Assessing the Feasibility of the Virtual Smartphone Paradigm in Countering Zero-Click Attacks

Narmeen Shafqat[*]
Northeastern University
shafqat.n@northeastern.edu

Cem Topcuoglu[*]
Northeastern University
topcuoglu.c@northeastern.edu

Engin Kirda
Northeastern University
ek@ccs.neu.edu

Aanjhan Ranganathan
Northeastern University
aanjhan@northeastern.edu

Abstract

Zero-click attacks exploit unpatched vulnerabilities in chat apps, such as WhatsApp and iMessage, enabling root access to the user's device without their interaction, thereby posing a significant privacy risk. While Apple's Lockdown mode and Samsung's Message Guard implement virtual sandboxes, it is crucial to recognize that sophisticated zero-click exploits can potentially bypass the sandbox and compromise the device. This paper explores the feasibility of countering such attacks by shifting the attack surface to a virtual smartphone ecosystem, developed using readily available off-the-shelf components. Considering that zero-click attacks are inevitable, our cross-platform security system is strategically designed to substantially reduce the impact and duration of any potential successful attack. Our evaluation highlighted several trade-offs between security and usability. Moreover, we share insights to inspire further research on mitigating zero-click attacks on smartphones.

Keywords: Zero-Click Attacks, Zero-Day, Pegasus Spyware, Mobile Security, Virtual Smartphone.

1. Introduction

The increasing use of smartphones for social networking and banking has made them prime targets for cybercriminals. Unlike traditional cyberattacks that require user interaction through social engineering, cybercriminals now use zero-click exploits utilizing zero-day (i.e., unpatched) vulnerabilities in popular chat apps to compromise smartphones without user involvement (Amnesty, 2021). These exploits avoid persistence to evade detection by forensics and anti-malware utilities (Sheikh, R., 2021). Notably, in 2016, a text containing suspicious links received by UAE activist Ahmed Mansoor led security researchers to uncover iOS vulnerabilities exploited by the zero-click spyware, named *Pegasus*. This spyware has since targeted high-profile figures worldwide, including Amazon's CEO Jeff Bezos, Al Jazeera reporters, allies of late Saudi journalist Jamal Khashoggi, and political dissidents in Hungary, Morocco, and India (Benjakob, 2022). A successful zero-click attack allows attackers to access sensitive data (e.g., messages, contacts, photos, and files), log keystrokes, track location, and even activate the device's camera and microphone for surveillance (Figure 1), posing severe privacy and security risk. In fact, attackers only require the victim's phone number to send the exploit; hence, gaining access to the victim's contact list also puts their contacts at risk.

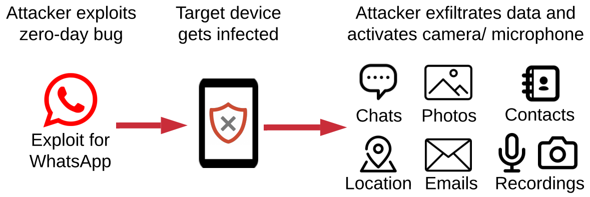
Attacker exploits zero-day bug
Target device gets infected
Attacker exfiltrates data and activates camera/ microphone
Exploit for WhatsApp
Chats
Photos
Contacts
Location
Emails
Recordings

Figure 1. Attacker exploits zero-click zero-day bug in a chat app to gain root access to the target device.

To counter zero-click attacks, Apple's iOS 16 introduced *Lockdown mode* (Apple Inc, 2022), and Samsung's Galaxy S23 implemented a virtual sandbox *"Message Guard"* (Samsung, 2023). Yet, sophisticated zero-click exploits can potentially circumvent them and compromise the device (Microsoft, 2022; Project Zero, 2017). Non-Samsung Android users also remain susceptible to these attacks. A naive way of protection is to avoid smartphones or use burner phones with

[*]Equal Contribution.

URI: https://hdl.handle.net/10125/107278
978-0-9981331-7-1

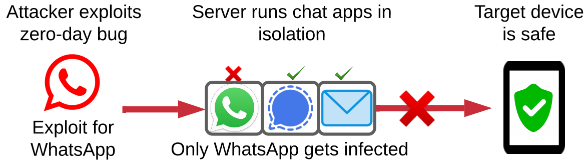

Figure 2. Envisioned Zero-Click Secure Framework: Users access remote, isolated apps via screen sharing, limiting zero-click exploit to the targeted app.

burner sims, but this restricts internet use. Mobile browser interfaces are not optimized for chat apps, and the exploit if penetrated into memory could risk device security. Using a secondary phone for chat apps is ineffective, as such exploits can leverage cross-app vulnerabilities to compromise other apps. The user may rely on desktop-based clients or web apps, e.g., for WhatsApp; however, carrying a laptop is inconvenient for real-time messaging during activities like hiking or shopping. Besides, not all chat apps have such interfaces, and permitting only SMS can jeopardize device security. The Mobile Verification Toolkit (MVT, 2021) analyzes device backups for zero-click compromise, but its effectiveness is limited to known attack vectors. Considering inevitable software bugs, it is imperative to develop a zero-click resilient framework that can at least limit information loss during an attack.

In this paper, we share our experiences while designing a zero-click secure framework for high-risk individuals, such as investigative journalists, who sought a secure method to use chat apps while ensuring maximum privacy during surveillance attacks. We analyzed the zero-click attack landscape, identified essential properties to limit the attack's impact and lifespan, and developed a remote access mechanism (Figure 2) that shifted the attack surface to a virtual smartphone ecosystem. We leveraged readily available commercial off-the-shelf (COTS) components for proof-of-concept, employing containerized Android emulators (Android, 2022c) for isolating apps and Web Real-Time Communication (WebRTC, 2021) for screen sharing and remote interaction. A user study, exempted by the Institutional Review Board (IRB), evaluated the system's usability and performance, revealing several shortcomings and challenges in securing smartphones against zero-click attacks.

