IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 25, 2021, accepted December 11, 2021, date of publication December 21, 2021,
date of current version December 30, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3137360

PoPL: Proof-of-Presence and Locality, or How to Secure Financial Transactions on Your Smartphone.

YONAS LEGUESSE[1], CHRISTIAN COLOMBO[1], MARK VELLA[1], (Member, IEEE), AND JULIO HERNANDEZ-CASTRO[2]

[1]Department of Computer Science, University of Malta, 2080 Msida, Malta
[2]School of Computing, Cornwallis South, University of Kent, Canterbury CT2 7NZ, U.K.

Corresponding author: Yonas Leguesse (yonas.leguesse.05@um.edu.mt)

This work was supported by the Lawful evidence cOllecting and Continuity plAtfoRm Development (LOCARD) Project under Grant H2020-SU-SEC-2018-832735.

ABSTRACT The security of financial apps on smartphones is threatened by a class of advanced and persistent malware that can bypass all existing security measures. Strong cryptography and trusted on-chip hardware modules are powerless against sophisticated attacks that supplant device owners through device input record/replay functionality, effectively hijacking their credentials, privileges, and actions. In this paper, we introduce Proof-of-Presence and Locality (PoPL), a new security measure that tackles this threat by leveraging sensors to prove the physical presence of device owners and therefore discriminate between malware-initiated transaction requests and legitimate ones. Moreover, PoPL neither imposes the expense of additional hardware nor compromises app usability. In order to demonstrate PoPL's practicality, we developed *PoPLar*, a challenge puzzle implementation of the PoPL concept that ensures usability even on limited screen sizes by the use of a dendrogram. We have made it available as an open-source library ready to be integrated with minimal effort with existing apps. We demonstrate PoPLar's effectiveness and ease of integration through case studies involving apps from the three top cryptocurrency exchanges and an open-source crypto wallet.

INDEX TERMS Cryptocurrency exchange app security, mobile malware, puzzle-based authentication, usable security.

I. INTRODUCTION

Smartphones are becoming an increasingly convenient way to process and exchange sensitive information with online services, and security-sensitive financial transactions are no exception. Mobile device vendors are well aware of this and try to offer secured mobile platforms based on the usage of trusted hardware components and strong cryptography [1]. Yet, these measures are not stopping cyberattacks from threatening the security of smartphone-based financial transactions, with cryptocurrency exchange apps being a particularly notable case in point [2].

Recently, the additional threat of performing these attacks with a significantly reduced forensic footprint was demonstrated [3]. The implication is that malware can hijack valuable accounts without being found out, at least not before it is too late, with the possibilities of recovering funds and accurate criminal attribution being severely reduced. In this context, a particularly worrying attack vector is Android accessibility, posing a long-term threat. No obvious mitigation exists unless one is ready to discard it, which comes with its own set of consequences and concerns all customers with special needs. Ultimately, all smartphone users can benefit from accessibility services on occasion, such as while driving a car or multitasking through one's schedule [4]. Accessibility trojans can hijack an app's UI-user channel, effectively taking over access to all credentials and privileges [5], even on secured mobile platforms, without the need of rooting/jailbreaking the device. Notorious examples of malware exploiting this attack vector include the Gustuff, Cerberus and DEFENSOR ID mass malware [6], [7]. While Android may be a prime target due to its market share, accessibility threats for iOS and other platforms are a ticking time bomb [8].

The associate editor coordinating the review of this manuscript and approving it for publication was Feng Lin.

Financial apps offer various security measures, for example, the use of two-factor authentication (2FA) or the temporary disabling of fund transfer functionality [9]. 2FAs and many types of multi-factor authentication [10] commonly involve the sending of a verification code through a channel other than the one used for user/transaction authentication. The premise is that security is increased when malware cannot thwart the additional channel(s). Yet, this approach can only be effective against device input record/replay threats if we can count on having a separate hardware token. If a compromised device offers an attacker the ability to record and replay device inputs, for example touch inputs, then it could also provide the ability to replicate an entire authentication sequence that makes use of the same input. Tools such as accessibility services, or ADB shell's getevent, sendevent, and input debugging tools, can be abused to facilitate record/replay device input attacks.

The replayed device input steps can also cover the retrieval and input of 2FA verification codes, defeating any form of freshness-based mitigation. For example, soft tokens, such as the ones sent via SMS or email and even the ones generated locally by applications such as Google Authenticator, despite being convenient, can still be easily read by accessibility trojans through UI channel subversion. Ultimately, the use of secure hardware offers the ideal solution, with even single-factor authentication being secure enough for most scenarios, e.g. Yubico Key with FIDO2 [11]. However, this incurs a non-negligible expense and becomes a hindrance to app usability, possibly scaring away potential customers.

In this work, we propose Proof-of-Presence and Locality (PoPL), a new security mechanism to defend against device input record/replay threats targeting financial apps. Our proposal works by proving the physical presence and locality of a smartphone's user to the remote exchange service provider through the solution of a challenge that cannot be recorded/replayed or automated (see Figure 1) nor solved by a remote attacker. PoPL works by leveraging the accelerometer sensor (though other sensors can be used as well). A successful challenge solution allows us to verify the presence of a legitimate local user as opposed to a remote attacker, even if the attacker has compromised the device and has remote control over it.

We chose the accelerometer sensor since it is the most common motion sensor found on smartphones, and is protected by default through the Android sensor security model that does not allow write access to the sensor [12].

As a reference implementation, we developed *PoPLar* that combines the malware-resilience of accelerometer sensors and a dendrogram-centric challenge that is perfectly suitable for a smartphone's limited screen size. Moreover, our proposal does not burden users with an overly lengthy or annoying activity. PoPLar can secure financial transactions in smartphones without incurring any additional economic costs. Experimentation with the Android apps of the main cryptocurrency exchanges, namely Binance, Coinbase and Huobi, demonstrate the futility of their current security measures in withstanding accessibility attacks. In a case study over the open-source Bitcoin Wallet app, we also show how integrating PoPLar into an existing app only requires minimal effort.

Overall, we make the following contributions:-

- *PoPL*: a Proof-of-Presence and locality countermeasure against device input record/replay threats on mobile apps.
- We demonstrate PoPL's effectiveness to mitigate accessibility attacks on three popular Android crypto exchange apps, without the need for additional hardware tokens, while ensuring usability. This was demonstated through PoPLar, a PoPL implementation based on mobile device sensors and a dendrogram-centric challenge.
- We made PoPLar publicly available as an Android library, and we demonstrate its ease of integration using the case study of a popular open-source crypto wallet.

II. BACKGROUND

Nowadays, smartphones are one of the most convenient media from which to initiate financial transactions. While users have generally accepted that fund transfers require more robust authentication methods than an app login, developers must also ensure usability or face a customer exodus. The threat of accessibility trojans is a long-term one precisely because it exploits this usability-security tension, in turn, also requiring a long-term solution.

A. FINANCIAL APPS

The convenience of performing financial transactions from a smartphone has popularised financial apps so much that nowadays one finds mobile-exclusive banking services [13], e.g. Revolut, and several virtual asset management services, that push their mobile app as their primary customer interface [14]. Besides being ubiquitous and easily connected to online services, smartphones offer simple app deployment. In addition, near-field-communication (NFC) and Bluetooth interfaces can connect them to point-of-sale terminals and other smartphones, offering further fund transfer options.

This is also the case with crypto exchanges, with their apps allowing users to buy/sell cryptocurrency. In general, users of financial apps are presented with security challenges prior to any substantial transfer of funds. Our PoPLar could be an example of such a challenge.

B. MULTI-FACTOR AUTHENTICATION AND SECURE HARDWARE TOKENS

2FA or even multi-factor authentication methods [15] typically cater to the need for additional authentication as a means of extra protection for sizeable fund transfers in financial apps. 2FA aims to blend the something you *know*, *are* or *have* authentication factors. The premise is that an attacker needs to work harder to bypass an aggregate of these components.

