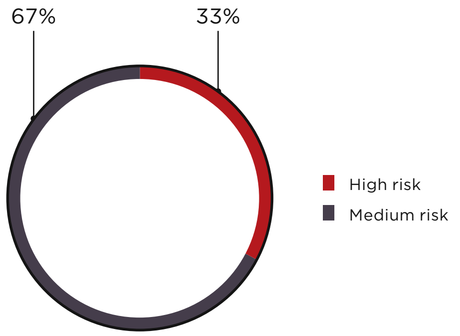


Figure 16. Maximum risk of vulnerabilities found (percentage of server-side components)

Server-side components contain vulnerabilities both in application code and in the app protection mechanisms. The latter include flaws in the implementation of two-factor authentication. Let us consider one vulnerability our experts encountered in an application. If two identical requests are sent to the server one right after the other, with a minimal interval between them, one-time passwords are sent to the user's device both as push notifications and via SMS to the linked phone number. The attacker can intercept SMS messages and impersonate the legitimate user, for instance, by cleaning out the user's bank account.

## Recommendations for developers

It is not necessary to send one-time passwords twice in both SMS messages and push notifications. Instead, use the password delivery method selected by the user

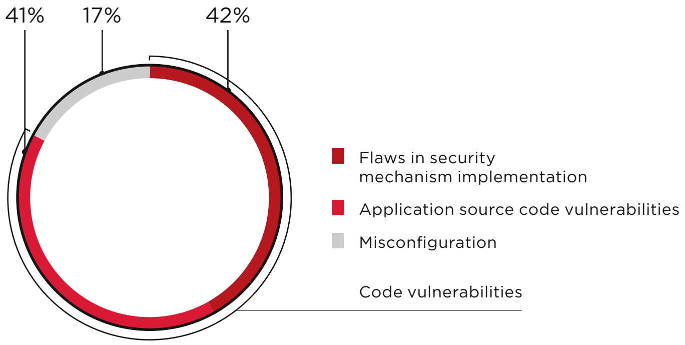


Figure 17. Vulnerabilities by type

The average server-side component contains five code vulnerabilities and one configuration vulnerability. Configuration flaws include disclosure of sensitive information in error messages, fingerprinting in HTTP headers, and TRACE availability.

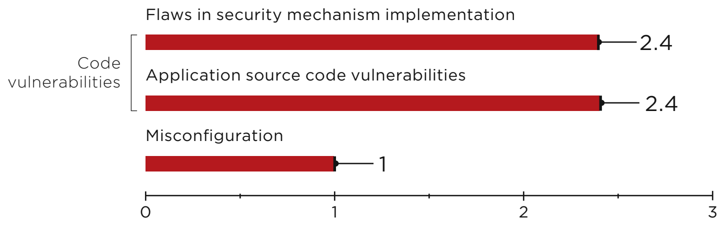


Figure 18. Average number of vulnerabilities per server-side component

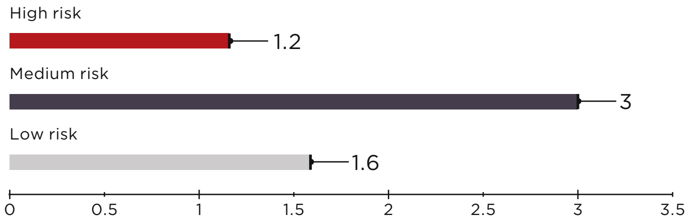


Figure 19. Average number of vulnerabilities per server-side component

When support for TRACE requests is combined with a Cross-Site Scripting (XSS) vulnerability, an attacker can steal cookies and gain access to the application. Because the server-side component of the mobile application tends to share the same code as the website, Cross-Site Scripting allows attacking users of the web application.

## Recommendations for developers

TRACE can be used to bypass cookie protection with the httpOnly flag. Disable handling of TRACE requests

Insufficient authorization issues were found in 43 percent of server-side components. This is one of the most common high-risk vulnerabilities, accounting for 45 percent of all critical vulnerabilities.

