Cybersecurity

RESEARCH

Open Access

Understanding security risks in mobile-to-PC screen mirroring: an empirical study

Zhaoyu Qiu[1,2], Shishuai Yang[2*], Yifan Yu[2], Yujia Luo[2] and Wenrui Diao[2*]

Abstract

To facilitate collaboration across multiple devices and benefit from larger screens and better user experiences, many users choose to mirror their screen content of smartphones to personal computers. The implementation of the Android screen mirroring feature varies across different manufacturers, resulting in significant security differences among screen mirroring apps. Moreover, actual incidents of screen content leakage have exacerbated users' concerns about the security of the Android screen mirroring feature. In this work, we systematically analyzed the system architecture of the Android screen mirroring feature and the security risks it faces. Specifically, we identified four critical security risks in the communication process between the mobile and PC sides of screen mirroring apps, including arbitrary access to screen content, MITM (Man-in-the-Middle) attacks, malicious commands injection, and data sniffing attacks. Attackers can exploit these identified security risks to arbitrarily access screen content or manipulate user's phone to perform malicious operations. To evaluate the security risks of the Android mirroring feature in real-world deployments, we conducted a security evaluation on over 20 popular screen mirroring apps from multiple sources. The results indicate that all of these apps are facing at least one of the aforementioned security risks. Finally, we provide the corresponding recommendations to mitigate the identified security risks.

Keywords Screen mirroring feature, Android app security, Security analysis

Introduction

In the field of mobile operating systems, Android has gained a large number of users and developers because of its openness, rich apps, and extensive device support. As of February 2024, Android held a 71.43% market share in the global mobile operating system market, firmly sitting at the top of the industry (Mobile Operating System Market Share Worldwide 2024). As the smartphone plays an increasingly important role in people's daily lives, it has become an indispensable digital assistant. To collaborate across multiple devices, more and more people are choosing to mirror screens of their smartphones to PCs (Personal Computers) in real-time, as shown in Figure 1. The screen mirroring apps allow users to use a larger screen, making it more efficient to handle information and tasks on their PCs. At the same time, they can receive real-time information from their smartphones, such as instant messaging, social media updates, and enjoy a more immersive and seamless user experience.

Although the screen mirroring feature provides convenience to users, it also presents numerous security risks in the real world, leading to many security incidents. For instance, attackers could exploit this feature to obtain passwords and account information from smartphones, allowing them to steal user's property (Don't Become the Victim of Screen Sharing Scams 2024). Moreover, the lack of a standardized approach to screen mirroring implementation, with different software vendors offering various solutions, further

*Correspondence:
Shishuai Yang
shishuai@mail.sdu.edu.cn
Wenrui Diao
diaowenrui@link.cuhk.edu.hk
[1] Faculty of Electronic and Information Engineering, Xi'an Jiaotong University, Xi'an, China
[2] School of Cyber Science and Technology, Shandong University, Qingdao, China

Springer Open

© The Author(s) 2025. **Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

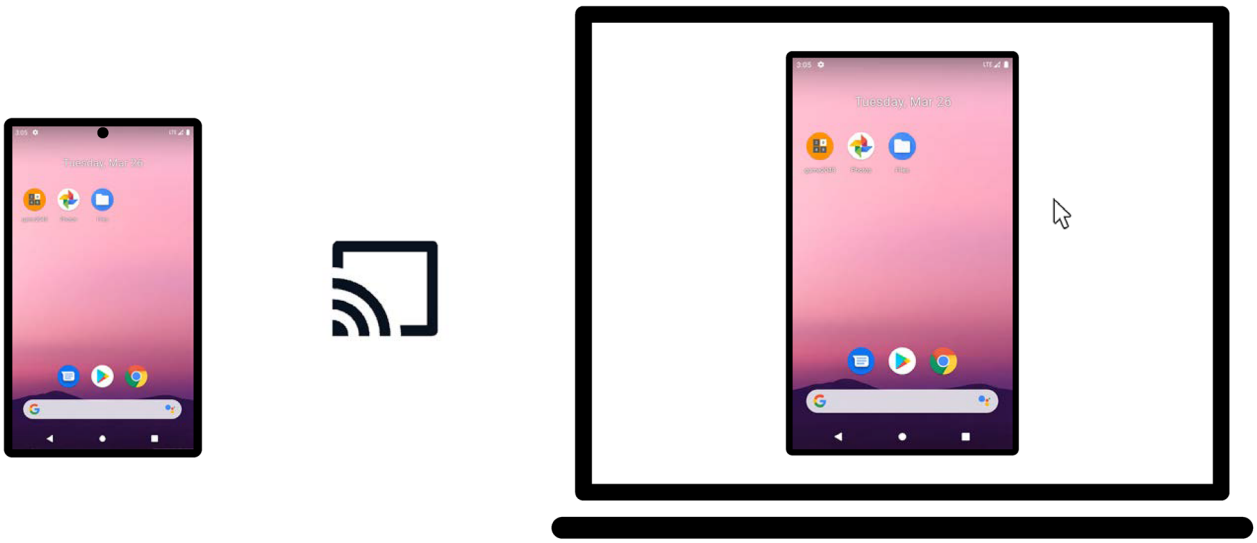

Fig. 1 Screen mirroring

heightens the potential for security risks. To mitigate the security threats posed by the screen mirroring feature, starting from Android 10, Google has prohibited privileged apps from being granted video capture permissions (Restricted Screen Reading 2024), such as `READ_FRAME_BUFFER` and `CAPTURE_VIDEO_OUTPUT`. In addition, screen content cannot be captured without user's consent. However, due to weak security awareness among users, they may inadvertently grant screen capture permissions, leading to the leakage of sensitive data. Furthermore, some screen mirroring apps also support users to control their smartphones from PCs, which could result in security risks such as malicious commands injection.

To the best of our knowledge, there have been very few studies focusing on the security of the Android screen mirroring features. The most relevant work was conducted by Tian et al. (2022), who systematically evaluated the security of casting multimedia files from phones to smart TVs based on the `DLNA` protocol, identifying a series of critical security issues. Unlike this study, we systematically analyzed the security risks associated with the Android screen mirroring feature, including the underlying communication protocols, authentication mechanisms, and the functionality of the PC remote control of the smartphone.

Our Work In this study, we systematically analyze the architectural design of the Android screen mirroring feature and the security risks it faces. In general, we identified four significant security risks (**SR** for short): (1) ***SR1: arbitrary access to screen content.*** (2) ***SR2: MITM attack for screen content theft.*** (3) ***SR3: malicious commands injection attack.*** (4) ***SR4: data sniffing attack.*** By exploiting these security vulnerabilities, attackers can easily steal sensitive information from user's screen or remotely inject commands to control user's smartphone for malicious purposes.

To measure the impact of the identified security risks, we collected over 20 popular screen mirroring apps and conducted a multi-dimensional security analysis using both static and dynamic methods. By statically analyzing APK files, hooking key functions, monitoring communication traffic, and dynamically tracking resource usage, we identified security risks and generated exploit scripts to validate these vulnerabilities. The results show that all apps are exposed to one of the above security risks, especially for **SR2**, where 71.43% of the apps are at risk of MITM attacks.

Responsible Disclosure We have reported discovered vulnerabilities to corresponding software vendors and `CNVD`. Six reports have been confirmed until now, one vulnerability has been confirmed by `vivoSRC` (vivoSRC 2024) (rated as **high** severity), and the other five vulnerabilities have been confirmed by `CNVD`: `CNVD-2024-25211` (rated as medium severity), `CNVD-2024-25640` (rated as medium severity), `CNVD-2024-26729` (rated as medium severity), `CNVD-2024-25639` (rated as low severity), and `CNVD-2024-25676` (rated as low severity).

Contributions The main contributions of this paper are:

- **Systematic Analysis.** We systematically analyzed and explored the architecture design of the Android screen mirroring feature, as well as the actual security risks during deployment. This study

can enhance the security of the Android screen mirroring feature.

- **New Security Risks.** By exploiting the four identified security risks, attackers can capture the current screen contents of users' smartphones and steal private and confidential data. More seriously, the attacker can inject commands to control the user's smartphone.
- **Real-World Measurements and Discussions.** We conducted a real-world measurement study on over 20 popular screen mirroring apps, and the results show that all apps have at least one security risk. Furthermore, we proposed mitigation measures for the identified security risks.

Background

This section will provide the necessary background knowledge about the Android screen mirroring feature.