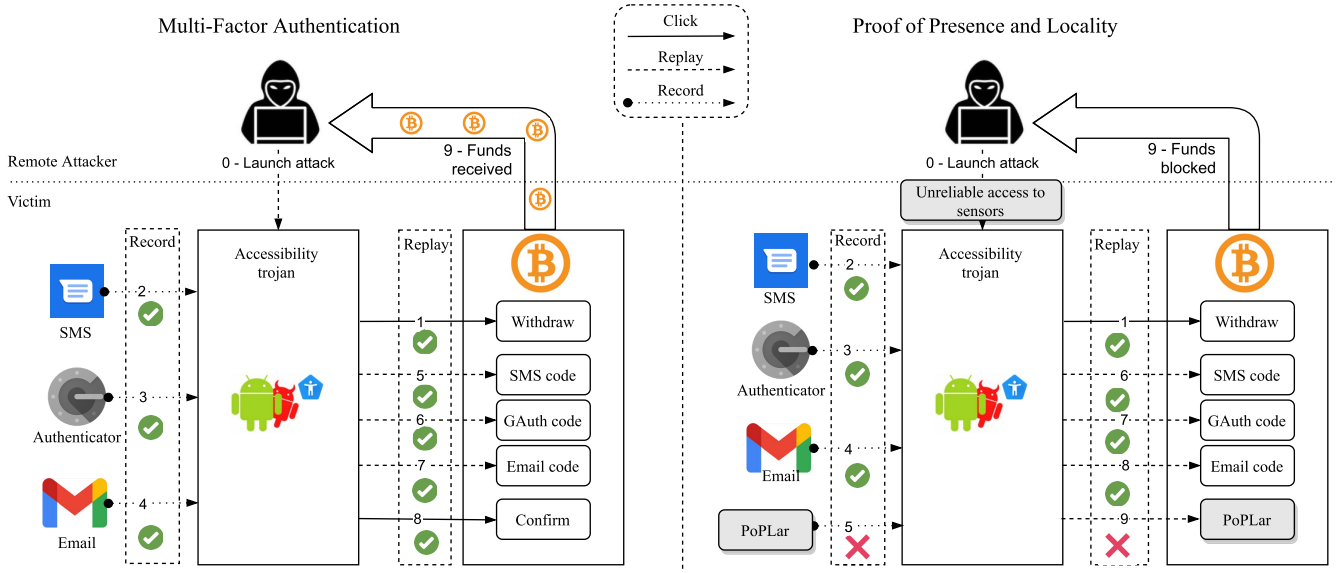

FIGURE 1. Successful accessibility trojan attack on crypto app (left) and an unsuccessful run blocked by PoPLar (right).

Online banking services have long used verification codes generated by hardware tokens.

In this case, the PIN-enabling token access provides the *know* factor, while the token itself is the *have* factor. Biometrics present the ideal something you *are* factor, but until practicality/privacy challenges are sorted out, CAPTCHAs [16] provide the next-best option by being able to discriminate between humans and computer programs (or bots). This kind of classification is, therefore, somewhat useful for PoPL.

With the onset of mobile phones, these devices started to be used as a replacement for physical 2FA tokens [17], [18], which is sub-optimal when the same smartphones are used for performing financial transactions. In order not to compromise the convenience of employing the smartphone as a token, soft 2FA tokens in the form of authentication apps, e.g. Google Authenticator, can be used. However, this option severely impacts security since any malware that takes over both the financial and the authenticator app effectively annuls any added security. A more secure alternative could be the adoption of secure hardware-based authentication solutions, e.g. FIDO2 [11], where verification codes can be provided by external hardware devices, e.g. Yubico Security Key. This way, a simpler authentication is offered, but at the loss of the single device convenience which is not to be underestimated in terms of customer retention [19], especially when considering the additional costs it incurs. PoPL attempts to strike a balance between not giving up on the single device approach while at the same time offering an extra layer of security at no additional cost.

C. USABLE SECURITY AND SMARTPHONE SENSORS

Usable security relates to the psychological acceptance of various security mechanisms. This concept has been studied mainly in the context of graphical passwords [20] and CAPTCHAs [16], [21]. Interestingly enough, smartphones are also revolutionising the world of usable security through their sensors, mainly the accelerometer and the gyroscope. In fact, sensor-based CAPTCHAs compare favourably in multiple studies [22], [23] to their classical counterparts. While puzzle-based CAPTCHAs tend to come across as a nuisance, sensor-based ones are generally considered as enjoyable due to being game-like.

Other works have also focused on accessible authentication schemes, for example, rhythm-based ones [24] intended for visually impaired users. Additional studies went to the extent of providing indirect biometrics based on user interactions with location and environment sensors, as well as the phone's touchscreen, in what is being called Be(havioral)CAPTCHAs [25], [26]. With PoPL, the concept of a sensor-based challenge is taken to a level where it can also withstand accessibility attacks.

D. PROGRAMMATIC DEVICE INPUT INTERACTION

Modern smartphones have built-in sensors that measure motion, orientation, and various environmental conditions. The device's screen is also equipped with sensors that allow the user to interact directly with what is displayed on the screen through touch. The operating system is responsible for receiving and interpreting the raw data received from the sensors as they sense physical environmental interactions.

There exist instances where the operating system also allows for programmatic device input interaction, allowing software to simulate operations that are otherwise triggered through sensor interactions. For example, using the Accessibility services, apps can read touch interactions using the `onAccessibilityEvent` event, as well as interact with the UI elements displayed on the screen using the `performAction()`. In so doing, they enable the development of assistive apps, e.g., voice-controlled apps, that interface with users in an alternative manner other than via GUI.

Besides accessibility, Android also offers debugging tools, namely ADB's `input`, `getevent`, and `sendevent` that also provide programmatic access to device input interaction. These tools are often used for debugging, testing, and automation. The `input` command-line tool can simulate a number of input events, such as touch interaction using the `adb shell input tap x y` command. The `getevent` tool provides information about input devices and a live dump of kernel input events, while `sendevent` is able to simulate event operations on the input devices.

The ability to interact with device inputs can have significant consequences on the security of the device. It can effectively replace users in their interaction with smartphone apps, inheriting all of their credentials and privileges while defeating security measures without even requiring rooting/jail-breaking [5], [12]. Given their sensitive nature, both accessibility services and ADB debugging are protected by special user-granted permissions. Yet, social engineering has been shown to be able to defeat the former [27] approach, while ADB shell privilege abuse has been shown to defeat the latter [12].

The getevent/sendevent tools are written in C,[1] and form part of toolbox utility. They interact directly with the `/dev/input` directory of the Linux input subsystem. While most Android devices offer these debugging tools, their functionality depends on the OEM specific hardware and firmware implementation, as well as the Android version. For example, some OEM implementations may not expose the accelerometer input device to get/sendevent, while more recent Android versions deny write access to input devices through sendevent on unrooted devices. This is implemented within the `adbd` process[2] when it drops its capabilities to those associated with the `AID_SHELL` group id.

On the other hand, `input`[3] operates at the Java layer of the Android framework and is based on the `android.hardware.input.InputManager` class. Therefore, all requests are mediated by Android's system server Linux user-space process and, unlike the case of get/sendevent, is accessible by `adbd` even on unrooted devices.

Notwithstanding this, programmatic device input interaction can pose a significant threat, possibly one that can defeat existing authentication security measures short of introducing additional hardware-based tokens. This is where a new mechanism like PoPL can contribute to improving security at no extra cost and with minimal additional burden.

III. PROOF-OF-PRESENCE AND LOCALITY (POPL)

We refer to Proof-of-Presence and Locality (PoPL) as a general security measure to distinguish between actions taken by users physically interacting with an application, as opposed to malware interacting with a device through programmatic device input. This can also be seen as a way to distinguish a local bot or malware from a legitimate user. Alternatively, PoPL can also be employed to tell apart a legitimate local user from a remote attacker, even if the attacker has compromised the device and has remote control over it.

At the same time, in our target scenario, it is fundamental not to incur additional hardware costs while ensuring usability. PoPL is not intended as a sole solution for the threat but rather as an additional layer aimed at reinforcing existing countermeasures. In the remainder of this work, we specifically discuss *PoPLar* as one such concrete realisation and a reference implementation of this concept.

A. THREAT MODEL

1) DEVICE INPUT RECORD/REPLAY ATTACK

In this work, we focus on the device input record/replay class of attacks. These attacks have the ability to record input data during device interaction, and later replay the input data, thus mimicking the device interactions programmatically. The threat considers device input record/replay attack vectors made possible through the aforementioned programmatic device input interaction, specifically ones that make use of accessibility or the ADB debugging tools, namely `get/sendevent` and `input`.