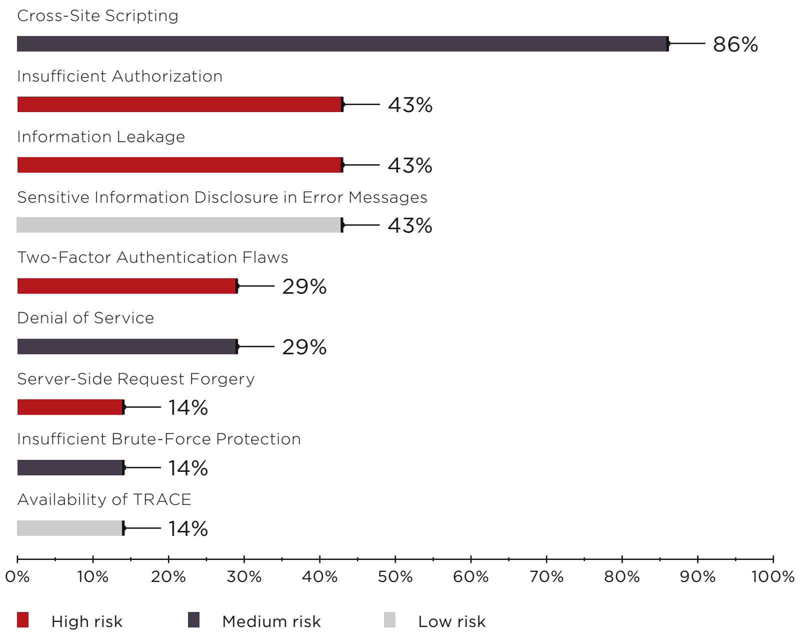


Figure 20. Most common vulnerabilities in server-side components (percentage of systems)

Information leaks are another widespread problem with server-side components, with potentially serious consequences. For instance, when we started a chat in one of the tested applications, we saw the full name and phone number of the other person in the server response. Another example of critical data disclosure is the session ID in the link to a document handled in the mobile application. If the attacker convinces the user to send a link to this document, and the link contains the session ID, the attacker can impersonate the user.

If the mobile application server accepts numeric input (for example, map coordinates), restrictions must be in place. Without restrictions, the attacker can indicate arbitrary coordinates to search for an object on the map. Invalid coordinates will cause a large delay in server response and, as a result, denial of service. Disruption of app operation is harmful to the reputation of the developer.

**29%**

**of server-side components** contain vulnerabilities that can cause disruption of app operation

## Mobile application threats

Almost all applications we studied were at risk of being accessed by hackers. In the client-side vulnerabilities section, we pointed out that the most common issue with mobile applications was insecure data storage. So how can information end up in hackers' hands? The most common scenario is malware infection. The chances of infection increase exponentially on devices with administrator privileges (root or jailbreak). But malware can escalate privileges on its own, too. For instance, ZNIU spyware does so by exploiting the infamous Dirty COW vulnerability (CVE-2016-5195). Once on the victim's device, malware can request permission to access user data, and after access is granted, send data to the attackers. Experts from TheBestVPN have studied 81 VPN applications from Google Play and found that many of them request questionable permissions.

## Recommendations for users

Be careful when apps request overly broad access to functionality or data. If the requested permissions seem unreasonable for the application's intended purpose, do not grant them

A smartphone can be easily lost or stolen. Even though mobile operating systems require setting a password by default, some users choose not to have one. In this case, an attacker with physical access to the device can plug it in to a computer and use special utilities to extract sensitive data from device memory. For example, if backup creation is switched on in Android, application data can be extracted from a backup using Android Debug Bridge (ADB). With root privileges, data can be extracted even when backups are disabled. On jailbroken Apple devices, users often do not change the default SSH credentials (root:alpine). An attacker can then copy application data to a computer via SSH. This threat is especially relevant for corporate phones or tablets used by multiple employees who know the device password.