Screen capture & remote control

Screen Capture Android offers various features for capturing screen content, such as screenshots and screen recording. For example, users can take a screenshot of the current screen by simultaneously pressing the power button and volume down button. If other devices want to capture the current screen content in real-time, they need to use APIs provided by the Android OS. For example, APIs in the `android.media.projection` package (Media Projection 2024) can be used to capture the current screen content and transmit the data to other devices. In this study, almost all screen mirroring apps use this method to capture the screen content.

Remote Control The Android mirroring feature often provides the ability to remotely control the phone from PC, and it is mainly implemented in the following two ways:

- Using Java reflection to access the non-SDK APIs (Yang et al. 2022) and execute remote control commands issued from the PC side, such as simulating clicks which require methods provided by the `InputManager` class. Implementing remote control through non-SDK APIs requires higher privileges and is only applicable in the following two scenarios: (1) using adb (Androd Debug Bridge) connection, and (2) the screen mirroring apps are system apps.
- Accessibility service is a special feature designed to assist users with disabilities or those temporarily unable to fully interact with their devices (Create Your Own Accessibility Service 2024). If users enable accessibility permission for screen mirroring apps, they can perform actions on behalf of users using accessibility service APIs.

Classification and workflow of screen mirroring

We categorize the existing implementations of the Android mirroring feature into the following four architectures (C1 to C4, short for Category 1 to Category 4) based on their availability on mobile and PC sides, as well as their connection methods. In addition, some screen mirroring apps may support one or more of the following architectures.

C1: Single-side HTTP Scheme The screen mirroring app is only available on the mobile side. It generates an HTTP URL on the smartphone via the screen mirroring app. Users can access this URL through the browser on the PC within the same LAN to view the phone's screen content.

When users launch the screen mirroring app on their smartphones, the embedded server within the app will start. Upon granting permission to read the screen content, the app will generate an HTTP URL for users to access. By entering this URL in the PC browser, users can view the phone's current screen content. Depending on the screen mirroring app, users may need to enter a PIN code or password during the URL access. If authentication succeeds, the connection proceeds. Otherwise, it will be terminated. Once these steps are completed, the screen mirroring app on the smartphone transmits the screen content to the PC browser, allowing user to view the screen on the PC.

C2: Single-side USB Scheme The screen mirroring app is only available for the PC side. Users need to connect the smartphone to the PC using a USB data cable and enable the USB debugging option on the smartphone. Afterward, users can view the phone's screen content through the screen mirroring app on the PC.

Screen mirroring apps using the Single-side USB Scheme are typically built based on the `Scrcpy` (Scrcpy 2024; Scrcpy - Best Android Screen Mirroring App 2024). When the PC side of the screen mirroring app is launched, it will push `scrcpy-server.jar` to the smartphone and execute it through adb commands. The `scrcpy-server.jar` provides functions such as capturing the smartphone's screen content and injecting touch events. It creates a UNIX abstract socket based on the command-line arguments and communicates with the PC app through TCP connections.

C3: Dual-side LAN Scheme For these screen mirroring apps, users have to install both the mobile and PC versions, and ensure they are connected to the same LAN. After launching both of them, the apps will automatically discover each other. After confirmation, the screen content can be mirrored to the PC.

The workflow of such screen mirroring apps is as follows:

- *Device Discovery.* The primary purpose of device discovery is to obtain the device name of the smartphone/PC and the port number for subsequent communications. Screen mirroring apps primarily use the following two methods for device discovery:

  (1) `mDNS` broadcasting. The mobile app or PC app sends a broadcast message from source port 5353 to destination port 5353, with the destination address 224.0.0.251 or ff02::fb. This message contains a string formatted as `DeviceName._ServiceType._Protocol.local`. The `DeviceName` field represents the device's name, the `ServiceType` field represents the service type name, which is used by apps to identify each other, and the `Protocol` field is usually set to `tcp`. When the screen mirroring app in the LAN receives the broadcasting message, it will send a response message containing the service port number if it provides the matching service. Subsequent communication will take place on this port.

  (2) `UDP` broadcasting. The mobile app sends a broadcast with a fixed destination `UDP` port in the LAN, with the destination address being either local broadcast 255.255.255.255 or directed broadcast (e.g., 192.168.1.255). After receiving the broadcast, the PC side replies with a `UDP` message containing the device name. The reverse process also applies.

- *Connection Confirmation.* Once the device discovery phase is completed, the user can select the device and send a connection request from either the smartphone or PC, and the other one needs to confirm the connection. In addition, the user needs to grant permission to capture the screen on the smartphone. If authorization is successful, the mobile app will prepare to capture the screen content.
- *Connection Establishment.* The protocols used during the connection establishment phase may vary depending on whether the mobile app or the PC app serves as the server.

  (1) Mobile app as the server. This typically involves communication via the `WebSocket` protocol. During the device discovery phase, the PC app obtains the `WebSocket` address and port of the mobile app and it can access the corresponding URL to retrieve screen and audio data.

  (2) PC app as the server. The PC app can also serve as one server. The ports of the PC app used for transmitting video and audio can be either fixed or dynamically assigned. If the ports are dynamically assigned, the mobile and PC app typically use the `TCP` protocol for data transmission. Otherwise, if the ports are fixed, the mobile and PC app will transmit data using either `TCP` protocol or `WebSocket` protocol.

C4: Dual-side Internet Scheme This scheme also needs a mobile app as well as a PC app, but they only need to be connected to the internet. After launching the apps, a connection request can be initiated from either the mobile or the PC side. Once the user confirms, the phone's screen content can be mirrored to the PC. Screen mirroring apps based on this scheme typically involve communication between the smartphone, the PC, and the software vendor's server. In addition, these apps usually offer remote control functionality.

To get started, the user must connect both the mobile and the PC app to the internet and log in to the same account. Once logged in, the app will query and display the current account's online devices. Then, the user can select the mobile device on the PC and send a request to the server, which forwards the request to the target. After the user confirms on the mobile device, a connection is established. Upon confirmation, the mobile device will transmit its screen and audio data to the PC through the server. For remote control features, control commands from the PC are also transmitted to the mobile device through the server. All communication processes use encrypted channels, ensuring secure data transmission.

Threat model

In this study, we assume that the attacker and the target device are on the same LAN. Unlike traditional threat models in LAN security research (Alrawi et al. 2019), this study not only considers attackers with strong capabilities, such as independent network devices on the same LAN, but also focuses on less capable attackers, such as Android or PC malware with only network access permissions. Depending on the actual attack scenarios, this study categorizes adversaries with different capabilities into the following types:

- *Android Malware* For malware installed on user's Android smartphone, it only needs network access permissions without declaring other sensitive permissions. Some existing security research has

adopted similar assumptions (Alrawi et al. 2019; Lee et al. 2017). Without additional permissions, the malware cannot directly steal the user's screen content or control the user's smartphone. Furthermore, due to the sandbox mechanism of the Android OS, the malware also cannot access screen mirroring apps' data.

- *PC Malware* Similarly, malware installed on user's PC only needs network access capability. In real attack scenarios, we do not consider stealing data from specific PC screen mirroring app because it usually doesn't store user's private data. Instead, we consider exploiting vulnerabilities in mobile apps to launch attacks.
- *Network Device* For the network device, the attacker can not only send network requests but also sniff traffic on the same LAN. This means the attacker can obtain the Wi-Fi access point password or compromise the related authentication mechanisms. In this scenario, the attacker's capabilities are relatively strong.

Flaws and attacks of screen mirroring

Security risks We collect popular Android screen mirroring apps from multiple sources and categorize them based on their features to systematically analyze the security of the Android mirror app feature. The most exposed aspect of the screen mirroring feature is the screen content of users' smartphone, which may include sensitive data such as accounts, passwords, and verification codes. In addition, some apps allow PC to control the smartphone remotely, which is also vulnerable. If attackers can perform actions on behalf of users, they could install malicious apps, grant permissions, and transfer funds to target accounts. Both of these security risks pose significant threats to users.

Security Guarantees The interaction of screen mirroring apps between smartphone and PC is very important, and the establishment of a secure channel requires several key mechanisms, such as authentication, data encryption, and the constraints and checks of connection channel. The lack of such mechanisms during channel establishment will lead to the security risks listed in Table 1.

- *Authentication.* Two communicating entities must securely and efficiently authenticate each other before exchanging data. However, most mirror apps either do not have authentication mechanisms or lack robust authentication mechanisms. Weak or missing authentication can lead to **SR1**, **SR2**, and **SR3**.
- *Data Encryption.* Once both communicating entities have been authenticated, the subsequent communication data should be encrypted before transmission. The lack of encryption measures could result in **SR4**.
- *Constraints and Checks of Connection Channel.* The typical design of a screen mirroring app is to mirror the screen of a smartphone onto a computer. If screen mirroring apps support one-to-many communication, they must ensure that users can sense and cancel connections. In this study, we found that some screen mirroring apps allow one smartphone to connect to multiple PCs without verifying the identity of connected devices. This provides an attack vector for attackers to steal screen content or remotely inject commands to maliciously operate the device. Lack of constraints and checks of connection channels could result in **SR1** and **SR3**.