2) THREAT VECTOR AND ASSUMPTIONS

The assumption is that the attack targets a non-rooted[4] Android device. With locality-based security mechanisms in place, the attacker is restricted to performing all of the attack steps from the victim's device. A trojan that evades App-Store scanning [28] and is able to record and replay device input interactions, is subsequently downloaded and granted permission by the victim through social engineering [27]. Device input record and replay functionality allow trojans to programmatically replicate recorded sequences of input interaction according to the chosen attack vector.

Once the trojan is on the device, it starts interacting with the phone's financial apps in a concealed manner, either through overlays [29] or via sheer speed of operation, while the user is not taking notice or cannot interrupt it. The attack is carried out entirely on the victim's device by a trojan that acts as a proxy for a remote attacker. The attacks within scope can also defeat 2FA, whose second factor leverages apps on the compromised smartphone. Given that the phone is non-rooted we assume that the attacker does not have the ability to take screenshots belonging to PoPLar, which are protected by Android's `LayoutParams.FLAG_SECURE` feature. This feature is commonly used in banking [30] apps, to protect sensitive UI elements.

B. SMARTPHONE SENSOR SURVEY

Table 1 shows the results of a survey carried out to determine sensor availability on smartphones, using data from the

[1] toolbox/getevent.c

[2] adb/daemon/main.cpp, private/android_filesystem_config.h

[3] cmds/input/src/com/android/commands/input/Input.java

[4] If the phone is rooted, the attacker would be able to bypass any controls in place.

PHONEDB website,[5] which maintains the world's largest and most detailed mobile device database. The *device specs* search was used to filter the entire Android smartphone list, totalling 12,691 on 23/01/2021, according to sensor availability. While there is no source providing the exact number of devices in circulation using each sensor, these figures indicate the trend among OEMs when it comes to including the additional sensors. We primarily focus on Android devices due to its largest market share. iOS devices, with the second-largest market share, have a more restricted device range and therefore are significantly more homogeneous.

The first two columns of Table 1 show the sensors sorted according to availability, while the last two show the minimum required API levels and permissions. It is desirable that the proposed PoPL mechanism has the widest support possible, so the top four sensors all lend themselves well for this purpose. Specifically, the accelerometer sensor can be used to detect device tilting, upon which one can construct a rolling-ball puzzle with acceleration forces along the x-y-z axes.

C. POPLAR

The following is a way of summarising and bringing together all the requirements that our proposal needs to take into consideration:

1) SECURE IN THE THREAT MODEL CONTEXT

Given the threats associated with device input record/replay attacks, the approach needs to ensure that a trojan is not able to both read the content of the challenge and interact with the challenge. For example, by limiting the challenge input to the accelerometer sensor, PoPL can ensure that accessibility trojans are unable to interact with the challenge. Moreover, the trojan should not be able to lure the user into solving the PoPL challenge through a combination of overlays and social engineering.

2) USABILITY AND ACCESSIBILITY

PoPL needs to be easy to use, ideally with a gamification element that users find fun. Furthermore, given that the main attack vector we want to hinder is enabled through accessibility, we also wish our approach to be accessibility-friendly. This is logical since if accessibility were not a requirement, then the best protection would be to simply turn off the accessibility capabilities of the app.

3) WIDESPREAD AVAILABILITY OF TECHNOLOGY

As highlighted, technology already exists to solve this problem in the form of dedicated secure hardware. However, we wanted our proposal to be based on widely available technology which users already carry around.

4) EASY TO DEPLOY ON A LARGE SCALE

Generating PoPL challenges which satisfy the above conditions should not require human intervention to design new puzzles or challenges.

[5]https://phonedb.net

TABLE 1. Smartphone sensor availability, according to *PHONEDB*.

Sensor	%	API	Permissions
Accelerometer	99.3	3	-
Proximity	89.9	3	-
Light	89.5	3	-
Compass	87.5	3	-
Gyroscope	51.6	9	-
Finger Print	47	23	USE_FINGERPRINT USE_BIOMETRIC
Hall	20.6	n/a	-
Barometer	11.6	9	-
Gesture	5.6	n/a	-
Step counter	5.1	19	ACTIVITY_RECOGNITION
Heart rate	4.2	20	BODY_SENSORS
Temperature	1.7	3	BODY_SENSORS
Humidity	1.22	9	-
Face recognition	1.1	29	USE_BIOMETRIC
Iris	0.9	29	USE_BIOMETRIC
Altimeter	0.8	14	-

The approach adopted by *SenCAPTCHA* [22] is currently the closest that satisfies the listed requirements. This approach makes use of the gyroscope sensor to push a ball towards a puzzle keypoint - an animal's eye. The puzzle expects a trajectory close to optimal to fend off brute-force attempts, while image mutation has been shown to render senCaptcha resilient to multiple machine learning attacks. The use of the gyroscope sensor renders fake sensor inputs out of reach of the record/replay attacks considered by our threat model (see Section IV-A2). For the same reason, image mutation to counter ML attacks is not required either. Therefore, the manual work necessary to seed senCaptcha puzzles, while not compromising practicality, could be eliminated altogether. Additionally, it is also desirable that PoPL's puzzle design will support alternative sensor interactions in the long run while not compromising the level of security. The addition of sensors is envisaged in future development with the aim of making PoPL itself more accessible. On the other hand, *SenCAPTCHA's* puzzle was designed with only the gyroscope sensor in mind.

Our proposal — PoPLar — entails users physically tilting the phone to direct a ball through a unique path, towards a green area on the screen. The maze is in the shape of a dendrogram, with each juncture branching into two, for simplicity. Should a wrong path be taken, the user can tilt the phone backwards to correct the situation. Once the green path is encountered, the user can continue downwards as before, until the final level is successfully completed and the initiated transaction is executed. Figure 2 provides an illustrated walk-through of the challenge as it appears on the screen. Each level displays three left-right choices, i.e., eight possible final outcomes per screen, with only one correct path. The number of screens (referred to as *levels*) is set by the app provider depending on the desired security level.[6]

PoPLar ticks all the requirements identified above by: (i) Presenting an easy puzzle/challenge in the form of a game that is also amenable to accessibility; we have catered for colour blindness by showing the path of the ball using a

[6]We expect five levels to provide an adequate security margin for most scenarios, with $(2^3)^5 = 32,768$ combinations, but these can easily be pushed up to 10 or higher.

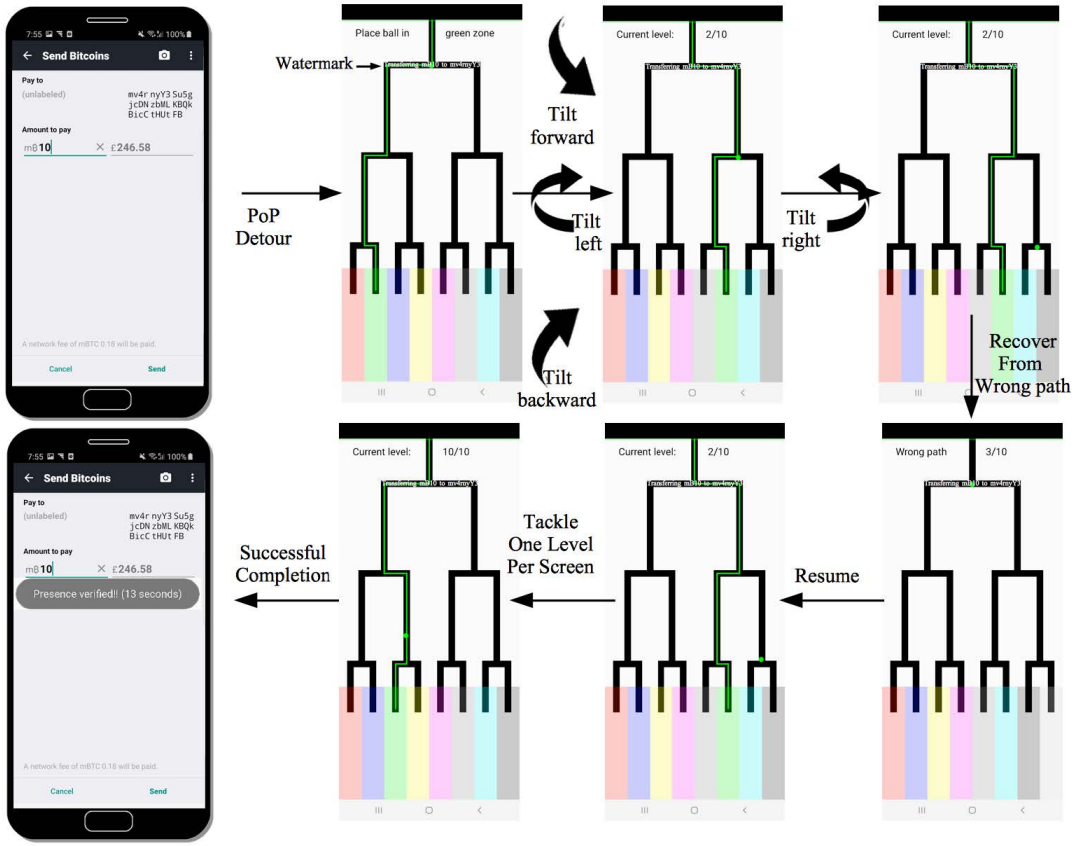

FIGURE 2. PoPLar challenge: Roll the ball down the path leading to a green zone, by physically tilting the smartphone.