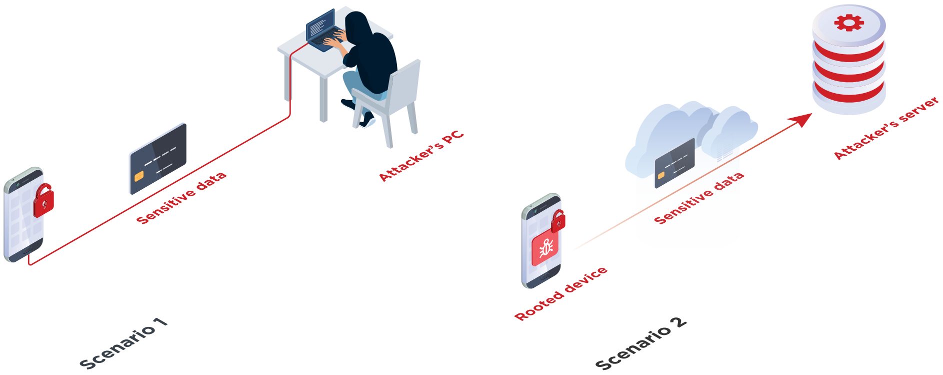


Figure 21. Possible scenarios for theft of user data from mobile applications

**18%**

**of applications**
do not restrict the number of authentication attempts

Sometimes a mobile application can be hacked without any malware or hacking utilities. For instance, the application may have no restriction on the number of attempts to enter the PIN code, or this restriction is set only on the client side and the count is reset when the application restarts. In both cases, an attacker can make an unlimited number of password entry attempts.

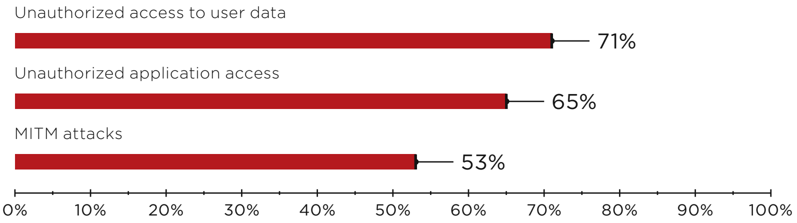


Figure 22. Threats to client-side components (percentage of systems)

## Recommendations for users

Your PIN code must be truly random. Do not use your date of birth, phone number, or ID number. Use biometric authentication (fingerprint, voice, or face) if your device supports it

## Recommendations for developers

Limits on authentication attempts must be implemented both on the server side and on the client side

Server-side vulnerabilities can enable attacks on users. Cross-Site Scripting, the most common web vulnerability, was found in 86 percent of server-side components. Attackers can use it to steal victim credentials, such as cookies, with the help of malicious scripts. This vulnerability can threaten mobile applications if they use components supporting HTML and JavaScript. For example, WebView is a system component allowing Android applications to show web content directly in an application.* iOS has similar components called UIWebView and WKWebView.

* In early 2019, our experts found that WebView contained a vulnerability (CVE-2019-5765) allowing access to Android user data through a malicious application or an Android instant app.

Injection of mail headers or HTML tags is useful for phishing attacks. By injecting mail headers, an attacker can send emails to application users posing as any employee of the company that owns the mobile app.

## Recommendations for users

Stay vigilant when going through your inbox. Carefully check links before opening them, even if you are a client of the company that sent the email. If the linked address contains any misspellings, the email is not genuine. Remember that bank employees never ask for full card information

## Recommendations for developers

Filter user-entered data on the server side. Use HTML coding for special characters