Workflow To identify security risks in the communication process of the Android mirroring feature, we primarily followed the steps outlined below, as shown in Fig. 2.

- *Application Collection.* Firstly, we try to collect screen mirroring apps from Github, Google Play and third-party markets (`Vivo`, `Xiaomi`, and `APKPure`), due to the limited number of screen mirroring apps and significant overlap among different sources, we finally selected 21 of the most widely used screen mirroring apps for our analysis. We need to specifically analyze the functions that implement the screen mirroring functionality within the apps to reconstruct the communication process, but these functions are hidden in packed apps. Therefore, we excluded the majority of packed apps to facilitate the subsequent security analysis.

Table 1 Security risks of screen mirroring

No.	Security Risk	Possible Threats and Consequences
SR1	Arbitrary access to screen content	Screen content leakage
SR2	MITM attack for screen content theft	Screen content leakage
SR3	Malicious commands injection attack	Attacker operates user's smartphone maliciously
SR4	Data sniffing attack	Screen content leakage, authentication bypassed

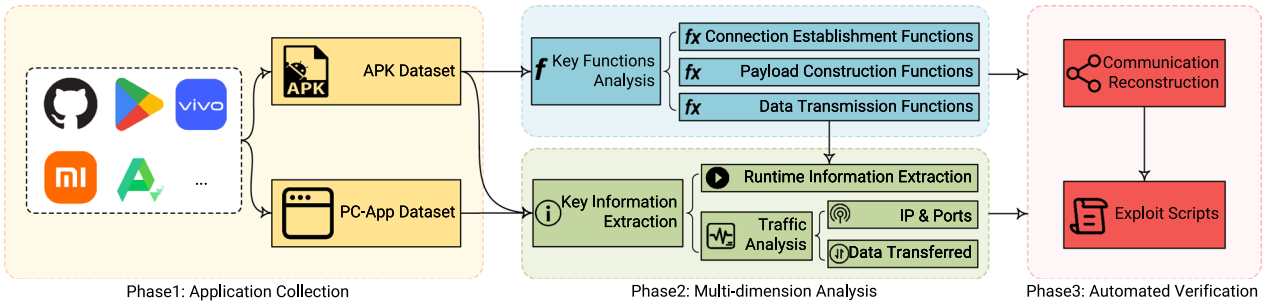

Fig. 2 Workflow of security analysis

- *Multi-dimension Analysis.* To identify security risks in screen mirroring apps, we conducted a multi-dimensional security analysis using both static and dynamic methods.

  (1) Key Functions Analysis. After obtaining the mobile screen mirroring apps, we first used `JADX` (JADX 2024) for static analysis. Due to the variety of screen mirroring schemes and different implementation of software vendors, we analyzed the decompiled code manually. We used clues such as keywords on the app's UI, the use of key resource files, and network related APIs to locate the communication logic of the app. Then, we extracted the connection establishment functions, payload construction functions, and data transmission functions as the key functions for further analysis. We focused on these functions to obtain details such as the format of the data transmission in the communication process, the meaning of different packets, and details of the communication process.

  (2) Key Information Extraction. To obtain more details about the communication process, we used `Frida` (Frida 2024) to hook the extracted key functions to obtain the runtime parameters of these functions and the complete details of the data transmission in the communication process. This step can facilitate the subsequent communication simulation at the automated attack verification phase. In addition, we use traffic analysis tools (`Wireshark` (Wireshark 2024) for PC, `Tcpdump` (TCPDUMP & LIBPCAP 2024) and `Mitmproxy` (mitmproxy 2024) for Android OS) to assist in constructing the communication process. The previously mentioned approaches focused on mobile apps, and we used traffic information to analyze PC apps and record their requests or replies, which provides references for subsequent exploit script construction. Another purpose for traffic analysis is to obtain the IP address and TCP/UDP port numbers during the communication process. Combining the key functions analysis and traffic analysis, we can also get more information on software vendors' customized implementation, e.g., whether the port number of the video channel is hard-coded or transmitted in communication. During the key information extraction phase, due to the different UI implementations of mobile apps, we manually triggered the relevant functionalities before analyzing them.

- *Automated Verification.* Based on the previous analysis results, we obtained the entire workflow and data transmission details during the communication. We reconstructed the communication process using this knowledge, and carefully analyzed whether the screen mirroring apps adopt complete authentication mechanisms and data protection measures. For mobile apps that lack protection and checking mechanisms during the communication process, we constructed exploit scripts to simulate the complete communication and try to intercept the screen content data or inject malicious control commands. We used these scripts to automate the verification of security risks in screen mirroring apps.

SR1: arbitrary access to screen content

Some screen mirroring apps provide the functionality to mirror the screen of one smartphone to multiple PCs. However, such apps often lack authentication for the connecting parties and do not impose constraints or checks on the number of connections. This means that any connection requests will be accepted, and users cannot sense the established connections. Moreover, users cannot

cancel potential malicious connections, easily leading to screen content leakage.

Attacker's Capabilities In this scenario, attackers only need to have network access to the LAN, such as Android Malware, or PC Malware.

Most screen mirroring apps based on the Single-side HTTP Scheme architecture allow one smartphone to connect with multiple PCs. As we mentioned in Section "Classification and workflow of screen mirroring", this scheme regards the mobile app as server, and other devices can access it via the URL like `http://{ip}:{port}`. However, since the IP of user's smartphone is fixed, and most apps use the default port 8080, so the attacker can easily obtain the IP and port of the screen mirroring apps in the LAN. Even if the screen mirroring apps use a custom port, attackers can determine the correct port number by enumerating all ports. Therefore, attackers can directly access the target URL to get the current screen content of user's smartphone. As shown in Fig. 3, attackers can obtain the screen content of two mobile phones in the LAN. When other PCs access the target URL, the established connection remains uninterrupted. This allows attackers to silently capture the screen content of the user's smartphone without affecting its normal usage.

Some screen mirroring apps using the Single-side HTTP Scheme, like `ScreenStream` (ScreenStream 2024), update the number of connected clients in real-time. However, the information about connected number can only be viewed on the main interface of the screen mirroring apps, which users typically do not notice. Additionally, some apps display a toast notification upon successful client connection but do not show the device name or the number of connected devices.