lighter colour, and audio cues can easily be provided for users with visual impairment. Furthermore, we kept the interface as clean as possible by limiting the number of choices to three per screen/level. (ii) By using the accelerometer, as opposed to other sensors (e.g. fingerprint), our approach is readily usable by over 99% of smartphone users (see Section III-B). (iii) Given that our threat model assumes the presence of a device input record/replay trojan, which can be attempting transfers in the background when the user is not aware, several precautions are taken to protect against this scenario: First, the puzzle is generated randomly, using a cryptographically strong random number generator,[7] with a significant number of combinations to avoid the possibility of a random combination of movements from a user unknowingly matching the solution pattern (using 10 levels translates to 1,073,741,824 combinations). Second, PoPLar is triggered only once the very last app UI widget has been interacted with, i.e., not giving the trojan any further opportunity to interfere with the UI flow from the time the user proves presence to the time of the intended action. Third, to protect against elaborate social engineering attacks tricking the user into solving the puzzle in a different context (e.g., by presenting it as a game during a trojan-initiated transfer unbeknown to the user), we write the details of the transfer on the path which the ball needs to follow, in the form of a watermark. In this way, overlays would prove futile as they would also hide the ball. Finally, given that the attacker does not have programmatic access to both the puzzle and the accelerometer, record/replay and AI-aided attacks are not an issue. (iv) PoPLar does not require any human input to generate different puzzles, making it easily scalable. It also brings no additional cost to the app provider and is easy to integrate within an existing application, as demonstrated in the next section.

Moreover, PoPL's simple yet effective dendrogram design provides the advantage of possible support for additional sensors. For example, by using the camera sensor with facial detection tools, the puzzle's solution could involve controlling the ball movement through facial gestures or head tilting.

We made our proof of concept (POC) PoPLar implementation for Android available for download.[8] PoPLar is also available as a Jitpack library, allowing app developers to easily integrate it into their app.[9]

IV. EVALUATION

We evaluate PoPLar's effectiveness to protect financial apps from accessibility trojans by first demonstrating successful attacks on three popular crypto exchange apps without

[7]https://developer.android.com/reference/java/security/SecureRandom

[8]https://github.com/PoPDroid/AndroidPoP/tree/main/app/release

[9]https://github.com/PoPDroid/AndroidPoP

this countermeasure. We then show how PoPLar foils these attacks on an unrooted Android phone.

We also demonstrate that simply disabling accessibility on specific UI elements, a mitigation measure put in place by a crypto exchange following our report, is still vulnerable to device input record/replay attacks. We then demonstrate how PoPLar itself is not vulnerable to existing accelerometer sensor manipulation driven device input record/replay attacks.

We finally demonstrate usability by measuring authentication times for varying security levels, and also show the ease by which PoPLar can be integrated into an existing app without requiring an overhaul of its source code.

A. ATTACK RESILIENCE

1) EXPERIMENT SETUP

In order to demonstrate PoPLar's effectiveness in securing financial apps, we chose the Android ones for the top three cryptocurrency exchanges[10]: Binance (v1.21), Coinbase (v6.55.0) and Huobi (v6.1.1). Between them, these apps have registered 16M+ installs on Google Play. Fortunately, setting up accounts on cryptocurrency exchanges with a small amount of funds is relatively easy, especially when compared to more traditional banking apps. This allowed us to set up accounts across different exchanges, and therefore test against different apps. The key steps of all attacks launched are summarised in Table 2. In all cases, we focus on the details for the replay aspect of the attacks. The record aspect was approached as a preparatory phase of the attacks, where a device/app-specific profiling exercise resulted in an attack phase/step sequence to be replayed.

At first, attacks were attempted using the accessibility attack vectors against all three apps. This vector has the least prerequisites and is widely used in the wild today. The first three attack phases are preparatory steps that bypass security measures. The *Unlock* phase gets the accessibility trojan past the unlock screen (e.g. unlock PIN or Pattern). The *Address White-list* phase makes sure that the attacker's deposit address is white-listed, and therefore the withdrawal component can't transfer the funds. The *Get 2FA code* component targets the soft-token 2FA app in question, i.e. SMS, email, or Google Authenticator. The attack culminates in the *Withdrawal* phase to steal funds while dealing with all the aforementioned security measures. To attain further stealth, the accessibility trojan may also opt to include an *Overlay* phase to hide all attack activity away from the user.

The key steps of the accessibility attacks make use of Android accessibility service APIs.[11] These APIs are callable from any app that registers an accessibility service component that extends `AccessibilityService` and is granted the `BIND_ACCESSIBILITY_SERVICE` permission. `onAccessibilityEvent(AccessibilityEvent event)` is the core event handler, where all information about an UI event can be retrieved from `event`, including all window UI widgets represented as a tree of `AccessibilityWindowInfo` and `AccessibilityNodeInfo` objects. In turn, these widgets can be interacted with using the latter's `performAction()` method. In the steps of Table 2 we assume the availability of the `getWidget(root, id)` function, that given a window's `root` node and its `id`entifier, returns the corresponding UI widget. It is noteworthy that all app UI navigation steps are app UI layout-specific, necessitating steps specific to different versions of the same app. Optionally, attacks can also hide underneath window overlays. The downside of this approach is that it additionally requires the `SYSTEM_ALERT_WINDOW` permission. Finally, `Intents` are core Android object enabling apps to interact with Android system services and other apps as well. In the case of these attacks, the core Android services are: `AccessibilityManager` (needed for all accessibility steps), `ActivityManager` (handles all calls to `startActivity()`), `ClipboardManager` (for copy/pasting the stolen 2FA verification codes), and `WindowManager` (for displaying overlays). All attacks were executed on a Samsung Galaxy S8 and a Google Nexus 5x running Android Pie and Oreo, respectively.

There may be cases where a widget cannot be interacted with using accessibility's `performAction()`, namely when the developer disables accessibility access to the widget by setting the `isImportantForAccessibility` flag to false. In such cases, an attacker can resort to the `adb shell input` attack vector. As shown in Table 2 the attack requires an initial installation phase, where the app and native service components are installed over `adb` following the setup described in SMAShedD [12]. The replayed steps interact with the victim app's UI widgets using `adb shell input tap x y`, where x and y represent the screen position of the disabled widget. This attack was protoyped using a bash script over an `adb` session and was tested on Binance (v1.42.5) since Binance disabled the `isImportantForAccessibility` flag on the withdrawal button following our report. The attack was successful on an unrooted Samsung Galaxy A30s running Android 10.

To demonstrate the resiliency of PoPLar against a device input record/replay attack, we attempted to record the accelerometer readings of a successful puzzle solution, and replay the recorded readings in an attempt to solve a second puzzle. The recording of sensor movements was done using `getevent`, whereas the replay was done using the `sendevent`. This approach is similar to the one used by RERAN [31] and SMAShedD [12], and once again assumes the installation phase described in the latter. This attack was prototyped with AndroidViewClient[12] and tested on a Nexus 5x (Android 8), Samsung A30s (Android 10), and Samsung J3x (Android 5.1.1), all of which were not rooted. We were unable to perform the attack on the Samsung A30s and Nexus

[10] As ranked by `coinmarketcap.com` on 24/01/2021

[11] https://developer.android.com/reference/android/view/accessibility/package-summary

[12] https://github.com/dtmilano/AndroidViewClient

TABLE 2. Device input record/replay attacks & key phases and steps.