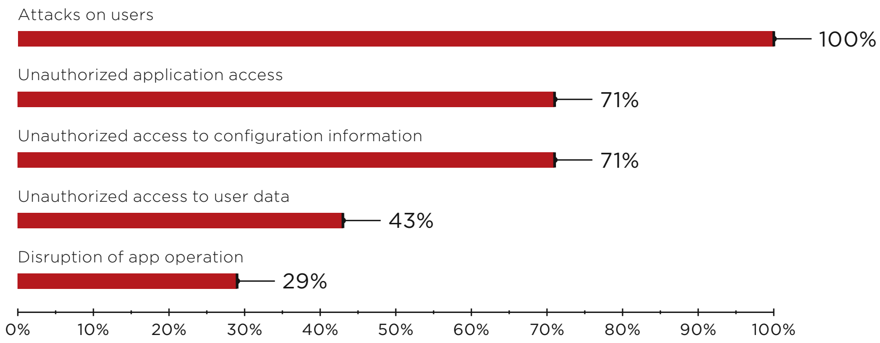


Figure 23. Top five threats for server-side components (percentage of systems)

**18%**

**of applications**
contain session hijacking vulnerabilities

Frequently, threats are caused by a combination of faults in the client side and the server. Imagine, for instance, that when the user exits the application, the session ID is not deleted on the client side and is instead sent to the server with every new request, including during re-authentication. The server, in turn, does not check session timeout, and after authentication it reactivates the old session ID. In this case, any attacker who knows the session ID can impersonate the user.

## Recommendations for developers

Session lifetime must be limited. The session ID must be deleted both on the client side and on the server side. The server must create a new session for the user every time authentication is required

**50% Android**
**22% iOS**

Percentages of applications with insecure data transfer

Communication between the client and the server can also be vulnerable. If the client side communicates with the server using insecure HTTP, an attacker can intercept sensitive data. In the case of a mobile bank, for instance, all payment information is jeopardized. To prevent interception, use the secure HTTPS protocol. This makes the connection secure because all data is encrypted. The device stores certificates. These special files tell the client the name of the server it is supposed to send data to.

But even with HTTPS, client–server communication is not always secure. On the device, the certificates are kept in a store used by all applications. Malware can install an attacker's root certificate on the victim's smartphone—in which case all certificates verified with the fake root certificate will be considered trusted. So if the victim connects to attacker-controlled network equipment (a Wi-Fi router, for example), the attacker can perform a man-in-the-middle (MITM) attack. By sitting in the middle of the connection and listening to traffic, the attacker compromises all data that is transferred.

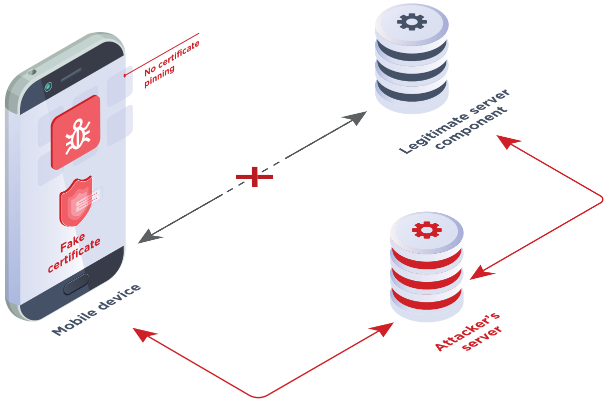


Figure 24. Man-in-the-middle attack (if certificate pinning is not used)

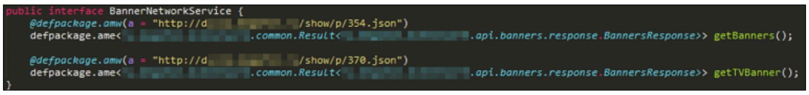
## Recommendations for developers

For maximum security of client-server communication, we recommend using certificate pinning. With this approach, the certificate is embedded directly in the code of the mobile application. As a result, the application becomes independent of the OS certificate store. This prevents MITM attacks