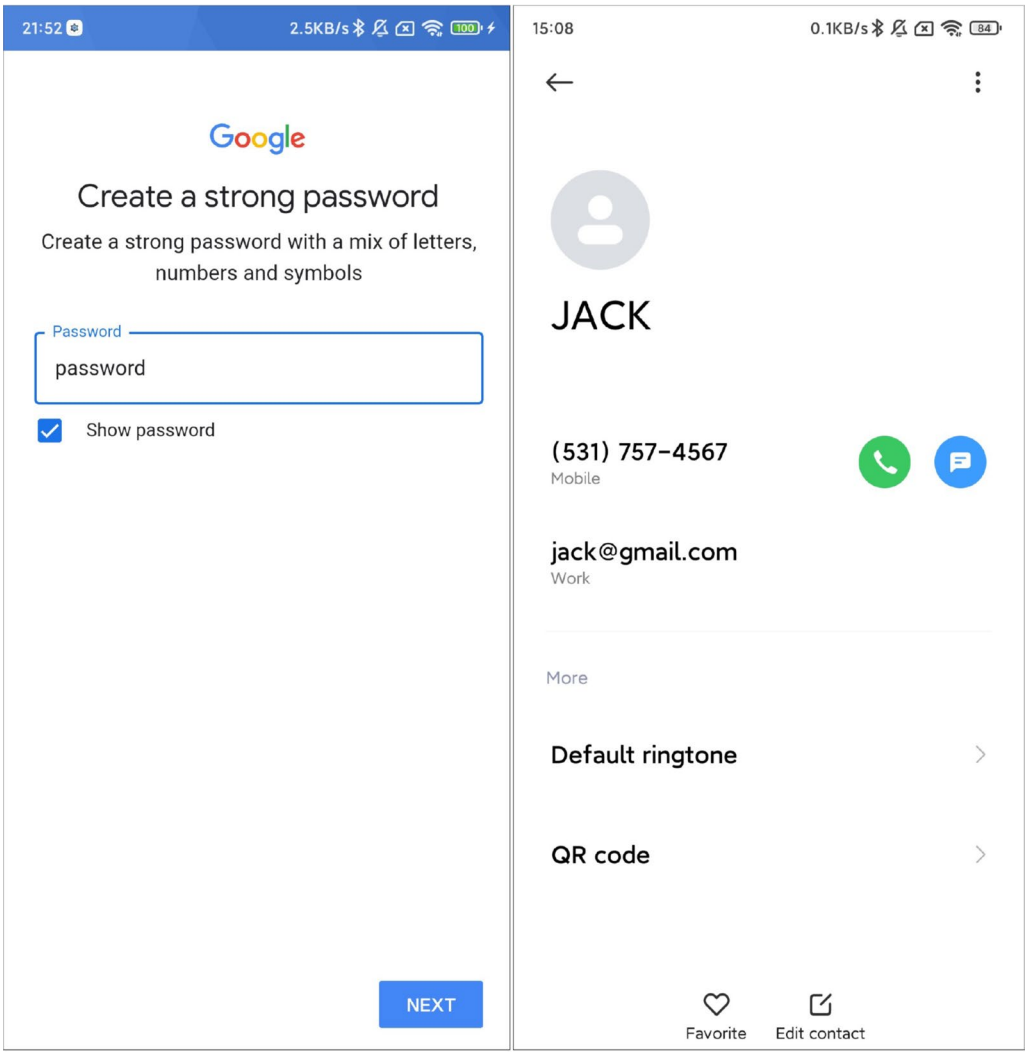

Fig. 3 Arbitrary access to screen content

Furthermore, some apps generate a QR code on the PC browser or display a URL on the mobile app in the format of `https://{AppDomain}/{RandomStr}`, which ultimately redirects to `http://{ip}:{port}`. This encoding method does not effectively mitigate the vulnerability.

SR2: MITM attack for screen content theft

Authentication is essential for ensuring that communication entities are genuine and trustworthy, and protect the screen content from being accessed by disguised attackers. However, we found that most software vendors using Dual-side LAN Scheme and Single-side USB Scheme have not implemented mutual authentication between the mobile and PC sides. This means that attackers can impersonate a legitimate PC to communicate with the smartphone, further intercept the phone's screen content transmission, and forward the data to the benign PC app, namely MITM attack.

Attacker's Capabilities For screen mirroring apps using Dual-side LAN Scheme, attackers can be either Android malware or PC malware. However, for those using Single-side USB Scheme, attackers can only be PC malware, since the wired connection uses the adb command for communication and the mobile app uses randomized sockets, making it difficult for Android malware to attack.

Attack against Dual-side LAN Scheme During the device discovery phase of such apps, attackers can send broadcast or response messages to the mobile phones within the same LAN. Depending on specific device discovery protocol, attackers can choose different attack methods. For the `mDNS` protocol, attackers can obtain the `DeviceName._ServiceType._Protocol.local` string broadcast from the mobile app and extract the `ServiceType` field to construct a forged broadcast. For the `UDP` protocol, since the ports on both sides are fixed, attackers can directly send response messages.

In the connection confirmation phase, the user selects the target PC listed by the mobile app and grants permission. The attacker only needs to wait for user's confirmation. After the connection is established, the malware needs to send specific requests to obtain the screen content. Once the connection is established, the malware can access the

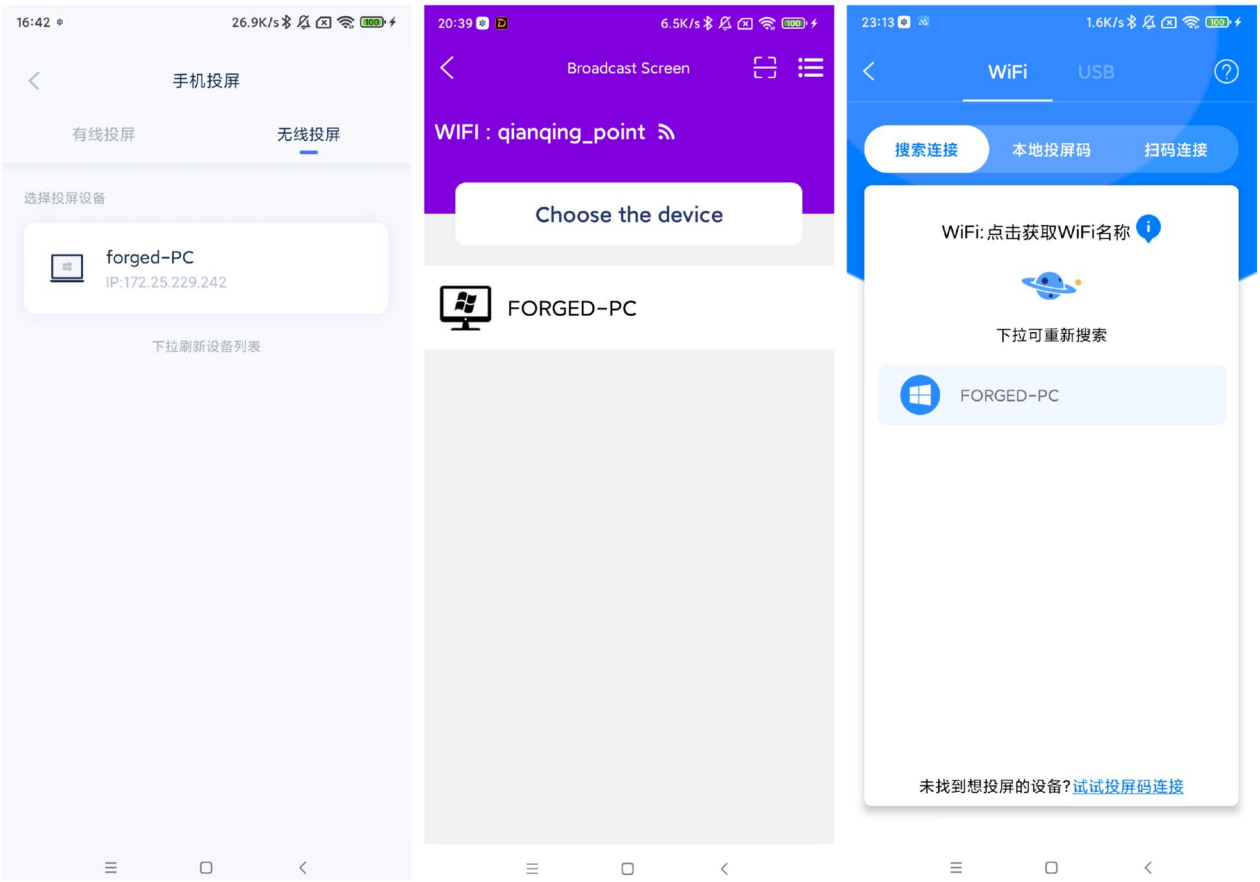

Fig. 4 Malware is identified as legitimate app

phone's screen content and forward the data to the benign PC app. The user can use the functions normally, but the screen content will be completely leaked. Through the methods mentioned above, almost all mobile apps will recognize the fake PC as legitimate PC, as shown in Fig. 4.

Attack against Single-side USB Scheme As we mentioned in Section "Classification and workflow of screen mirroring", the communication of these screen mirroring apps is established between the PC app and the `scrcpy-server`. However, attackers can intercept the communication to obtain the screen content. The optimal time for the attack is when `scrcpy-server` has been pushed to the smartphone and executed, but the PC app has not yet established the connections with it. Within this vulnerability window, PC malware can establish TCP connections with `scrcpy-server`, and once the connections are established, the attacker can access the screen content.

The core idea of this attack is to repeatedly attempt and make the PC malware establish connections with `scrcpy-server` before the benign PC app. Once the connections are established, the PC malware will forward the received data to the benign PC app, allowing the user to continue using the screen mirroring function. However, attackers can access the screen content in real-time, ultimately leading to screen content leakage.

SR3: malicious commands injection attack

For screen mirroring apps that provide remote control features, some of these apps lack sufficient authentication mechanisms as well as constraints and checks on the connection channels. The mobile side of the apps does not verify the identity of the requesting parties and directly execute remote commands issued by them. In addition, some apps allow connections with multiple PCs.

Attacker's Capabilities For apps using Dual-side LAN Scheme, the attacker only needs network access to the LAN, such as Android malware or PC malware. For apps using Single-side USB Scheme, the attacker can only be PC malware, which is the same as we assumed in Section "SR2: MITM attack for screen content theft".

Screen mirroring apps built on Single-side USB Scheme use plaintext transmission for remote control commands. Additionally, these apps do not verify the identity of the connected PC, posing a risk of MITM attacks. Therefore, when `scrcpy-server` connects to the PC malware, attackers can send commands to manipulate the user's smartphone.