This paper makes the following contributions:

1. We define design requirements to protect smartphones from zero-click attacks, emphasizing scalability and usability, while minimizing attack impact and duration.
2. We evaluate the performance and usability of the implemented zero-click secure framework and highlight challenges of using COTS components.
3. We share our experiences and lessons learned in finding a reliable and scalable sandboxing solution for 24/7 hosting of chat apps, while ensuring seamless user interaction and high-quality performance over the Internet.

Recognizing the impracticality of achieving a perfect solution to zero-click attacks, our objective is to identify challenges, limitations, and research opportunities in constructing a zero-click secure system, providing valuable insights to researchers working in this field.

2. Background and state-of-the-art

2.1. Zero-click exploits

Zero-click exploits allow the attacker to take control of a device without user interaction. They target smart apps, such as WhatsApp, Signal, and iMessage, that receive messages and calls from any source, including untrusted ones. The attacker gears a zero-click attack by sending a specially crafted message, image, or voicemail, containing malicious code, to the target device via a wireless connection (Wi-Fi, cellular, or Bluetooth). The code exploits a zero-day vulnerability in the app, granting the attacker root access to the device.

Zero-click exploits are not new. Apple discovered a zero-click vulnerability in iOS 6 in 2012 (Yasar, K., 2021). In 2020, Samsung patched a zero-click vulnerability in its graphics library that affected all their devices since 2014 (WccfTech, 2020). Zero-click exploits from QuaDream, Paragon, and Cognyte have targeted encrypted chat apps like iMessage, WhatsApp, and Signal (Gallagher, R., 2022). However, the discovery of zero-click exploits in Pegasus spyware (NSO-Group, 2021) drew significant attention due to its global misuse by nation-states to monitor their opponents and critics. Pegasus exploited WhatsApp through unanswered messages or calls, and even bypassed BlastDoor security utility on iOS devices (Amnesty, 2021; Benjakob, 2022). Typically, it operates in the phone's temporary memory without causing performance issues or excessive battery drainage, thereby evading detection by endpoint security systems.

2.2. Current state-of-the-art

The technical details of how zero-click exploits compromise smartphones are still largely unknown. Thus, existing security solutions that analyze requested permissions (Canbay et al., 2017), API and system calls (Cai et al., 2018; Saracino et al., 2016), network addresses, resource consumption (Galal et al., 2016), etc., have failed to detect and prevent them. Although the MVT tool can analyze device backups and detect known Pegasus infections, it cannot identify real-time zero-click attacks or prevent data leakage post-attack.

Unfortunately, most studies on detecting zero-day exploits focussed on non-mobile OS (Blaise et al., 2020; Kumar and Sinha, 2021), while zero-click exploits target vulnerabilities in mobile apps. The few studies on mobile-based zero-day malware relied on signature or anomaly-based detection methods. Signature-based approaches (Kouliaridis et al., 2020) only detected repackaged zero-day malware and required extensive sample collection. Anomaly-based methods (Barbhuiya et al., 2020; Jang et al., 2015) focussed on deviations from normal smartphone activity, but required high processing power and could not detect malware using anti-malware or encryption techniques. Some studies also used machine learning (Amin et al., 2020; Millar et al., 2021), but were slow and ineffective against sandbox-evading malware (Qamar et al., 2019).

Apple's iOS 16 recently introduced *Lockdown mode* as an optional protection against zero-click exploits. It restricts wired connections, shared albums, device management, and new configuration profiles. In the Messages app, it turns off link previews, search, Apple service invitations, and non-image/video attachments. If the user deactivates Lockdown mode to view specific attachments (e.g., PDF), it will expose the device to potential attacks. The mode also blocks non-contact's Facetime calls, overlooking potential threats from users' contacts. Moreover, it disables custom website fonts to prevent JavaScript code execution, resulting in missing favicons and images on web pages. Website administrators can detect its use through missing fonts and can use logged IP addresses to fingerprint devices and high-profile targets (Peterson, M., 2022). Despite its strength, Lockdown mode can be potentially bypassed by advanced zero-click exploits (Microsoft, 2022).

Samsung asserts that its Knox platform also protects against zero-click exploits involving video and audio formats, although it has been bypassed before (Project Zero, 2017). To counter image-based attacks on Samsung Messages and Messages by Google, the Galaxy S23 series introduces the *Samsung Message Guard* sandbox. It analyzes images in a virtual space on the smartphone and blocks malicious ones. However, it does not protect the device against image-based attacks on third-party apps and is limited to the S23 series.

Acknowledging that zero-day vulnerabilities will continue to exist and advanced zero-click exploits can potentially bypass sandbox to impact the user's device, we chose to migrate the chat apps remotely. While this concept of virtual smartphones is not new, prior implementations (Chen and Itoh, 2010; MITRE, 2014) are outdated and fall short against zero-day malware that utilizes native libraries and advanced obfuscation techniques. In summary, most Android users lack immediate holistic protection against zero-click attacks.