**18%**

**of mobile applications**
contain insecure external links

Beside client-server communication, an app can also contain links for sending data externally via insecure HTTP. Insecure data transfer is more common on Android. Starting with version 9, iOS has provided App Transport Security, which prohibits insecure data transfer by default. However, an developer can expressly list exceptions in the form of addresses with which insecure communication is still allowed. This might be useful during application debugging, but insecure links often end up in the final versions used by the public.



```
public interface BannerNetworkService {
    @defpackage.amw(a = "http://d[redacted]/show/p/354.json")
    defpackage.ame<[redacted].common.Result<[redacted].api.banners.response.BannersResponse>> getBanners();

    @defpackage.amw(a = "http://d[redacted]/show/p/370.json")
    defpackage.ame<[redacted].common.Result<[redacted].api.banners.response.BannersResponse>> getTVBanner();
}
```

Figure 25. Insecure links found in the source code of a mobile application

## Risks for users

Our study indicates that all mobile applications are vulnerable. In a handful of cases exploiting vulnerabilities might require physical access to the device, but usually this can be accomplished remotely via the Internet. Every tested mobile application contained at least one vulnerability that could be exploited remotely using malware.

Sometimes the hacker needs full access to the file system: jailbreak on iOS or root privileges on Android. But even that is not always a challenge. Many mobile device owners escalate their privileges in the OS on purpose when trying to bypass various restrictions, sideload software, or customize the user interface. According to researchers' data, 8 percent of iOS users have jailbroken their devices and 27 percent of Android devices are running with root privileges. Devices with such privileges are at greater risk, because these privileges can be abused by malware. For instance, KeyRaider malware spread through an app distribution platforms for jailbroken devices and stole credentials, certificates, and encryption keys from 225,000 iOS users.

## Recommendations for users

Do not escalate privileges. Rooting or jailbreaking a device opens up access to the device file system and disables protection mechanisms

Because of the scale of the malware problem, Google and Apple are taking active measures to combat cybercriminals. For protection from hackers, Google offers Google Play Protect to scan applications on Android devices and Google Play itself. To prevent distribution of malware through the Apple App Store, Apple performs manual analysis of developer apps before making them available for download.

This analysis helps to reduce the number of malicious applications, but cannot catch all of them. Malware can come even from official app stores. Hackers managed to upload 39 malicious programs to the App Store using XcodeGhost, a fake version of the legitimate Xcode development environment used to create applications for Apple devices. Another example is Anubis, a banking Trojan that successfully evaded security checks by both Google Play and the Android security system.

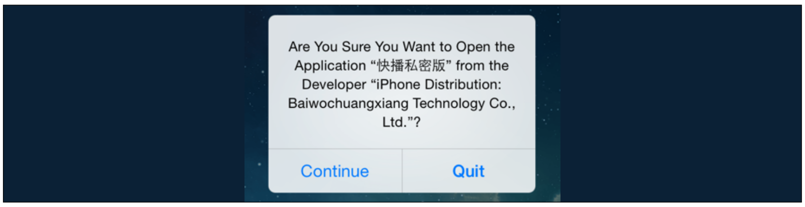
## Recommendations for users

Update your OS and applications regularly. If you have rooted or jailbroken your device, remember that it may not update automatically

Official app stores are just one way for malware to infect a device. Even a brand-new smartphone can contain malicious code. For instance, a developer attack resulted in spyware being pre-installed on Alcatel smartphones. User devices were compromised even before they had been started for the first time. Another example is the TimpDoor backdoor, which hackers distributed by sending a link to victims using SMS.

To prevent attacks, iOS prohibits downloading software from sources other than the App Store. But there are ways to work around this restriction. These include use of the user's own certificates and Mobile Device Management (MDM). To do that, the user must manually confirm that the application developer's certificate is trusted and allow downloading and installing the app from an untrusted source. In a phishing attack, hackers may succeed in convincing the user to perform these steps.