For some system apps using the Dual-side LAN Scheme, the mobile and PC apps communicate via IP addresses and fixed port. However, these mobile apps allow connections from multiple PCs and process commands from them. Some of the commands are constructed in `JSON` format, and malware can easily construct and send malicious remote commands to control user's smartphone.

For apps using the Dual-side Internet Scheme, more comprehensive authentication mechanisms and encryption measures are usually implemented. We did not find related security risks.

SR4: data sniffing attack

Except for screen mirroring apps using Dual-side Internet Scheme, other schemes rarely use encryption to protect communication data. They usually transmit data in plaintext, including encoded byte streams of screen content and credentials related to authentication.

Attacker's Capabilities Attackers are network devices within the LAN, possessing strong capabilities. They can not only access the LAN but also sniff and analyze the traffic within the LAN.

Sniffing and Decoding Screen Content Data Since communication data is transmitted in plaintext, attackers can easily obtain relevant data from the network traffic. Although the data is usually encoded, such as video streams encoded using H.264 (Advanced Video Coding 2024) or H.265 (High Efficiency Video Coding 2024) algorithm, these decoding algorithms are publicly available. Attackers can decode the data and ultimately obtain the smartphone's screen content.

Bypassing Authentication Mechanisms Some screen mirroring apps using Single-side HTTP Scheme have added authentication mechanisms that require users to enter a PIN code for verification. If the verification is successful, users are redirected to a URL with a random string parameter. However, they still use plaintext for communication, allowing attackers to obtain the final URL from the network traffic, therefore they can bypass the authentication mechanism and ultimately gain access to the screen content.

Practical attack cases

This section will detail a few cases of practical attacks against the Android screen mirroring feature.

Case study 1: PC device forgery and screen content stealing attack

Screen mirroring apps based on the Dual-side LAN Scheme typically do not authenticate the identity of the PC. Therefore, attackers can create a fake PC app and trick the user to connect with it, thereby conducting a MITM attack to steal the user's screen content.

Attack Setup We selected `JinZhou` app[1] as the target of this attack. Assume the attacker is a PC malware

[1] package name: `com.jx.jzscreenx`

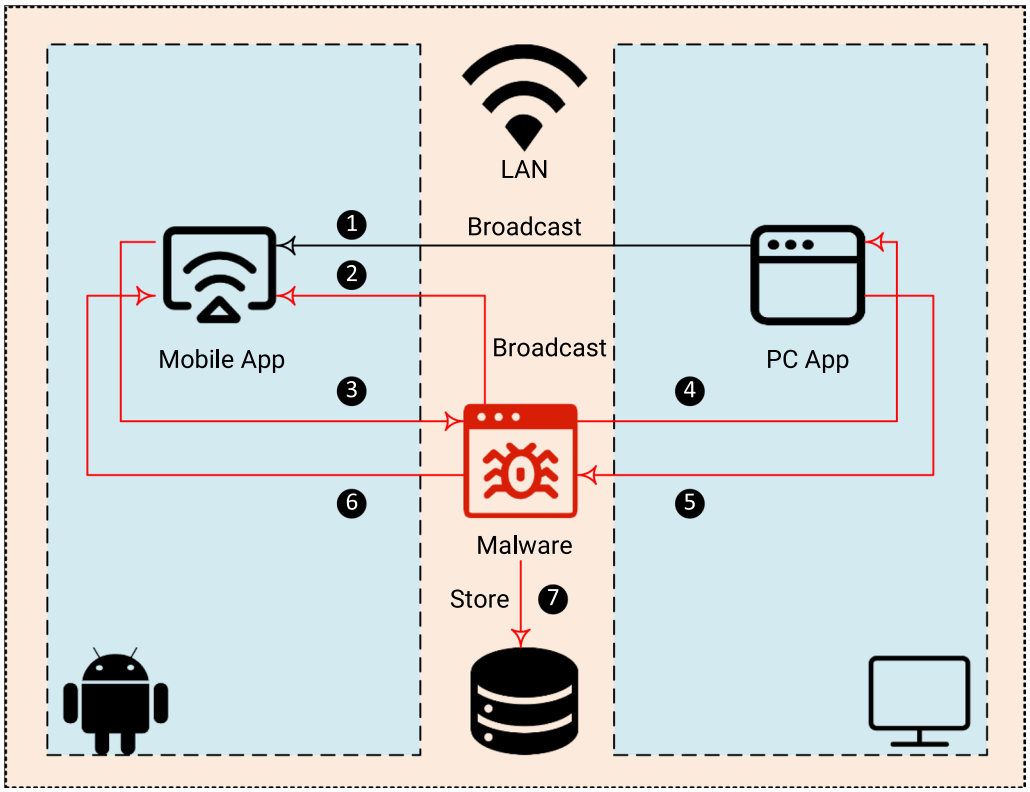

Fig. 5 PC device forgery and screen content stealing attack

installed on the victim's PC with network access, running in the background. This PC malware is built using the `Zeroconf` (Zeroconf 2024) framework.

Attack Process The main process of this attack is shown in Figure 5. When the user launches the `JinZhou` app on both the mobile and PC to mirror the screen content, the PC malware will detect the related broadcast. PC malware can forge the broadcast to make it appear in the device list on the mobile app. The mobile side of `JinZhou` app will use the broadcasted port for subsequent communication, so we set the broadcasted port to a fixed value of 1234. We set the forged hostname to `FORGED-PC`, but in practice, a more deceptive hostname, such as the real device name, can be used to trick the user into connecting.

After the mobile app connects to the PC malware, the PC malware will also establish a connection with the benign PC app using the IP address and port number from the broadcast. Once both connections are established, the PC malware will transfer the relevant data to the benign PC app to ensure normal usage for the user. In addition, the PC malware can also steal sensitive screen data for subsequent attack, such as verification codes and passwords.

Impact Such security vulnerabilities have been confirmed by `CNVD`, and four IDs have been assigned: `CNVD-2024-25211` (rated as medium severity), `CNVD-2024-25640` (rated as medium severity), `CNVD-2024-26729` (rated as medium severity), and `CNVD-2024-25639` (rated as low severity).

Case study 2: command injection attack

Some screen mirroring apps that support wireless connections and PC remote control features lack authentication mechanisms and checks on the number of connections. Some apps allow a single phone to connect with multiple PCs, enabling attackers to exploit this vulnerability to arbitrarily control the user's phone.

Attack Setup We selected the `Easyshare` app[2] as the target of this attack. This app is based on the Dual-side LAN Scheme and has over 500 million downloads on Google Play. It also allows users to control their phones from PCs, including mouse clicks and keyboard inputs. We assume that the attacker is PC malware installed on the victim's PC, which only has network access permission and runs in the background.