Phase	Steps
Attack vector: Accessibility	
Unlock	[Detect app launch] `AccessibilityEvent.TYPE_VIEW_TEXT_CHANGED` → `accEvent.getPackageName().equals("com.app")` → [Steal pattern] `startActivity(intent) new Intent(this, com.attack.FakeBinanceActivity)` → [Time elapses + Re-launch + Unlock] `context.startActivity(new Intent.setComponent("com.app"))` → [Detect app launch + get root node] → `AccessibilityNodeInfo root = getRootInActiveWindow()` → [Search for next node to click + click + repeat] `getWidget(root, id).performAction(ACTION_CLICK)`
Address White-list	[Launch app] `context.startActivity(new Intent.setComponent("com.app"))` → [Navigate up till account settings starting from root] `AccessibilityNodeInfo root = getRootInActiveWindow()` → [Search for next node to click + click + repeat] `getWidget(root, id).performAction(ACTION_CLICK)` → [Add address to white-list] `getWidget(root, id).performAction(ACTION_SET_TEXT)` → [Save settings: Search for next node to click + click + repeat] `getWidget(root, id).performAction(ACTION_CLICK)`
Get 2FA code	[Launch app] `context.startActivity(new Intent.setComponent("com.2faapp"))` → [Navigate up till view code starting from root] `AccessibilityNodeInfo root = getRootInActiveWindow()` → [Search for next node to click + click + repeat] `getWidget(root, id).performAction(ACTION_CLICK)` → [Read and copy 2FA code] 2fa = `getWidget(root, id).performAction(ACTION_GET_TEXT)` → `getSystemService(Context.CLIPBOARD_SERVICE).setPrimaryClip(clip).ClipData.newText(label, 2fa)` → [Exit app: Search for next node to click + click + repeat] `getWidget(root, id).performAction(ACTION_CLICK)`
Withdrawal	[Launch + unlock app] `context.startActivity(new Intent.setComponent("com.app"))` → [Unlock] → [Navigate up till withdrawal screen starting from root] `AccessibilityNodeInfo root = getRootInActiveWindow()` → [Search for next node to click + click + repeat] `getWidget(root, id).performAction(ACTION_CLICK)` → [Input withdrawal details: Type for each field] `getWidget(root, id).performAction(ACTION_SET_TEXT)` → [Confirm + exit: Search for next node to click + click + repeat] `getWidget(root, id).performAction(ACTION_CLICK)`
Overlay (optional)	[Accessibility service `onCreate`] → [Launch Android settings] `startActivityForResult(news Intent(android.provider.Settings.ACTION_SETTINGS), 0)` → [Navigate up till com.trojan starting from root] `AccessibilityNodeInfo root = getRootInActiveWindow()` → [Navigate till overlay permission] `getWidget(root, id).performAction(ACTION_CLICK)` → [Display overlay] → `prms=new WindowManager.LayoutParams(..,WindowManager.LayoutParams.TYPE_APPLICATION_OVERLAY,..)` → `this.getSystemService(WINDOW_SERVICE).addView(..., params)`
Attack vector: `adb shell input`	
Installation	[Install native service] `adb push binsrvc /data/local/tmp` → `adb shell chmod u+x /data/local/tmp/binsrvc` → `adb shell /data/local/tmp/binsrvc &` [Install app] `adb install app.apk` → [Start app & grant INTERNET permission] → [Connect to socket] `SocketChannel.open(StandardProtocolFamily.UNIX)` → `SocketChannel.connect(...)`
Replay	[App requests replay] `SocketChannel.write()` → [Native service executes replay]`input tap x y` →*
Attack vector: `adb shell get/sendevent`	
Installation	ditto
Replay	[App requests replay] `SocketChannel.write()` → [Native service executes replay]`sendevent /dev/input/x <recorded events>` →*

5x because `sendevent` was write protected on the former, and the accelerometer device was not visible on the latter. We, therefore, performed the experiment on the Samsung J3x running Android 5.1.1.

2) RESULTS

Table 3 shows the attack outcomes on all three crypto exchange apps and on the PoPLar challenge itself. In all cases, the first column shows the target of the attack, while the second column shows the type of record/replay attack used. In the case of Binance (v1.42.5), the accessibility attack was supplemented with the use of ADB shell's `input` to bypass the mitigation measure implemented by the exchange. The next four columns show the additional support attacks required in order to circumvent security measures or else to gain additional stealth. The included attacks depended on those measures supported by the individual apps. As for the *Overlay*-based stealth, in the case of Huobi we opted out of using it to demonstrate the alternative of executing the attacks at lightning-speed, basically making any human reaction to stop them implausible. Ultimately, in all attacks on crypto exchanges, the attacks succeeded in performing illicit withdrawals despite all the security measures in place.

For each case, the same attacks steps were attempted on PoPLar as a standalone app.[13] The inability of the record/replay attack to bypass the PoPL challenge would result in a failed attack, as depicted in Figure 1.

The underlying functionality of the dendrogram library only allows for sensor-based input to control the UI elements. The only input that is able to control the puzzle movement is the accelerometer. Therefore, the only way to solve the challenge is through physical sensor manipulation.

To further test PoPLar's resiliency, we attempted to perform a record/replay attack against PoPLar, specifically targeting the accelerometer. The aim was to record a successful PoPLar challenge, and try to replay the solution to solve the next solution. The dendrogram approach already protects against a record/replay attack by design since the random puzzle generation ensures that an attacker cannot predict the next puzzle solution. Notwithstanding this, we decided to test an attack with the assumption that the subsequent puzzle is equal to the previous one.

While newer Android versions protect the sensors through the sensor security model, which in itself protects PoPLar,

[13] https://github.com/PoPDroid/AndroidPoP/tree/main/app/release

we were able to read from and write to the accelerometer sensor by using `getevent` and `sendevent` on an older non-rooted Samsung J3x, running Android 5.1.1. However, even in this limited case, the record/replay attack on the accelerometer, as performed by SMASheD, failed due to the continuous nature of the sensor. While we were able to record the accelerometer raw data during a successful PoPLar puzzle solution, replaying the recorded data does not result in the same movement since the replayed data is mingled with the actual sensor data, that is being received during the replay. In our test, we received circa 90 accelerometer events per second. Replaying the recorded 90 events results in the device receiving the recorded 90 events plus the additional 90 events received within that second. The above resulted in a failed attack. This further strengthens the argument in favour of the accelerometer, or any other motion, sensor for controlling the PoPL puzzle. An attack would fail even in exceptional cases where an attacker has the ability to inject their own data in order to simulate a user. This is not the case with other inputs such as simple screen taps.

Videos for the accessibility attacks against crypto exchanges are available[14] for download. Screenshots from the Huobi Withdrawal attack are shown in Figure 3. In this particular attack, the exchange was configured to protect withdrawals with SMS, Google authenticator and email verification security mechanisms. The accessibility trojan starts by launching the victim exchange app and makes its way to the withdrawal page. It then populates the withdrawal information, specifically the attacker's wallet address and the withdrawal amount (All funds). At this point, the exchange prompts the 2FA verification challenges, and the trojan then opens the SMS, Google Authenticator, and Gmail apps in sequence, stealing all of the verification codes in the process. Finally, the trojan returns to the exchange app and populates all three verification codes before confirming the transaction.

3) RESPONSIBLE DISCLOSURE

We have responsibly disclosed our findings (i.e. accessibility-based withdrawal attacks) to the three cryptocurrency exchange apps. Binance acknowledged our attack within a day. Within a month, they took mitigation measures specifically aimed against this attack. Their solution was to release a patch disabling accessibility features for critical UI elements. They also awarded us a $500 bounty for the report. However, this approach still leaves the app vulnerable to other device input record/replay attacks.

Coinbase responded, acknowledging our report after a week. They surprisingly lowered the level of the report to low due to the 'difficulty of the exploit'. However, they awarded us a $200 bounty for the responsible disclosure. They have since released a significantly different version of their app. Strangely enough, their new software does not implement any direct mitigation measures against this attack. Consequently, this can still be carried out. We reported the issue to Huobi in early February 2021, but at the time of writing, they have not yet acknowledged our report.