Apple prohibits App Store applications from using private APIs. These APIs contain methods that could be used to download other apps and perform other actions. A Trojan could use private APIs to install other, non-App Store software on the victim's device, therefore bypassing any security checks by Apple. However, Apple's checks themselves are not perfect, judging by distribution of malware such as YiSpecter. The technique used by the YiSpecter attackers was very simple. A user opened an infected link, confirmed installation of software from outside of the App Store, and the device became infected. Once on the victim's device, YiSpecter used private APIs and automatically downloaded other programs to steal personal data.



Are You Sure You Want to Open the Application “快播私密版” from the Developer “iPhone Distribution: Baiwochuangxiang Technology Co., Ltd.”?

Figure 26. Request to confirm installation of third-party software
(source: zdnet.de/88248255/ios-malware-yispecter-auch-fuer-geraete-ohne-jailbreak-gefaehrlich/)

## Recommendations for users

Do not open links received from unknown senders in SMS messages and chats. Even if you know the person suggesting an application, remain vigilant. Never confirm requests for installation of third-party software on your smartphone

One of the alternate ways of installing malware on Apple devices is downloading application files (.ipa) to the victim's computer and installing them with the help of the victim's Apple ID (iCloud account) and application installer (such as Cydia Impactor) via a USB connection. This malware can be distributed on unofficial stores as free ("cracked") versions of App Store software. This is how devices were infected with WireLurker.

## Recommendations for users

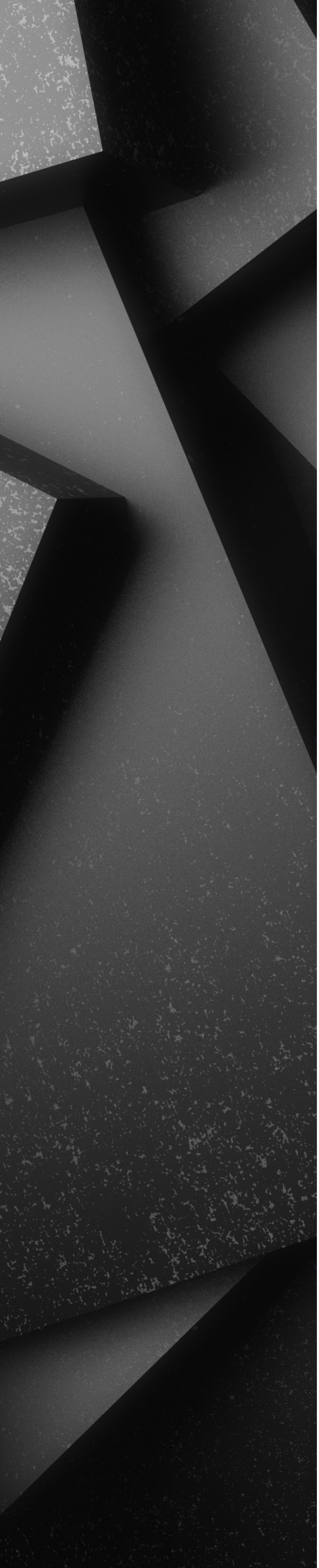
Do not connect your device to untrusted PCs or charging stations. Modern mobile OS versions ask the user to confirm trust. Never confirm trust if you are unsure about the security of the computer to which you are connecting your device

Google's policy regarding downloading apps from alternate sources is less stringent. During OS setup, the user decides whether to allow downloading software from unofficial sources. According to statistics, every fifth Android device allows installation of applications from third-party sources. In addition, 7 percent of Apple devices and 3 percent of Android devices have at least one application installed from unofficial stores. Remember that administrator privileges, as mentioned already, remove any iOS or Android restrictions on software downloading.

## Recommendations for users

Do not trust third-party mobile app stores. Suspicious software (such as allegedly "cracked" free versions of commercial applications) can contain malicious code

Security depends on users. Device owners must take responsibility for protecting the data they store in mobile applications. But user precautions will still fall short if developers leave vulnerabilities in their applications. Unfortunately, not all developers of mobile software have risen to the occasion.