[2] Package name: `com.vivo.easyshare`

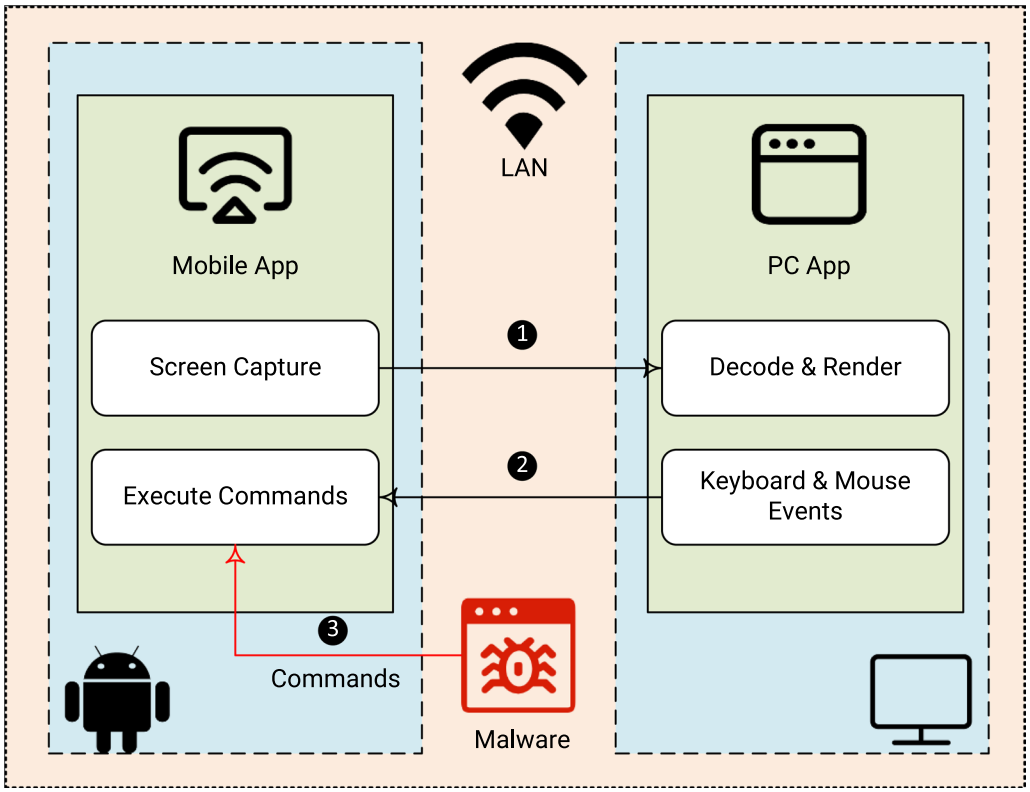

Fig. 6 Command injection attack

Attack Process The main process of this attack is shown in Figure 6. The `Easyshare` app uses the mobile app as the server and receives connection requests from the PC side via the `WebSocket` protocol, with the server using a fixed port number 10178. Due to the lack of authentication mechanisms and restrictions on the number of connected PCs, once the user launches the `Easyshare` app on the smartphone, both the PC malware and the benign PC app can establish connections with the mobile app. After the PC malware establishes connections with the mobile app, it can send remote commands to the mobile app, such as clicks and text inputs. These commands are typically transmitted in `JSON` format, allowing the attacker to easily construct malicious commands to make the victim's phone perform dangerous actions or retrieve sensitive data.

Impact Such security vulnerability has been confirmed by `vivoSRC`, rated as **high** severity.

Case study 3: race condition attack

Scrcpy-based apps usually adopt the Single-side USB Scheme, typically featuring screen mirroring and PC remote control capabilities. However, these apps lack authentication measures, resulting in a race condition vulnerability. Attackers can masquerade as a PC app and exploit the race condition to establish connections with the `scrcpy-server`, thereby intercepting screen content or injecting malicious control commands.

Attack Setup We have chosen the `Scrcpy` app (Scrcpy 2024; Scrcpy - Best Android Screen Mirroring App 2024) as the target of this attack, which is only available on the PC side. The attacker is the PC malware, which does not need access to the LAN.

Attack Process The main process of this attack is shown in Figure 7. The PC malware runs in the background and listens on port 27183, which is the default port for `Scrcpy` app. Additionally, the `Scrcpy` app also listens on this port after launching. Therefore, the PC malware needs to enable the port reuse option for the sockets. Moreover, launching the PC malware first and `Scrcpy` app afterward will significantly increase the success rate of this attack.

When the user connects the smartphone to the PC via a USB cable at a certain moment and launches `Scrcpy` app, the `Scrcpy` app pushes `scrcpy-server` to the phone and executes it. Afterward, `scrcpy-server` attempts to establish connections with the `Scrcpy` app. At this point, besides the `Scrcpy` app, the PC malware is also listening on port 27183. Due to the race condition and the lack of necessary authentication mechanisms, the connection request from `scrcpy-server` is likely to be accepted by the PC malware. Once the connections are

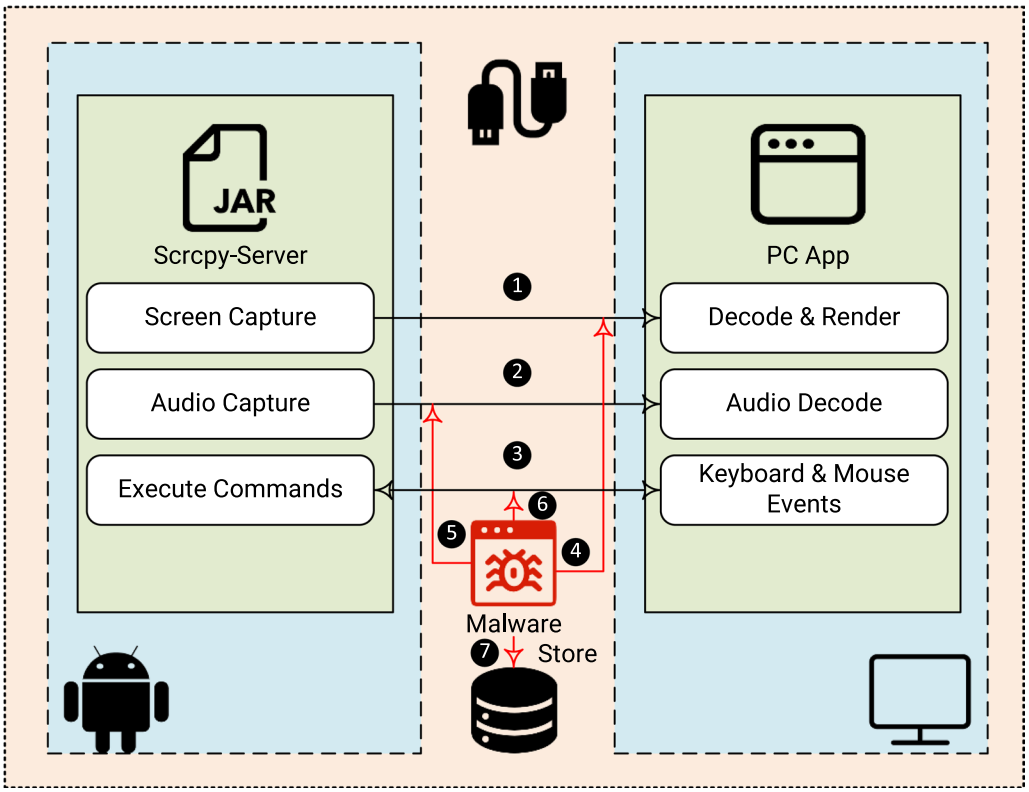

Fig. 7 Race condition attack

established, the PC malware will also establish connections with the `Scrcpy` app, intercepting and forwarding the communication data between `scrcpy-server` and the `Scrcpy` app. The user can use the app as usual, but the PC malware obtains the data transmitted during the communication, including the screen content and control commands. In addition, the PC malware can also inject remote control commands, allowing the attacker to manipulate the victim's phone arbitrarily.

Impact The vulnerability about race condition attack to obtain screen content has been confirmed by `CNVD` and `CNVD-2024-25676` (rated as low severity) has been assigned.

Case study 4: data sniffing attack

Screen mirroring apps built on Single-side HTTP Scheme, Single-side USB Scheme, and Dual-side LAN Scheme do not encrypt communication data, which could lead to data sniffing attacks.

Attack Setup We selected the `LetsView` app[3] as the target of this attack, which is based on the Dual-side LAN Scheme and has over 1 million downloads on Google Play. The attacker is one network device connected to the LAN, and we assume it to be an independent PC.

Attack Process The attacker's goal is to obtain the screen content data of the smartphone from network traffic, and there are various ways to achieve this, such as implementing ARP spoofing (Moon et al. 2016) to forward the smartphone's traffic. When the user launches the `LetsView` app to mirror the screen content to a PC, the attacker can sniff the network traffic related to the screen content on the LAN. The `LetsView` app uses the H.264 algorithm to encode video streams, and the attacker can decode the video streams to obtain some sensitive data, such as passwords and verification codes.

Results and discussion

In this section, we will describe the security analysis results of this work, and discuss about some mitigation measures as well as the limitations of this study.

Measurement in the Wild

We categorized the collected 21 mainstream screen mirroring apps according to the architectures mentioned in Section "Classification and workflow of screen mirroring" and conducted in-depth research about the security risks. As shown in Table 2, all screen

[3] package name: `com.apowersoft.letsview`

Table 2 Security analysis results of screen mirroring apps

App ID§	Architecture	SR1	SR2	SR3	SR4	Popularity‡
info.dvkr.screenstream	C1	✓	✗	✗	✓	5M+
com.livescreenapp.free	C1	✓	✗	✗	✗	500K+
com.screenmirrorapp	C1	✓	✗	✗	✗	10M+
com.nero.swiftlink.mirror	C1, C3	✓	✗	✗	✗	1M+
com.koushikdutta.vysor	C1, C2	✗	✓	✓	✓	5M+
com.wujie.connect	C4	✓	✗	✗	✗	750K+
com.vivo.easyshare	C3	✗	✓	✓	✓	500M+
com.tensorshare.phonemirror	C2	✗	✓	✓	✗	50K+
com.apowersoft.letsview	C2, C3	✗	✓	✓	✓	1M+
com.jx.jzscreenx	C2, C3	✗	✓	✓	✓	1M+
com.sigma_rt.projector_source	C3	✗	✓	✗	✓	5K+
cn.i4.mobile*	C2, C3	✗	✓	✗	✓	200K+
com.apowersoft.mirror	C2, C3	✗	✓	✓	✓	10M+
com.screencast	C1	✓	✗	✗	✗	1M+
com.easywork.easycast	C3	✗	✓	✗	✓	500K+
com.imyfone.mirrorto	C2	✗	✓	✓	✗	100K+
cn.ieway.evmirror*	C2, C3	✗	✓	✓	✓	200K+
Scrcpy	C2	✗	✓	✓	✓	103K+ Stars
AnLink	C2	✗	✓	✓	✗	rating: 4.8/5.0
Scrcpy GUI	C2	✗	✓	✓	✓	3.4K+ Stars
QtScrcpy	C2	✗	✓	✓	✓	17.2K+ Stars

§: For screen mirroring apps that have the mobile side, we use the package name as the App ID. For apps that only have the PC side, we use the app name as the App ID.