3. Zero-click secure framework

3.1. Threat scenario

Based on our discussions with dissident journalists, we envision a practical scenario where the victim, a high-profile target, relies on popular, end-to-end encrypted chat apps (i.e., WhatsApp and Signal) for official and private communication on their smartphone. The victim prefers these apps over email due to their popularity among informants and resilience against eavesdropping by authorities. The attacker aims to attain unfettered access to the victim's device to extract sensitive information, track location, or record real-time audio and video, thereby mirroring recent cases of surveillance using Pegasus. We assume that the attacker knows the victim's phone number and possesses a potent zero-click exploit code that leverages a zero-day vulnerability in one of the installed apps (e.g., WhatsApp) to compromise the smartphone without the victim's knowledge and interaction.

3.2. Desirable characteristics

We identified key properties for a zero-click resistant system that prioritize optimal security and minimize information leakage during a successful zero-click attack, with minimal system and attacker assumptions.

No direct delivery of messages to the smartphone: The mere delivery of a message containing the payload exploit is enough to compromise the device; hence it is crucial to avoid any direct delivery of messages.

Remote access to messages: Zero-click exploits can potentially bypass input sanitization or sandbox, thereby allowing malicious code to infiltrate the device memory. Hence, even temporarily receiving messages on the device or accessing them via email or web interface is deemed insecure. Users should have a remote access mechanism to retrieve messages externally.

Temporal and spatial application isolation: It is required to prevent zero-click exploits targeting a specific chat app (e.g., WhatsApp) from infecting other apps and to minimize prolonged eavesdropping.

Scalable and usable: The system should support additional apps without compromising performance and enable easy access to messages over the Internet without extensive user interaction or technical expertise.

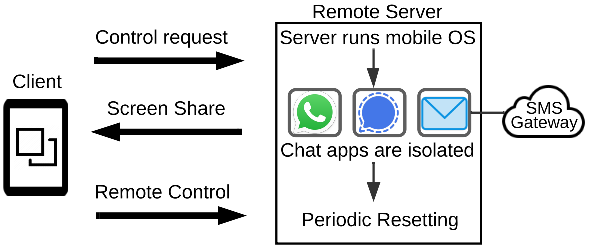

Figure 3. Zero-Click Secure Architecture: Remote app access via screen sharing, Message routing via cloud SMS Gateway, and periodic server reset.

3.3. Experimental architecture

We designed a zero-click secure architecture using COTS components that attempts to block (or reduce) all possible attack vectors. Our framework (Figure 3) shifts the attack surface by migrating sensitive apps from the user's device *(client)* to a virtual smartphone (remote server). As chat apps parse messages, even from untrusted numbers, they are an obvious attack vector. Hence, we shifted all chat apps to the server. While third-party apps like WhatsApp and Signal can be registered remotely, shifting built-in messaging app was challenging. To address this, we used a cloud-based SMS gateway that enables SMS communication on any device, including computers, over the telecom network.

Moving chat apps remotely is insufficient to prevent zero-click exploits, as they can evade input sanitization at the server and reach the client. To allow secure remote access, we developed a user-friendly screen sharing and remote control mechanism that shares screen pixels with the client, ensuring that malware cannot reach it. Additionally, the remote control feature allows the user to control the remote apps by forwarding keystrokes from the mirrored screen to the server. We recognized that zero-click exploits targeting a specific app could exploit cross-app vulnerabilities, potentially compromising the remaining apps on the server. This makes it imperative to isolate each app in a sandbox-like environment, such as separate virtual machines or docker containers. Currently, there is no evidence of *cross-OS exploitation* for zero-click attacks, as these exploits specifically abuse zero-day bugs in *smart* apps. Hence, a zero-click exploit targeting App A in sandbox A cannot escape to the non-mobile host OS (e.g., Linux server) and into sandbox B to compromise App B. This setup confines a zero-click attack to the targeted app.

To prevent prolonged eavesdropping on the vulnerable app, we periodically reset each instance to its unaffected snapshot (as required by the user). Pegasus avoids persistence to evade detection; hence, resetting the smartphone to factory settings eliminates it, terminating the attacker's connection and compelling app re-infection (Sheikh, R., 2021). In short, we aim to restrict the attack's scope by accessing remote, isolated apps through screen mirroring.

3.4. Implementation

We explored various methods for remote execution of mobile OS, app isolation, and remote interaction (Section 5). The most effective approach was running an Android emulator (Android, 2022c) on a Linux server. Android OS was chosen for its open-source nature and wider accessibility, although the client OS can be Android or iOS. We assumed that underlying computing resources (client, server, screen sharing protocol, and SMS Gateway) were secure against cyber and physical attacks, with the exception of zero-click exploitation.

Server setup and application isolation. We set up an Android cloud emulator (Google, 2021) on Google Cloud Platform (GCP) using a Linux instance, allowing us to run the official Android Emulator as a web service within Docker (Docker, 2021). Docker is an open-source platform that leverages OS-level virtualization to isolate apps in containers, providing separation from the host system. To mitigate the risk of cross-app zero-click exploitation, we assigned each app (WhatsApp, Signal, and the Message app) to a separate Docker container (Figure 4). WhatsApp and Signal were set up on the server using a Google voice number. The Message app was configured with Twilio (Twilio, 2022), a cloud-based SMS gateway service, that forwarded messages to and from the server securely over HTTPS (TLS 1.2 encryption), eliminating the need for a physical SIM card. To reduce the risk of zero-click infections, the API can be customized to allow messages from verified contacts only and disable link previews in text messages. IPhone users can also use this setup, as iMessages appear as regular SMS on the server. Moreover, with a Kernel-based Virtual Machine (KVM), we could smoothly execute Android OS on a dedicated Linux server and other cloud platforms: Amazon Web Services (AWS) and Microsoft Azure.