# Conclusions

Hackers love targeting mobile devices, which are rich with personal data and payment card information. Our results indicate that developers of mobile applications often neglect security, with the main issue being insecure data storage. User information stored in cleartext, unmasked data in screenshots, and keys and passwords in source code are just a few of the flaws that offer opportunities to cyberattackers.

Users themselves may unwittingly help to compromise their devices by expanding smartphone capabilities, disabling protection, opening suspicious links in SMS messages, and downloading software from unofficial sources. Securing user data requires a responsible attitude on the part of both application developers and device owners.

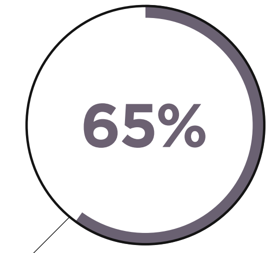
Nor can we underestimate the role of server vulnerabilities. Protection of mobile application servers is no better than that of clients. In 2018, every tested server-side component contained at least one vulnerability enabling various attacks on users, including impersonation of the developer in phishing emails, placing the developer's reputation at risk. To prevent exploitation of server vulnerabilities, we recommend using a web application firewall (WAF).

Beyond client and server vulnerabilities, risks also include client–server communication. Data sent over an insecure protocol can be completely compromised. But even secure connections are not always safe. Developers still have yet to attain a deep understanding of the importance of security.

Protection mechanisms are the weak spot in mobile applications. Most of the discovered vulnerabilities were introduced during the design stage and result from failure to "think through" security-related questions. We recommend a methodical approach to designing and following through on mobile application security, regularly testing it starting from Day 1 of the software lifecycle. The most effective method is white-box testing, in which security analysts have full access to source code.

## About the research

This report includes data from comprehensive security assessments of 17 fully functional mobile applications tested in 2018. It does not include applications whose owners did not provide their consent to using results of security assessment for research purposes, and applications for which we analyzed only some functionality.



Percentage
of applications covered
with white-box testing

8 client-side components tested Android

9 client-side components tested iOS

7 server-side components tested

The security level of each application was assessed using black-, gray-, or white-box methods with the assistance of automated tools. Black-box testing means looking at an information system from the perspective of an external attacker who has no prior or inside knowledge of the application. Gray-box testing is similar to black-box testing, except that the attacker is defined as a user who has some privileges in the application. White-box testing includes use of all relevant information about the application, including source code.

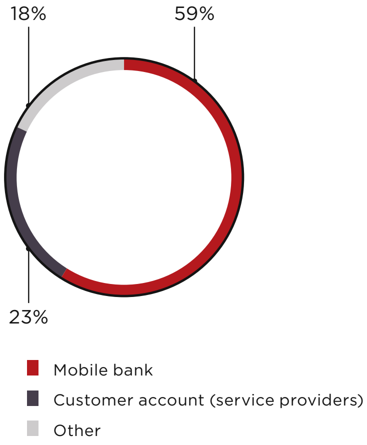


Figure 27. Types of applications

This document describes vulnerabilities in client-side and server-side components. In addition, we reviewed mobile application threats, including those caused by client–server communication. The report describes only vulnerabilities related to faults in application code and configuration. Other common information security issues (such as flaws in software update management) have not been considered here. Code vulnerabilities were split into two groups:

- Vulnerabilities in mobile application code (made by programmers during development)
- Errors in implementation of security mechanisms (made during the design stage)

The risk level of vulnerabilities was assessed based on the impact of the potential attack on user data and the application itself, taking feasibility into account. We made a qualitative assessment of vulnerabilities as high-, medium-, or low-risk.

### About Positive Technologies

Positive Technologies is a leading global provider of enterprise security solutions for vulnerability and compliance management, incident and threat analysis, and application protection. Commitment to clients and research has earned Positive Technologies a reputation as one of the foremost authorities on Industrial Control System, Banking, Telecom, Web Application, and ERP security, supported by recognition from the analyst community. Learn more about Positive Technologies at ptsecurity.com.

ptsecurity.com
info@ptsecurity.com