‡: The popularity of each screen mirroring app, including downloads in mobile app stores, stars in GitHub, and rating in Microsoft Store.

∗: The mobile side app is packed

mirroring apps have at least one security risk. **SR2** and **SR3** are relatively common in popular screen mirroring apps, particularly **SR2**, where 71.43% of the apps are vulnerable to MITM attacks. Additionally, 83.33% of screen mirroring apps built on the Single-side HTTP Scheme suffer from **SR1**. An attacker can exploit specific security risks to access the screen content of users' smartphone or control them. In addition, some of these apps use packer protection techniques to hide the source code. For those apps with packers, we only performed dynamic analysis, which involves examining user interactions at startup, monitoring device resource consumption and network traffic during execution, etc. Through this analysis, we uncovered their security risks based on the identified results.

Accuracy We combined static code analysis with multiple rounds of dynamic analysis. Then, we captured relevant communication traffic repeatedly to perform cross-verification, so the communication process analysis yielded results with essentially no false positives. However, due to the complexity of code implementation and the diversity of communication under specific conditions, the process may result in some false negatives. For our security analysis, the impact of these false negatives is negligible.

Mitigation measures

To mitigate the identified security risks, we propose the following recommendations for software vendors and users, respectively.

Recommendations for software vendors

Robust Authentication Mechanisms Software vendors need to adopt robust authentication mechanisms and ensure that these mechanisms cannot be circumvented.

- For screen mirroring apps built on the Single-side HTTP Scheme and Dual-side LAN Scheme, communicating entities need reliable credentials to identify each other. For example, a QR code containing a random token can be displayed on the PC app, and if the mobile app wants to establish connections with the PC app, it needs to scan this QR

code to obtain the token, with subsequent connections and communications requiring this token.
- For screen mirroring apps built on the Single-side USB Scheme, the PC app can generate a random string before startup and pass it to `scrcpy-server` via argument before launching. When establishing the connection, both of them need to validate the string first, rather than establish the connection immediately.

Adopting robust authentication mechanisms will effectively mitigate **SR1**, **SR2**, and **SR3**, particularly the race condition risk in apps like `Scrcpy`.

Data Encryption Authentication and data encryption mechanisms are inseparable. Using only authentication mechanisms without data encryption mechanisms can easily lead to authentication being bypassed. For screen mirroring apps using the Single-side HTTP Scheme, they can change to use the HTTPS protocol for secure data transmission. For screen mirroring apps using the Dual-side LAN Scheme and Single-side USB Scheme, sensitive data should be encrypted during communication. Data encryption can effectively mitigate the threat of **SR4**.

Constraining and Checking of Connection Channels Screen mirroring apps are typically used to mirror a smartphone's screen onto one PC. If the app supports connecting one smartphone to multiple PCs, users should be prompted with the current host name and the number of connected devices when connecting to a new PC. In addition, screen mirroring apps should provide an interface for managing all connected devices. If the app is designed for one-to-one connection, the connection channels need to be restricted and verified to ensure that multiple video or control connections are not allowed. These restrictions and verifications will further mitigate **SR1** and **SR3**.

Defense in Depth If the screen mirroring apps support controlling the smartphone from the PC, it is better to prompt explicit authorization before executing remote commands. For example, when the smartphone executes one remote command for the first time, the user should be explicitly asked for consent, and the remote command should only be executed after authorization. Once an unusual connection is detected, such as a large number of connection attempts, all connections should be immediately disconnected to protect the user's privacy and security.

Recommendations for users

From the user's perspective, they can take the following recommendations to mitigate the security risks associated with screen mirroring apps:

- Potential attackers on the same LAN may exploit vulnerabilities in screen mirroring apps to steal the phone's screen content or send malicious control commands to the phone arbitrarily. Therefore, users should avoid using screen mirroring apps on untrusted LANs.
- When users use screen mirroring apps based on the Dual-side LAN Scheme, they need to carefully check the information of the selected target device (such as device name) to avoid establishing connections with unknown devices.
- When the screen mirroring feature is no longer needed, the user should promptly close the established connections to reduce the potential attack surface.

Limitations

Challenges of Automated Analysis Due to the diverse implementation approaches of the Android screen mirroring feature, automated analysis poses challenges. This diversity limits the efficiency of our current analysis. Future work could involve designing specialized automated analysis solutions tailored to specific implementations, allowing for more efficient and accurate identification of security risks.

Limited Analysis Scope This study mainly focuses on the security risks of screen content leakage and malicious control of smartphones. Since the PC apps are primarily responsible for rendering and sending user's control commands, the security risks posed by them are relatively low. Therefore, this study do not perform security analysis on PC apps. Besides, some screen mirroring apps provide the functionality to mirror PC screen content to smartphones. This functionality deviates from the main topic of this research, so it has been left as a task for future studies.

Related work

Screen Casting Security There are several studies that focus on the security of user's screen content and the security of control functionality. Tian et al. (2022) conducted a detailed security analysis of the `DLNA` screen casting protocol used in smart TVs and discovered several security vulnerabilities, including inadequate protection measures and insufficient authentication mechanisms. Zhang et al. (2023) proposed the EvilScreen attack to bypass deployed authentication and isolation policies of smart TVs, and attackers can access or control smart TV resources remotely. Tekeoglu and Tosun (2015) conducted a study on the security of Chromecast screen casting and found possible security issues in its communication process. Tian et al. (2014) found security

implications in the screen sharing API of `HTML5`, and attackers can capture sensitive information from the user's screen without consensus.

The remote control feature is offered by certain screen casting apps, enabling the controlled app to project the phone's screen content to the controller app and allowing the controller to operate the phone remotely. To the best of our knowledge, few studies have specifically examined the security of this feature. Feal et al. (2020) analyzed data privacy and sharing security in Android parental control apps, identifying significant non-compliant private data sharing practices and widespread use of dangerous permissions. Wang et al. (2024) investigated the security of Android remote assistance apps, focusing on the insecure use of communication protocols and permission usage.

Unlike these studies, our research systematically analyzes the security risks associated with the custom implementation and deployment of Android screen mirroring functionality.

Android Customization The security issues introduced by Android customization have been widely investigated in numerous studies. Li et al. (2021) analyzed the security of Android custom permissions and designed CuperFuzzer to automatically detect related vulnerabilities. Yang et al. (2022) conducted a large-scale measurement to understand non-SDK APIs used in Android apps and their malicious usage. Zheng et al. (2014) designed DroidRay to detect malware that may be preinstalled in the firmware. Wu et al. (2013) investigated the problem of preinstalled apps with elevated privileges and privacy leakage. Gamba et al. (2020) conducted a large-scale security analysis of preinstalled apps on Android devices. Elsabagh et al. (2020) analyzed privilege elevation vulnerabilities in preinstalled apps on Android devices. In this study, we explored the security of customized screen mirroring apps.

Communication Security of Smart Devices Research on the security of smart devices and their communications is a popular topic. Fernandes et al. (2016) conducted an in-depth empirical security analysis and found overprivileged issues of SmartApps as well as no sufficient protection measures for sensitive information transmitted in the communication. Cui et al. (2013) found attackers may inject firmware modifications to the smart device while it is updated. Zhang et al. (2018) designed HoMonit to validate the working logic of SmartApps and detect their misbehaviors from encrypted wireless traffic. Wang et al. (2019) conducted a large-scale security analysis of smart devices from their companion apps. Recently, Nan et al. (2023) conducted a large-scale analysis of SmartApps and found lots of smart devices expose user data without proper disclosure.