Remote interaction with chat apps. For security, messages were delivered to the server and then relayed to the client via screen mirroring. The Cloud Emulator utilized WebRTC protocol (WebRTC, 2021) for real-time remote screen display on the web without requiring any plugins. It eliminates the need for manual connection acceptance and sustains connectivity after the system restarts. As direct peer-to-peer connections are not always possible over the Internet, we accessed

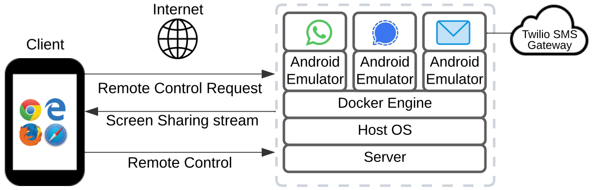

Figure 4. Framework Realization: Remote execution of chat apps in containerized Android Emulators on GCP, accessed via WebRTC on the client's browser.

the server by its public IP address and employed a TURN (Traversal Using Relays around NAT) server to relay screen contents to the client. Note that iOS does not fully support screen sharing via WebRTC on mobile web browsers. Hence, an alternate method using PNG screenshots was implemented for screen capturing. We utilized a React web application to remotely control the server by relaying user keystrokes on the mirrored screen (client) to the server. We also modified it to provide a full-screen view, enhancing user experience. This setup restricts zero-click attacks to the targeted app and does not affect other remote apps or the client.

Other security features. To limit the attack's impact and prevent eavesdropping, we periodically reset each instance to its initial state (e.g., every three days or as needed). This eliminates undetected infections and terminates the attacker's connections. Moreover, as each remote instance operates as an independent Android phone, we enabled Kiosk mode using the freely available GoKiosk app (Intricare, 2022), ensuring constant accessibility to predefined apps on the server.

4. Evaluating performance and usability

We evaluated our framework's performance and usability through: 1) *System Testing* conducted by our team, and 2) *Quality Assurance Testing* performed by prospective users. Note that the security analysis of the framework involves sending a zero-click exploit to the client and forensically examining it for potential infection. Due to the absence of zero-click binaries, we could not conduct comprehensive security testing. Nevertheless, the fact that chat apps were isolated across a cross-platform system means the established design principle of separation of privilege safeguards them against cross-application attacks, i.e., a vulnerability in one app does not compromise other apps.

4.1. Setup

We thoroughly tested our implementation on various smartphones; Android (Motorolla Nexus 6, LG V40 ThinQ, Samsung Galaxy S10E, Oppo A5, and Vivo U1), and iOS (iPhone XR, iPhone X, and iPhone 7). The server was set up on a Linux-based VM instance on GCP with nested virtualization enabled and was publicly accessible via its IP address. We utilized separate containers for each app (WhatsApp, Signal, and Message app) and configured the apps with a Google Voice number. We set up additional GCP accounts with different Google Voice numbers, allowing user study participants to test the framework without linking their personal accounts until they were fully satisfied. To encourage active participation, random texts, images, gifs, etc., were frequently shared on group chats.

4.2. System testing

We evaluated the system's performance in terms of connection time, usability, latency, and scalability.

Connection time: We evaluated it by accessing the server on multiple phones at different times and using various networks (3 Wi-Fi and 2 cellular). We utilized WebRTC-based screen-sharing for Android clients and PNG-based screen-sharing for iOS clients. We made 25 connection attempts to the remote server, documenting whether the connection succeeded on the first attempt, the number of reconnections needed otherwise, and if the connection terminated during system usage. Each experiment lasted for an hour. In all attempts, the clients were able to connect seamlessly to the server on the first attempt, no re-attempt was required, and the connection remained stable throughout the experiment.

Usability: We tested usability on devices with varying screen sizes, manufacturers, and web browsers (Chrome, Firefox, Microsoft Edge, UC, Oppo, and Opera). In all tests, the text was clear, the remote display quality was excellent, and the entire screen was effectively utilized. However, the additional communication layer introduced a slight delay that may present usability challenges in certain situations.

Measuring lag: In practice, it is difficult to precisely measure the introduced lag due to client and server asynchronization. Using dedicated Android apps, we estimated additional transmission time as half of the Round Trip Time (RTT). The client app sent a test message, and the server app replied with an acknowledgment (ACK), allowing the client app to calculate RTT as the difference in timings. We minimized measurement error by averaging the lag of ten test messages. Although RTT is influenced by network speed and traffic load, these measurements provided a quantitative value for the lag. Table 1 presents the average lag and sample's standard deviation (SD) in seconds (s) across networks. Considering the

Table 1. Lag introduced by COTS components

Client Device	Introduced Lag (s)				
	Wi-Fi			Cellular	
	A	B	C	A	B
LG V40 ThinQ	0.44	0.47	0.42	0.49	0.51
Nexus 6	0.45	0.51	0.43	0.50	0.53
Galaxy S10E	0.47	0.52	0.49	0.53	0.56
Oppo A5	0.42	0.53	0.54	0.54	0.55
Vivo U1	0.45	0.48	0.49	0.48	0.50
Average Lag = 0.49 s, Standard Deviation = 0.04 s					

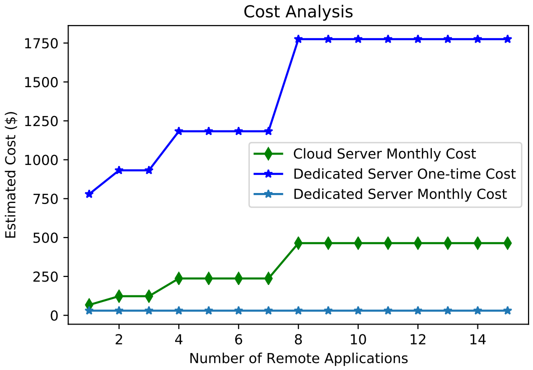

Figure 5. Cost and resource analysis per user for deploying server end on GCP vs. dedicated server.

security benefits of our solution, the user study further assessed whether the minute lag of 0.49 seconds (with 0.04 seconds SD) was acceptable to users.