[14]https://github.com/PoPDroid/AccAttacks

B. USABILITY

1) EXPERIMENT SETUP

We also conducted a usability study to get a feel of the average time required to complete a PoPLar authentication, as well as the user's overall impression of the usability of the PoPLar dendrogram-centric challenge. The usability study was performed using Amazon Mechanical Turk, which was approved by our institution's research ethics and data protection committee, and in total, 102 participants completed the survey assignment. The participants were given a maximum of 40 minutes to complete their task, and they were paid $4 (USD) for their participation. The average time spent per assignment was 22 minutes and 36 seconds. The assignment consisted of three main steps:

a: PoPLar CHALLENGE TIMING

First, they were asked to install and run a test PoPLar app that was published to the Android Playstore specifically for the purpose of this study. The test app would guide the participants in performing the dendrogram-centric challenge with different levels of depth (from 3-10). The users were asked to perform each depth level three times, and the results were saved to the device's clipboard, allowing the participants to paste their results into the survey.

b: USABILITY OF EXISTING MECHANISMS

Second, the participants were asked to assign a score from 1-10 indicating the usability of various popular authentication mechanisms, namely: Google Authenticator, SMS 2FA, email 2FA, hardware token 2FA, hardware authentication device (e.g. Yubikey), Google's reCAPTCHA, and fingerprint authentication. The user was asked to rate the usability based on factors such as the time to authenticate, convenience, and ease of use, where a score of 1 represents ''Horrible - (Slow and complicated)'', and 10 represents ''Great - (Fast, convenient, and easy to use)''.

c: USABILITY OF PoPLar

Finally, the participants were asked to rate the usability of the PoPLar dendrogram-centric challenge, when compared to popular and already accepted authentication mechanisms.

2) RESULTS

Table 4 shows the mean, median, and standard deviation survey usability scores of each authentication mechanism in the survey, including PoPLar. The results show that PoPLar's dendrogram puzzle approach was well received by the end-users, even in comparison with existing and widely used authentication mechanisms.

Figure 4 shows the mean duration for passing different PoPLar levels in the 3-10 range. Each participant performed the PoPLar challenge three times per level, meaning that in

TABLE 3. Attack outcomes for illicit withdrawal, before and after our proposed solution.

Target	Record/Replay	Unlock	White-list	2FA	Overlay	Attack w/o PoPLar	Attack w/ PoPLar
Binance(v1.21)	Accessibility	Pattern	Wallet address	SMS+Email+Google	Used	Success	Fail
Coinbase	Accessibility	PIN	Not available	SMS+Google	Used	Success	Fail
Huobi	Accessibility	Pattern	Not available	SMS+Email+Google	Unused	Success	Fail
Binance(v1.42.5)	Accessibility+input	Pattern	Wallet address	SMS+Email+Google	Used	Success	Fail
PoPLar	getevent/sendevent	N/A	N/A	N/A	N/A	N/A	Fail

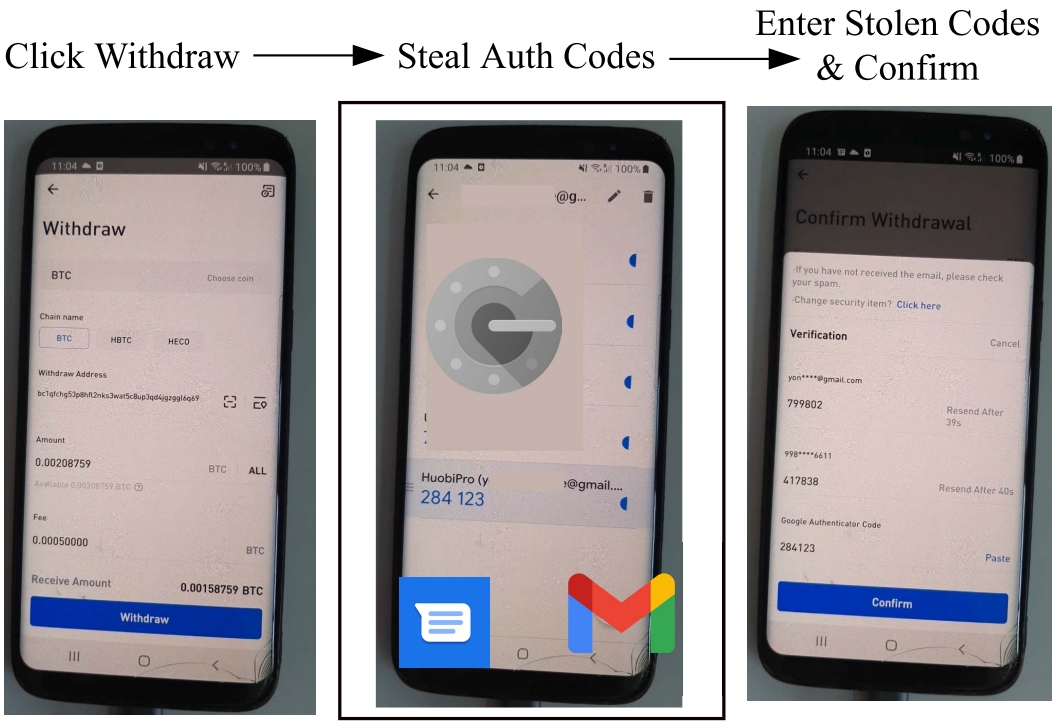

FIGURE 3. Walk-through of the attack on one of the apps, while all security measures were enabled.

total, every level was tested 303 times. The mean duration times fall in the 13131-36209 millisecond range for levels ranging from 3-10. It is important to consider that users will likely not be performing financial transactions in substantially large batches, particularly from a smartphone, so these times are, in our opinion, practical enough and compare favourably with those required by other mechanisms such as SMS, email or soft token verification. It is also worth noting that increasing PoPLar's depth-level results in a linear time increase. However, the increase in puzzle complexity, and therefore security, increases exponentially.

C. APP INTEGRATION CASE STUDY

One of the main design goals of our approach was ease of deployment. In this subsection, we outline the steps required for integrating PoPLar into a prototypical app that serves us as a case study, concretely, a popular Bitcoin Wallet.

1) ADDING DEPENDENCIES

Add the jitpack repository to the root `build.gradle` in the `repositories` list:

```
maven url 'https://jitpack.io'
```

and add the dependency:

TABLE 4. Usability scores.

Authentication mechanism	Mean	Median	SD
Fingerprint	8.55	9.00	1.78
PoPLar	8.38	9.00	1.62
Google Authenticator	8.21	9.00	1.69
SMS 2FA	7.48	8.00	2.01
Hardware authentication devices	6.98	7.00	2.37
Hardware 2FA tokens	6.56	6.50	2.36
Google's reCAPTCHA	6.09	6.00	2.62
Email 2FA	5.83	7.48	3.32

```
implementation 'com.github.AndrPoP:0.2'
```

under `dependencies`.

2) EVENT HANDLING

In the source code, identify the event handler which will trigger the PoPLar challenge (e.g. `onClick`). Define an identifier for the return code (e.g. `LAUNCH_SECOND_ACTIVITY`) Replace code (e.g. `doButtonStuff()`) in the event handler with the PoPLar challenge code. Refer to Listing 1 (top).

3) RESULT HANDLING

Handle the `onActivityResult` event and look for the PoPLar identifier return code: `LAUNCH_SECOND_ACTIVITY`. Ensure the result code is successful (i.e.,

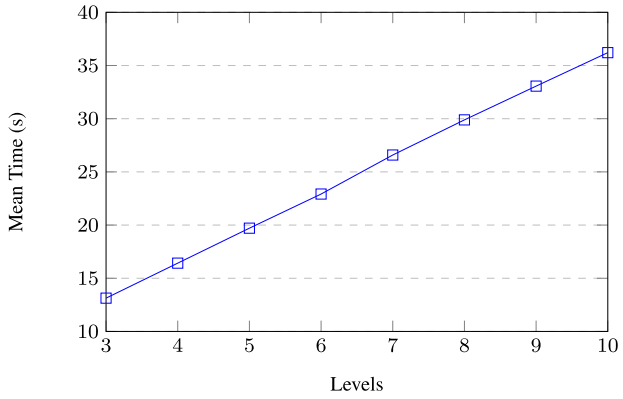

FIGURE 4. Mean authentication times for different PoPLar levels.

as returned by PoPL activity). Check that the PoPLar challenge was successful through intent data `PoPLPuzzle` and if so, execute the original event handler code (`doButtonStuff()`). Refer to Listing 1 (bottom).