Conclusion

In this paper, we systematically analyzed the architectural design of the Android mirroring feature and identified the security risks they face in real-world deployments. This study focuses on the communication and interaction process between the mobile and PC sides of screen mirroring apps, identifying four potential security risks. For these security risks, we proposed specific attack methods and demonstrated actual attack results in practical attack cases. Besides, we investigated and analyzed over 20 popular screen mirroring apps, and the results showed that almost all screen mirroring apps have security risks. Finally, corresponding mitigation measures were proposed for these identified security risks.

Acknowledgements
We would like to thank the anonymous reviewers for their detailed comments and useful feedback.

Author contributions
ZQ: Conducted the experiments and wrote the manuscript. SY: Supervised the experimental work and revised the manuscript. YY: Performed data analysis. YL: Assisted in conducting experiments. WD: Conceived the idea and led the overall project.

Funding
This work was supported by Taishan Young Scholar Program of Shandong Province, China (Grant No. tsqn202211001) and Shandong Provincial Natural Science Foundation (Grant No. ZR2023MF043).

Data availability
Not applicable.

Declarations

Competing interests
The authors declared that they have no conflict of interest.

Received: 3 August 2024 Accepted: 8 January 2025

References

Advanced Video Coding. 2024. https://www.itu.int/ITU-T/recommendations/rec.aspx?rec=14659. Accessed 8 May 2024

Alrawi O, Lever C, Antonakakis M, Monrose F (2019) SoK: Security Evaluation of Home-Based IoT Deployments. In: Proceedings of the 40th IEEE Symposium on Security and Privacy (IEEE S &P), San Francisco, CA, USA, May 19-23, 2019

Create Your Own Accessibility Service. 2024. https://developer.android.com/guide/topics/ui/accessibility/service. Accessed 4 May 2024

Cui A, Costello M, Stolfo S (2013) When Firmware Modifications Attack: A Case Study of Embedded Exploitation. In: Proceedings of the 20th Annual Network and Distributed System Security Symposium (NDSS), San Diego, California, USA, February 24-27, 2013

Don't Become the Victim of Screen Sharing Scams. 2024. https://www.uccu.com/security-alert-screen-sharing-fraud/. Accessed 29 May 2024

Elsabagh M, Johnson R, Stavrou A, Zuo C, Zhao Q, Lin Z (2020) FIRMSCOPE: Automatic Uncovering of Privilege-Escalation Vulnerabilities in Pre-Installed Apps in Android Firmware. In: Proceedings of the 29th USENIX Security Symposium (USENIX Security), Boston, MA, USA, August 12-14, 2020

Feal Á, Calciati P, Vallina-Rodriguez N, Troncoso C, Gorla A (2020) Angel or devil? a privacy study of mobile parental control apps. In: 20th Proceedings on Privacy Enhancing Technologies (PoPETs), Montreal, Canada, July 15-19, 2020
Fernandes E, Jung J, Prakash A (2016) Security Analysis of Emerging Smart Home Applications. In: Proceedings of the 37th IEEE Symposium on Security and Privacy (IEEE S &P), San Jose, CA, USA, May 22-26, 2016
Frida. 2024. https://frida.re/. Accessed 6 May 2024
Gamba J, Rashed M, Razaghpanah A, Tapiador J, Vallina-Rodriguez N (2020) An Analysis of Pre-installed Android Software. In: Proceedings of the 41st IEEE Symposium on Security and Privacy (IEEE S &P), San Francisco, CA, USA, May 18-21, 2020
High Efficiency Video Coding. 2024. https://www.itu.int/ITU-T/recommendations/rec.aspx?rec=15647. Accessed 8 May 2024
JADX. 2024. https://github.com/skylot/jadx. Accessed 6 May 2024
Lee Y, Li T, Zhang N, Demetriou S, Zha M, Wang X, Chen K, Zhou X, Han X, Grace M (2017) Ghost Installer in the Shadow: Security Analysis of App Installation on Android. In: Proceedings of the 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Denver, CO, USA, June 26-29, 2017
Li R, Diao W, Li Z, Du J, Guo S (2021) Android Custom Permissions Demystified: From Privilege Escalation to Design Shortcomings. In: Proceedings of the 42nd IEEE Symposium on Security and Privacy (IEEE S &P), San Francisco, CA, USA, May 24-27, 2021
Media Projection. 2024. https://developer.android.com/media/grow/media-projection. Accessed 4 May 2024
mitmproxy. 2024. https://github.com/mitmproxy/mitmproxy. Accessed 6 May 2024
Mobile Operating System Market Share Worldwide. 2024. https://gs.statcounter.com/os-market-share/mobile/worldwide. Accessed 26 April 2024
Moon D, Lee JD, Jeong Y-S, Park JH (2016) RTNSS: A Routing Trace-based Network Security System for Preventing ARP Spoofing Attacks. The Journal of Supercomputing
Nan Y, Wang X, Xing L, Liao X, Wu R, Wu J, Zhang Y, Wang X (2023) Are you spying on me? large-scale analysis on iot data exposure through companion apps. In: Proceedings of the 32nd USENIX Security Symposium (USENIX Security), Anaheim, CA, USA, August 9-11, 2023
Restricted Screen Reading. 2024. https://source.android.com/docs/core/permissions/restricted-screen-reading. Accessed 26 April 2024
Scrcpy. 2024. https://github.com/Genymobile/scrcpy. Accessed 4 May 2024
Scrcpy - Best Android Screen Mirroring App. 2024. https://scrcpy.app/. Accessed 4 May 2024
ScreenStream. 2024. https://github.com/dkrivoruchko/ScreenStream. Accessed 4 May 2024
TCPDUMP & LIBPCAP. 2024. https://www.tcpdump.org/. Accessed 6 May 2024
Tekeoglu A, Tosun A.Ş (2015) A Closer Look into Privacy and Security of Chromecast Multimedia Cloud Communications. In: Proceedings of the 34th IEEE Conference on Computer Communications Workshops (INFOCOM Workshops), Hong Kong, China, April 26 - May 1, 2015
Tian Y, Liu YC, Bhosale A, Huang LS, Tague P, Jackson C (2014) All Your Screens are Belong to Us: Attacks Exploiting the HTML5 Screen Sharing API. In: Proceedings of the 35th IEEE Symposium on Security and Privacy (IEEE S &P), San Francisco, CA, USA, May 18-21, 2020
Tian G, Chen J, Yan K, Yang S, Diao W (2022) Cast Away: On the Security of DLNA Deployments in the SmartTV Ecosystem. In: Proceedings of the 22nd International Conference on Software Quality, Reliability and Security (QRS), Guangzhou, China, December 5-9, 2022
vivoSRC. 2024. https://security.vivo.com.cn/#/home. Accessed 6 May 2024
Wang L, Liu X, Lei T, Song W, Guo S, Ren P (2024) Security research for android remote assistance apps. In: 29th Australasian Conference on Information Security and Privacy (ACISP), Sydney, NSW, Australia, July 15-17, 2024
Wang X, Sun Y, Nanda S, Wang X (2019) Looking from the Mirror: Evaluating IoT Device Security through Mobile Companion Apps. In: Proceedings of the 28th USENIX Security Symposium (USENIX Security), Santa Clara, CA, USA, August 14-16, 2019
Wireshark. 2024. https://www.wireshark.org/. Accessed 6 May 2024
Wu L, Grace M, Zhou Y, Wu C, Jiang X (2013) The Impact of Vendor Customizations on Android Security. In: Proceedings of the 20th ACM SIGSAC Conference on Computer and Communications Security (CCS), Berlin, Germany, November 4-8, 2013
Yang S, Li R, Chen J, Diao W, Guo S (2022) Demystifying Android Non-SDK APIs: Measurement and Understanding. In: Proceedings of the 44th IEEE/ACM International Conference on Software Engineering (ICSE), Pittsburgh, PA, USA, May 21-29, 2022
Zeroconf. 2024. https://pypi.org/project/zeroconf/. Accessed 8 May 2024
Zhang Y, Ma S, Chen T, Li J, Deng RH, Bertino E (2023) EvilScreen Attack: Smart TV Hijacking via Multi-channel Remote Control Mimicry. IEEE Transactions on Dependable and Secure Computing (TDSC)
Zhang W, Meng Y, Liu Y, Zhang X, Zhang Y, Zhu H (2018) HoMonit: Monitoring Smart Home Apps from Encrypted Traffic. In: Proceedings of the 25th ACM SIGSAC Conference on Computer and Communications Security (CCS), Toronto, ON, Canada, October 15-19, 2018
Zheng M, Sun M, Lui JCS (2014) DroidRay: A Security Evaluation System for Customized Android Firmwares. In: Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security (Asia CCS), Kyoto, Japan, June 3-6, 2014

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.