Resource requirement and scalability: In practice, users may want to add more chat apps remotely (e.g., Viber, Telegram). We conducted a cost analysis for a single user based on the number of remote apps added. Each Android emulator requires a minimum of 4GB RAM and 2GB disk space to run an app efficiently. For scalability evaluation, we used a cost-effective GCP cloud instance with an Intel Haswell CPU, 8GB RAM, and 100GB storage as the baseline for running one chat app in a containerized Android emulator. We recorded the monthly costs as we increased the RAM (16, 32, and 64GB) corresponding to the number of apps. Note that users may also deploy the server side themselves to reduce recurring expenditures. For this, we used a Dell PowerEdge server with 8GB RAM and 1TB disk space ($780) as the base server for running a chat app, scaling the RAM as the number of apps increased. Figure 5 shows that the dedicated server is more cost-effective in the long run, despite the higher initial investment. However, the cloud setup is practically more secure in terms of the server's physical and logical security.

4.3. Quality assurance testing

To evaluate the usability of our envisioned system in practice, we solicited feedback from potential users.

Ethical considerations: Our study was exempted under the IRB Exemption Category 3 - Benign Behavioral Interventions by our institution's IRB, as we did not collect sensitive or personal information or involve deception or attacks. We recruited participants (aged 18+) by sending invitation emails to 30 potential targets of zero-click attacks from diverse regions, including three journalists and privacy-conscious smartphone users. Out of the 30, 27 agreed to participate. We scheduled Zoom meetings with them to obtain informed consent and provide instructions. To address privacy concerns, we pre-configured the apps using Google voice numbers and did not collect any personally identifiable information (e.g., name or email) or IP addresses. Participants' responses were analyzed and reported as group data, ensuring confidentiality. Moreover, all conversations were deleted after the study.

User study and results: Participants accessed the server via the mobile web browser using the public IP and temporary login credentials. Once logged in, they exchanged random text messages, images, gifs, etc., with saved contacts or engaged in group chats. Our team oversaw the other contacts and group chats, enabling participants to actively communicate and observe for potential delays or performance issues. After the one-hour test, participants shared their feedback regarding following through a Google survey: 1) prior knowledge of zero-click attacks, 2) experience with connecting to the remote server, 3) experience with sending test messages, and 4) suggestions for system improvement. Out of 27 participants, 21 had a prior understanding of the subject. All participants successfully connected to the remote server on their first attempt, regardless of location and time, with stable connections throughout the study. The evaluation also assessed user-friendliness (e.g., text readability, full-screen display) and if the lag caused by COTS components is acceptable. 21 participants found the system user-friendly and accepted the lag, while six participants reported significant lag despite finding the system user-friendly. Users suggested to keep the native keyboard on-screen and ensure a lag-free experience.

5. Experiences and lessons learned

Below, we summarize key lessons learned from designing and implementing a zero-click secure framework with COTS components, aiming to assist researchers in building more robust security solutions.

5.1. Running mobile OS on the server

Lesson learned # 1: *The official Android Emulator stands out as the most reliable, usable, and scalable option among the limited available solutions for running the mobile OS virtually 24/7 in portrait orientation.*

To set up a virtual smartphone on a Linux-based dedicated server initially, we explored local and cloud-based emulators for Android and iOS. Emulating iOS was challenging as Apple restricts it to iOS devices and Xcode simulator (Apple, 2023), making even web-based iOS emulators like Appetize.io (Appetize, 2022) and Corellium (Corellium, 2022) costly for 24/7 service.

In contrast, we found various open solutions for emulating Android devices (Table 2). The official Android Emulator, despite having high system requirements and limited devices with PlayStore, offered a stable environment. This is because we ran one app per emulator, which prevented these limitations from affecting the user experience. We also explored Cuttlefish (Android, 2022b), an Android virtual device that replicates physical phone behavior at the OS level. However, Cuttlefish required excessive resources, and its compatibility was restricted to API levels after 28 and a few Debian distributions, making it unsuitable to set up multiple Android instances. Unofficial Android gaming emulators like Memu (Memu, 2021) and BlueStacks (BlueStacks, 2021) had sluggish performance, booting issues, excessive ads, and were often flagged as malicious by antivirus software. We also tested Android-x86 OS (OSDN, 2021), a port of the Android Open Source Project (AOSP) for Intel x86 or AMD-powered devices, on VirtualBox, but it was resource-intensive even for running a single chat app. The cloud-based emulators we explored, Anbox (Anbox, 2018) and GenyMotion (GenyMotion, 2021), also had limitations. Anbox, while stable, could not display the screen in portrait mode despite having screen rotation enabled. GenyMotion ran Android OS in portrait orientation using a custom AOSP ROM and an OpenGL-capable graphics card but was costly ($0.5 per hour per instance) and lacked a built-in Google Play Store and compatibility with most Google Apps (GApps). Besides, Microsoft's Windows 11 subsystem for Android (Microsoft, 2021b) encountered crashing issues and also lacked Play Store. Overall, the official Android emulator proved to be the most reliable option for remotely running mobile OS in portrait mode.