D. DISCUSSION

While fingerprint authentication performed better than PoPLar in terms of usability, as stated in section III-C, the ideal solution also requires widespread availability of the underlying technology and must be secure in the context of the defined threat model. Our device survey 1 clearly shows that across Android devices, the accelerometer sensor is more widely available than the fingerprint sensor. Moreover, PoPLar was designed in such a way that a trojan would not be able to lure the user into solving the PoPLar challenge through a combination of overlays and social engineering. This is not true for biometric authentication, however, since a trojan could convince the user to perform a quick biometric authentication during an attack. For example, through Fingerprint-Jacking [32], or face ID spoofing [33], none of which require device rooting. Moreover, fingerprint authentication is also susceptible to replay and spoofing attacks, which is even made possible by simply obtaining the fingerprint image through a photo [34].

E. RELATED WORK

1) COMPARISON TO SENSOR-BASED CAPTCHAs

SenCAPTCHA [22] represents the most mature work concerning sensor-based CAPTCHAs. Considering that accessibility trojans can be seen as a form of (local) bots, sensor-based CAPTCHAs are the closest in concept to PoPLar and could also protect users from our proposed threat model. Yet, the main difference between mitigating bots and defending against malware is the need for practical security. SenCAPTCHA, through its picture puzzle-based approach, fails to satisfy this requirement by depending on the tagging of large volumes of images before it can be deemed sufficiently secure. Given that image tagging is a manual process and, at the same time, as their authors argue, difficult to automate for the task at hand, creating a repository of a sufficient size to offer a reasonable security level looks impractical.

```
private static  int LAUNCH_SECOND_ACTIVITY = 1;
public void onClick(View v){
  //doButtonStuff(); //old code
  //define start PoP intent
  Intent myint=new Intent(this,PoPPuzzleChallenge.class);
    //configure PoP challenge Depth
  myint.putExtra("PoPDepth",2);
  //Launch PoP intent
  startActivityForResult(myint, LAUNCH_SECOND_ACTIVITY);
}
@Override
protected void onActivityResult(int requestCode,
  int resultCode, Intent data) {
  super.onActivityResult(requestCode, resultCode, data);
    //check result code
  if (requestCode == LAUNCH_SECOND_ACTIVITY) {
    if(resultCode == Activity.RESULT_OK){
      //check for success or fail returned from
      Boolean result=
        data.getBooleanExtra("PoPPuzzle",false);
       //if success, continue with original handler code
      if(result){
        doButtonStuff();
      }
    }
    if (resultCode == Activity.RESULT_CANCELED) {
    }
  }
}
```

Listing 1. Event handling (top) and result handling code (bottom).

In contrast, PoPLar's security can be easily controlled via the setting of a certain dendrogram level depth, without requiring any additional manual work. In terms of timing, a low-level depth PoPLar challenge took an average of 13.1 seconds to complete, which is comparable to SenCaptcha's 4.1 - 11.5 seconds duration range. On the other hand, the results show that an increase in PoPLAr's depth-level results in a linear time increase with an exponential security increase. Our usability study, which tested PoPL for depths ranging from three to ten, confirmed that the participants were comfortable with PoPL's usability and timing. Also, contrary to SenCaptcha, PoPLar is production-ready and can generate as many distinct challenges as needed, and we believe this characteristic far outweighs the lengthier time requirements.

2) MACHINE LEARNING ATTACKS

Machine learning poses only a minimal threat to PoPLar, becoming significantly less of an issue than for other CAPTCHAs [35]. While image recognition can possibly recognise the next steps needed to solve the puzzle, effecting them requires programmatic access to the accelerometer sensor. In our threat model, this is not possible since the accessibility attack vector does not provide such access. On the other hand, this would have been possible had the malware included a rooting exploit [36] or if it was installed on an already rooted device. In this case the malware would be capable of injecting a mock sensor provider class inside the victim application or else write to its device file in `/dev/input/`. We do not see these threats as too concerning, especially considering that malware has free reign anyway on a rooted smartphone.

3) FINANCIAL FRAUD PREVENTION

Be it through Accessibility attacks or more traditional attacks such as phishing and social engineering, passwords and

common 2FA security measures are constantly under attack, and a successful compromise can easily lead to financial fraud. Researchers are always looking for innovative and secure alternatives or enhancements to existing authentication processes. One such example is Hacksaw [37], an authentication system using a wearable device that authenticates the user continually by correlating the input events with the user's corresponding hand movements captured via the device's motion sensors. While opting for a different approach, Hacksaw recognises that motion sensors can be used to determine the presence of the legitimate user. Similarly, other work investigates the use of alternative sources such as cellular infrastructure [38], location [39], and vibration [40], [41] data. An interesting authentication mechanism even proposes authenticating users based on their cognitive skills [42], in combination with sensor and touch interaction data collected while challenging the user to solve small games.

4) SYSTEM-LEVEL MITIGATION

While our work proposes a mitigation measure that can be implemented by non-system apps on non-rooted devices without the need for additional OS enhancements, an alternative approach could be to implement mitigation measures at system level. When presenting Cloak and Dagger [5], Fratantonio proposed a defense mechanism, implemented as an extension to the current Android API, which would protect Android apps from accessibility attacks. Similarly, one could take advantage of Android's TEE to protect against such attacks [43], [44].

V. CONCLUSION AND FUTURE WORKS

In this work, we addressed the threat of accessibility attacks, particulary its impact on current financial apps, putting victims at risk of financial loss. In this regard, we proposed Proof-of-Presence and Locality (PoPL), a new security mechanism that can act as a countermeasure and is based on the ability to prove the physical presence of a user, therefore blocking any automated actions that may be attempted by an accessibility trojan. This can also stop any kind of remote attack, even if executed live by a human, as this proof of presence includes a proof of locality as well. We released PoPLar, a reference open-source, free implementation of PoPL, with no requirements for additional hardware tokens.

Experimentation with three popular crypto exchange apps demonstrates superior attack resilience when compared to any combination of security measures currently deployed. While the use of our proposal results in longer authentication times, PoPLar is a production-ready solution with no further manual configuration required. Furthermore, a Bitcoin Wallet case study demonstrates the ease of integration with existing apps. In the future, we plan to develop a more accessible version that uses additional sensors, making it ready for users with any kind of impairments. The upgraded version will also be implemented for other mobile platforms. We urge Coinbase, Huobi and all other cryptocurrency exchanges, as well as classical banking apps to incorporate our inexpensive and easy to use solution to thwart accessibility and remote attacks.

REFERENCES

[1] J.-E. Ekberg, K. Kostiainen, and N. Asokan, "The untapped potential of trusted execution environments on mobile devices," *IEEE Secur. Privacy*, vol. 12, no. 4, pp. 29–37, Jul. 2014.

[2] L. H. Newman. (2020). *Flaws Could Have Exposed Cryptocurrency Exchanges to Hackers*. Accessed: Jan. 27, 2021. [Online]. Available: https://www.wired.com/story/cryptocurrency-exchanges-key-flaws-hackers/

[3] Y. Leguesse, M. Vella, C. Colombo, and J. Hernandez-Castro, "Reducing the forensic footprint with Android accessibility attacks," in *Proc. 16th Int. Workshop*, Guildford, U.K. Springer, 2020, pp. 22–38.

[4] Google. (2017). *Google Developer Training: Accessibility*. [Online]. Available: https://google-developer-training.github.io/android-developer-advanced-course-concepts/unit-3-make-your-apps-accessible/lesson-6-accessibility/6-1-c-accessibility/6-1-c-accessibility.html

[5] Y. Fratantonio, C. Qian, S. P. Chung, and W. Lee, "Cloak and dagger: From two permissions to complete control of the UI feedback loop," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2017, pp. 1041–1057.