5.2. Sandboxing applications on the server

Lesson learned # 2: *Android Application Sandbox is ineffective against zero-click cross-app and cross-OS attacks. Isolate apps via virtualization or containerization, with security and resource trade-offs.*

Android's default Application Sandbox (Android, 2022a) runs each app in a separate process with a unique User ID (UID), protecting OS applications, native code, and higher-level components. However, it can be bypassed by zero-click exploits, granting unauthorized root access (WccfTech, 2020). To enhance isolation, we explored virtualization and containerization on the server. Running apps in Android Emulator within separate virtual machines provided the highest security against cross-app and cross-OS exploits; as the exploit cannot exfiltrate from Android instance A running app A to the Linux host and into Android instance B to compromise app B. However, this approach was resource-intensive. Alternatively, we used Docker to run multiple instances of Android Emulator, isolating each app in a separate container. Containerization offered a lightweight solution to mitigate zero-click attacks and was thus preferred over separate virtual machines.

Table 2. Current remote mobile OS solutions.

#	Solution	Limitations
1	Xcode simulator	Need iOS device, Costly,
2	iOS web emulators	Limited run-time, Costly,
3	Official Android Emulator	Specific requirements, Few PlayStore devices,
4	Cuttlefish	Resource intensive, Specific requirements,
5	Android Gaming Platforms	RAM hungry, many ads booting issues,
6	Android-x86 VM	Doesn't boot on cloud, Resource intensive,
7	Anbox	No portrait mode, No PlayStore, few distros,
8	GenyMotion	Costly, No PlayStore, Not GApps compatible,
9	Windows 11 Android Subsystem	Not stable, Specific system requirements.

5.3. Implementing server on cloud platform

Lesson learned # 3: *Mobile OS requires hardware virtualization for enhanced performance and graphics, but running it on bare-metal cloud servers is expensive and lacks long-term scalability. Alternatively, employ software acceleration to run mobile OS on the cloud.*

After successfully implementing a containerized Android Emulator setup on our Linux server, we tested its feasibility on cloud platforms like GCP, AWS, and Azure. However, the emulator's performance in the cloud was sluggish. To avoid the high costs associated with bare-metal cloud instances or GPUs to enable machine and graphic acceleration (Android, 2021), we opted for software acceleration. By simulating GPU processing using the computer's CPU, we achieved satisfactory performance when running the Android Emulator in Docker containers on the cloud.

5.4. Exploring remote connection protocols

Lesson learned # 4: *Remote connection protocols (VNC and RDP) default to landscape orientation, which is inconvenient for viewing on the client. In contrast, WebRTC offers screen sharing in portrait mode.*

Next, to establish remote connectivity between the server and client without direct message delivery, we explored popular remote connection protocols like Microsoft's Remote Desktop Protocol (RDP) (Microsoft, 2021a) and Virtual Network Computing (VNC) (Richardson et al., 1998). These protocols are primarily designed for desktop use and only support landscape display orientation, which is not user-friendly. Despite several attempts, the shared screen remained in landscape mode. In contrast, WebRTC offers screen-sharing capabilities in both web and native applications, supporting portrait mode. Hence, we integrated WebRTC's API with a React web application to provide screen sharing and remote control services.

5.5. Exploring remote connection apps

Lesson learned # 5: *Unlike computer-to-computer and mobile-to-computer screen sharing apps, mobile-to mobile solutions are scarce. Most of them are also expensive, limited to local networks, lack remote control features, and do not allow unattended server access.*

We evaluated popular screen sharing and remote control apps on Google Play Store and Apple App Store against WebRTC. Most apps, including Remote Chrome Desktop (Google LLC, 2020), RealVNC (RealVNC, 2021), and SplashTop (Splashtop, 2022), required either the server or client to be a desktop computer, leaving the screen orientation issue unresolved. Therefore, we focused on mobile-to-mobile remote access solutions listed in Table 3. While Skype (Skype, 2022), ScreenTalk (RMT, 2020), and Inkwire (ClockworkMod, 2020a) offered high-quality screen sharing, they lacked remote control functionality. Vysor (ClockworkMod, 2020b) and Scrcpy (Pkk3345678, 2021) required physical device access, which was impractical for our setup. DroidVNC-ng (Beier, C., 2022), RemoDroid (Dimitrov, I., 2020), and AirpowerMirror (Apowersoft, 2022) provided both features but had stability, latency, and network limitations. Proprietary solutions like TeamViewer (TeamViewer, 2022), AnyDesk (AnyDesk, 2022), and AirDroid (SandStudio, 2021) offered quality screen sharing and remote access but were expensive and lacked control over the mirrored data. Hence, we preferred WebRTC for our needs.

Table 3. Mobile-to-mobile screen sharing utilities.

#	Solution	Limitations
1	Skype	No remote control functionality,
2	ScreenTalk, Inkwire	Sluggish performance, No remote control functionality,
3	Scrcpy, Vysor	Requires physical connection to mirrored screen,
4	DroidVNC	Unstable (often crashes),
5	RemoDroid	Local network, Slow, Root req,
6	Airpower	Local network only, Slow,
7	TeamViewer, Anydesk	Very costly license,
8	AirDroid	Limited usage in free version.

5.6. Open-ended security and usability issues

Our virtual smartphone setup mitigates zero-click attacks but entails several usability and security trade-offs to enhance security and user experience.

Security: Keeping the server updated with latest COTS versions is key to reducing attack vectors. While chat apps can be vulnerable to zero-click attacks, a compromised SMS gateway can also risk the Messages app. However, spatial isolation localizes the attack to the affected app. Our setup also benefits iOS users. While hosting iMessages remotely is costly, our setup balances client security and iMessage features (e.g., encrypted chat and screen effects) by receiving iMessages as regular SMS on the server, thus mitigating iMessage-based exploits. Upon availability of exploit binaries, we plan to assess our system's resilience by intentionally compromising the vulnerable app and using MVT to ensure other apps and the client at least remain protected against known zero-click exploits.

Usability: Our implementation lacks a notification mechanism, requiring manual checking for new messages. As iOS and Android's sandbox allow limited content sharing across apps, developing a server-side notification app is only feasible with rooting/ jailbreaking the phone, which can compromise device security. This trade-off between security and usability is similar to secure government systems that require manual message checking. Unlike Apple's Lockdown mode, our setup allows remote viewing of any file type, but received files are only saved until the server reset time. Additionally, while videos and audio messages can be received, enabling audio playback requires relaying the audio of communicating parties.

6. Conclusion

Zero-click attacks exploit unpatched vulnerabilities without user interaction, posing privacy risks. While sophisticated zero-click exploits can potentially bypass device-based sandboxing, this paper explored the feasibility of countering such attacks using COTS-based virtual smartphone ecosystem. Essentially, we redirected the attack from the client by running chat apps in remote, isolated instances, accessed via screen sharing. This confined the attack to the vulnerable app, highlighting that zero-click attacks can be mitigated but not entirely eradicated. We evaluated our setup through an IRB-exempted user study, with 21 out of 27 participants expressing full satisfaction. We also shared lessons learned and highlighted additional components required to achieve optimal security, albeit at the expense of usability. Our work aims to inspire further research in countering zero-click spyware.

References

Amin, M., Tanveer, T. A., Tehseen, M., Khan, M., Khan, F. A., & Anwar, S. (2020). Static malware detection and attribution in android byte-code through an end-to-end deep system. *Future generation computer systems*, 112–126.

Amnesty. (2021). *Forensic methodology report: How to catch nso group's pegasus doc 10/4487/2021* (tech. rep.). Amnesty International. UK.

Anbox. (2018). *Android in a box*. https://anbox.io/

Android. (2021). *Configure hardware acceleration for the Android Emulator*. https://developer.android.com/studio/run/emulator-acceleration

Android. (2022a). *Application Sandbox*. https://source.android.com/security/app-sandbox

Android. (2022b). *Cuttlefish Virtual Android Devices*. https://source.android.com/setup/create/cuttlefish (accessed: 03.11.2023)

Android. (2022c). *Run apps on Android Emulator*. https://developer.android.com/studio/run/emulator

AnyDesk. (2022). *AnyDesk Remote Desktop*. https://play.google.com/store/apps/details?id=com.anydesk.anydeskandroid

Apowersoft. (2022). *ApowerMirror- Screen Mirroring*. https://play.google.com/store/apps/details?id=com.apowersoft.mirror

Appetize. (2022). *Appetize - Run native mobile apps*. https://appetize.io/ (accessed: 04.10.2023)

Apple. (2023). *Xcode*. https://developer.apple.com/documentation/xcode (accessed: 04.13.2023)

Apple Inc. (2022). *About Lockdown Mode*. https://support.apple.com/en-us/HT212650

Barbhuiya, S., Kilpatrick, P., & Nikolopoulos, D. S. (2020). Droidlight: Lightweight anomaly-based intrusion detection system for smartphone devices. *Proceedings of the 21st International Conference on Distributed Computing and Networking*, 1–10.

Beier, C. (2022). *droidVNC-NG*. https://play.google.com/store/apps/details?id=net.christianbeier.droidvnc_ng

Benjakob, O. (2022). *The nso file*. https://www.haaretz.com/israel-news/tech-news/2022-04-05/ty-article-magazine/nso-pegasus-spyware-file-complete-list-of-individuals-targeted/0000017f-ed7a-d3be-ad7f-ff7b5a600000

Blaise, A., Bouet, M., Conan, V., & Secci, S. (2020). Detection of zero-day attacks: An unsupervised port-based approach. *Computer Networks*, *180*, 107391.

BlueStacks. (2021). *BlueStacks Play Bigger*. https://www.bluestacks.com/ (accessed: 05.17.2023)

Cai, H., Meng, N., Ryder, B., & Yao, D. (2018). Droidcat: Effective android malware detection and categorization via app-level profiling. *IEEE Transactions on Information Forensics and Security*, *14*(6), 1455–1470.

Canbay, Y., Ulker, M., & Sagiroglu, S. (2017). Detection of mobile applications leaking sensitive data. *5th International Symposium on Digital Forensic and Security*, 1–5.

Chen, E. Y., & Itoh, M. (2010). Virtual smartphone over ip. *IEEE International Symposium on World of Wireless, Mobile, Multimedia Networks*, 1–6.

ClockworkMod. (2020a). *Inkwire Screen Share*. https://play.google.com/store/apps/details?id=com.koushikdutta.inkwire

ClockworkMod. (2020b). *Vysor - Android control*. https://play.google.com/store/apps/details?id=com.koushikdutta.vysor

Corellium. (2022). *Virtual devices with real accuracy*. https://www.corellium.com/

Dimitrov, I. (2020). *RemoDroid*. https://play.google.com/store/apps/details?id=de.im.RemoDroid

Docker. (2021). *Docker*. https://docs.docker.com/get-started/overview/ (accessed: 04.10.2023)

Galal, H. S., Mahdy, Y. B., & Atiea, M. A. (2016). Behavior-based features model for malware detection. *Journal of Computer Virology and Hacking Techniques*, *12*(2), 59–67.

Gallagher, R. (2022). *Zero-click' hacks are growing in popularity*. https://www.inquirer.com/

business/zero-click-hacks-spy-phone-pegasus-20220227.html

GenyMotion. (2021). *Android as a service*. https://www.genymotion.com/ (accessed: 05.17.2023)

Google. (2021). *Build Your Own Cloud Emulator*. https://source.android.com/devices/automotive/start/avd/cloud_emulator (accessed: 04.10.2023)

Google LLC. (2020). *Chrome Remote Desktop*. https://play.google.com/store/apps/details?id=com.google.chromeremotedesktop

Intricare. (2022). *Gokiosk*. https://play.google.com/store/apps/details?id=net.intricare.enterprisedevicekiosklockdown

Jang, J.-w., Kang, H., Woo, J., Mohaisen, A., & Kim, H. K. (2015). Andro-autopsy: Anti-malware system based on similarity matching of malware and malware creator-centric information. *Digital Investigation*, *14*, 17–35.

Kouliaridis, V., Barmpatsalou, K., Kambourakis, G., & Chen, S. (2020). A survey on mobile malware detection techniques. *IEICE Transactions on Information and Systems*, *E103.D*(2), 204–211. https://doi.org/10.1587/transinf.2019INI0003

Kumar, V., & Sinha, D. (2021). A robust intelligent zero day cyber-attack detection technique. *Complex & Intelligent Systems*, *7*(5), 2211–2234.

Memu. (2021). *Memu*. https://www.memuplay.com/

Microsoft. (2021a). *Understanding RDP*. https://docs.microsoft.com/en-us/troubleshoot/windows-server/remote/understanding-remote-desktop-protocol (accessed: 04.10.2023)

Microsoft. (2021b). *Windows Subsystem for Android*. https://docs.microsoft.com/en-us/windows/android/wsa/ (accessed: 05.17.2023)

Microsoft. (2022). *Gatekeeper's achilles heel*. https://www.microsoft.com/en-us/security/blog/2022/12/19/gatekeepers-achilles-heel-unearthing-a-macos-vulnerability/

Millar, S., McLaughlin, N., del Rincon, J. M., & Miller, P. (2021). Multi-view deep learning for zero-day android malware detection. *Journal of Information Security and Applications*, *58*.

MITRE. (2014). *SVMP System Design and Architecture*. https://svmp.github.io/architecture.html

MVT. (2021). *MVT*. https://github.com/mvt-project/mvt

NSO-Group. (2021). *Pegasus - Product Description*. https://s3.documentcloud.org/documents/4599753/NSO-Pegasus.pdf

OSDN. (2021). *Android x86*. https://osdn.net/projects/android-x86/releases (accessed: 04.05.2023)

Peterson, M. (2022). *Apple's secure Lockdown Mode may reduce web browsing anonymity*. https://appleinsider.com/articles/22/08/25/apples-secure-lockdown-mode-may-reduce-web-browsing-anonymity (accessed: 05.17.2023)

Pkk3345678. (2021). *Scrcpy*. https://play.google.com/store/apps/details?id=com.wujiyun.scrcpy.pro

Project Zero. (2017). *Lifting the (hyper) visor: Bypassing samsung's kernel protection*. https://googleprojectzero.blogspot.com/2017/02/lifting-hyper-visor-bypassing-samsungs.html

Qamar, A., Karim, A., & Chang, V. (2019). Mobile malware attacks: Review, taxonomy & future directions. *Future Generation Computer Systems*, *97*, 887–909.

RealVNC. (2021). *VNC Viewer*. https://play.google.com/store/apps/details?id=com.realvnc.viewer.android

Richardson, T., Stafford-Fraser, Q., Wood, K. R., & Hopper, A. (1998). Virtual network computing. *IEEE Internet Computing*, *2*(1), 33–38.

RMT. (2020). *Screen Talk*. https://play.google.com/store/apps/details?id=com.kentuckyrmt.screentalkremotemobile

Samsung. (2023). *Message guard protects you from new and invisible threats*. https://news.samsung.com/global/samsung-message-guard-protects-you-from-new-and-invisible-threats

SandStudio. (2021). *AirDroid*. https://play.google.com/store/apps/details?id=com.sand.airdroid

Saracino, A., Sgandurra, D., Dini, G., & Martinelli, F. (2016). Madam: Effective and efficient behavior-based android malware detection and prevention. *IEEE Transactions on Dependable and Secure Computing*, *15*(1), 83–97.

Sheikh, R. (2021). *"Legal" Spyware*. https://wccftech.com/legal-spyware-how-pegasus-exploits-iphones-through-zero-click-bugs/

Skype. (2022). *Skype*. https://play.google.com/store/apps/details?id=com.skype.raider

Splashtop. (2022). *Splashtop Personal Access*. https://play.google.com/store/apps/details?id=com.splashtop.remote.pad.v2

TeamViewer. (2022). *TeamViewer Remote Control*. https://play.google.com/store/apps/details?id=com.teamviewer.teamviewer.market.mobile

Twilio. (2022). *Send SMS with Twilio*. https://www.twilio.com/docs/sms (accessed: 04.10.2023)

WccfTech. (2020). *Samsung Finally Patches the 0-Click Vulnerability*. https://wccftech.com/samsung-finally-patches-the-0-click-vulnerability/

WebRTC. (2021). *Real-time communication for web*. https://webrtc.org/ (accessed: 04.10.2023)

Yasar, K. (2021). *What Is a Zero-Click Attack*. https://www.makeuseof.com/what-is-a-zero-click-attack-what-makes-it-so-dangerous/