[6] L. Stefanko. (2020). *Insidious Android Malware Gives Up All Malicious Features But One to Gain Stealth*. Accessed: Jan. 27, 2021. [Online]. Available: https://www.welivesecurity.com/2020/05/22/insidious-android-malware-gives-up-all-malicious-features-but-one-gain-stealth/

[7] Threat-Fabric. (2020). *2020—Year of the RAT*. Accessed: Jan. 27, 2021. [Online]. Available: https://www.threatfabric.com/blogs/2020_year_of_the_rat.html

[8] Y. Jang, C. Song, S. P. Chung, T. Wang, and W. Lee, "A11y attacks: Exploiting accessibility in operating systems," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2014, pp. 103–115.

[9] Binance. (2020). *How to Resume the Withdrawal Function*. Accessed: Jan. 27, 2021. [Online]. Available: https://www.binance.com/in/support/articles/360038583951-How-to-Resume-the-Withdrawal-Function

[10] A. Ometov, S. Bezzateev, N. Mäkitalo, S. Andreev, T. Mikkonen, and Y. Koucheryavy, "Multi-factor authentication: A survey," *Cryptography*, vol. 2, no. 1, p. 1, Jan. 2018.

[11] S. G. Lyastani, M. Schilling, M. Neumayr, M. Backes, and S. Bugiel, "Is FIDO2 the kingslayer of user authentication? A comparative usability study of FIDO2 passwordless authentication," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2020, pp. 268–285.

[12] M. Mohamed, B. Shrestha, and N. Saxena, "SMASheD: Sniffing and manipulating Android sensor data," in *Proc. 6th ACM Conf. Data Appl. Secur. Privacy*, Mar. 2016, pp. 152–159.

[13] A. A. Shaikh and H. Karjaluoto, "Mobile banking adoption: A literature review," *Telematics Informat.*, vol. 32, no. 1, pp. 129–142, Feb. 2015.

[14] S. Haig. (2020). *Five Mega Exchanges Hold 10% of Bitcoin's Entire Supply*. Accessed: Oct. 27, 2021. [Online]. Available: https://cointelegraph.com/news/five-mega-exchanges-hold-10-of-bitcoin-s-entire-supply

[15] K. Reese, T. Smith, J. Dutson, J. Armknecht, J. Cameron, and K. Seamons, "A usability study of five two-factor authentication methods," in *Proc. 15th Symp. Usable Privacy Secur. (SOUPS)*, 2019, pp. 357–370.

[16] J. Yan and A. S. E. Ahmad, "Usability of CAPTCHAs or usability issues in CAPTCHA design," in *Proc. 4th Symp. Usable Privacy Secur.*, 2008, pp. 44–52.

[17] F. Aloul, S. Zahidi, and W. El-Hajj, "Two factor authentication using mobile phones," in *Proc. IEEE/ACS Int. Conf. Comput. Syst. Appl.*, May 2009, pp. 641–644.

[18] S. Sundin. (2016). *2FA QR Code Generator*. Accessed: Jan. 27, 2021. [Online]. Available: https://stefansundin.github.io/2fa-qr/

[19] K. Krol, E. Philippou, E. De Cristofaro, and M. A. Sasse, "'They brought in the horrible key ring thing!' analysing the usability of two-factor authentication in U.K. online banking," 2015, *arXiv:1501.04434*.

[20] B. B. Zhu, J. Yan, G. Bao, M. Yang, and N. Xu, "Captcha as graphical passwords—A new security primitive based on hard AI problems," *IEEE Trans. Inf. Forensics Security*, vol. 9, no. 6, pp. 891–904, Jun. 2014.

[21] G. Reynaga and S. Chiasson, "The usability of CAPTCHAs on smartphones," in *Proc. Int. Conf. Secur. Cryptogr. (SECRYPT)*, 2013, pp. 1–8.

[22] Y. Feng, Q. Cao, H. Qi, and S. Ruoti, "SenCAPTCHA: A mobile-first CAPTCHA using orientation sensors," *Proc. ACM Interact., Mobile, Wearable Ubiquitous Technol.*, vol. 4, no. 2, pp. 1–26, Jun. 2020.

[23] T. Hupperich, K. Krombholz, and T. Holz, "Sensor CAPTCHAs: On the usability of instrumenting hardware sensors to prove liveliness," in *Proc. 9th Int. Conf.*, Vienna, Austria. Springer, Aug. 2016, pp. 40–59.
[24] Y. Chen, J. Sun, R. Zhang, and Y. Zhang, "Your song your way: Rhythm-based two-factor authentication for multi-touch mobile devices," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2015, pp. 2686–2694.
[25] A. Acien, A. Morales, J. Fierrez, R. Vera-Rodriguez, and O. Delgado-Mohatar, "BeCAPTCHA: Behavioral bot detection using touchscreen and mobile sensors benchmarked on HuMIdb," *Eng. Appl. Artif. Intell.*, vol. 98, Feb. 2021, Art. no. 104058.
[26] S. N. Alotaibi, S. Furnell, and N. Clarke, "A novel transparent user authentication approach for mobile applications," *Inf. Secur. J., Global Perspective*, vol. 27, nos. 5–6, pp. 292–305, Feb. 2018.
[27] K. Sun. (2019). *Google Play Apps Drop Anubis, Use Motion-Based Evasion*. Accessed: Jan. 28, 2021. [Online]. Available: https://www.trendmicro.com/en_us/research/19/a/google-play-apps-drop-anubis-banking-malware-use-motion-based-evasion-tactics.html
[28] Y. Leguesse, M. Vella, and J. Ellul, "AndroNeo: Hardening Android malware sandboxes by predicting evasion heuristics," in *Proc. Int. Conf. Inf. Secur. Theory Pract. (WISTP)*, Crete, Greece. Springer, 2017, pp. 140–152.
[29] E. Alepis and C. Patsakis, "Trapped by the UI: The Android case," in *Proc. 20th Int. Symp. Res. Attacks, Intrusions Defenses (RAID)*, Atlanta, GA, USA. Springer, 2017, pp. 334–354.
[30] S. Chen, T. Su, L. Fan, G. Meng, M. Xue, Y. Liu, and L. Xu, "Are mobile banking apps secure? What can be improved?" in *Proc. 26th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, Oct. 2018, pp. 797–802.
[31] L. Gomez, I. Neamtiu, T. Azim, and T. Millstein, "RERAN: Timing- and touch-sensitive record and replay for android," in *Proc. 35th Int. Conf. Softw. Eng. (ICSE)*, May 2013, pp. 72–81.
[32] X. Wang, Y. Chen, R. Yang, S. Shi, and W. C. Lau, "Fingerprint-jacking: Practical fingerprint authorization hijacking in Android apps," Blackhat, Europe, Tech. Rep. Blackhat 2020, 2020.
[33] Z. Yu, X. Li, X. Niu, J. Shi, and G. Zhao, "Face anti-spoofing with human material perception," in *Proc. 16th Eur. Conf. Comput. Vis. (ECCV)*, Glasgow, U.K. Springer, Aug. 2020, pp. 557–575.
[34] A. Taneja, A. Tayal, A. Malhorta, A. Sankaran, M. Vatsa, and R. Singh, "Fingerphoto spoofing in mobile devices: A preliminary study," in *Proc. IEEE 8th Int. Conf. Biometrics Theory, Appl. Syst. (BTAS)*, Sep. 2016, pp. 1–7.
[35] F. H. Alqahtani and F. A. Alsulaiman, "Is image-based CAPTCHA secure against attacks based on machine learning? An experimental study," *Comput. Secur.*, vol. 88, Jan. 2020, Art. no. 101635.
[36] W. Xu and Y. Fu, "Own your android! Yet another universal root," in *Proc. 9th USENIX Workshop Offensive Technol. (WOOT)*, 2015, pp. 1–6.
[37] P. Shrestha and N. Saxena, "Hacksaw: Biometric-free non-stop web authentication in an emerging world of wearables," in *Proc. 13th ACM Conf. Secur. Privacy Wireless Mobile Netw.*, Jul. 2020, pp. 13–24.
[38] F. S. Park, C. Gangakhedkar, and P. Traynor, "Leveraging cellular infrastructure to improve fraud prevention," in *Proc. Annu. Comput. Secur. Appl. Conf.*, Dec. 2009, pp. 350–359.
[39] C. Marforio, N. Karapanos, C. Soriente, K. Kostiainen, and S. Capkun, "Smartphones as practical and secure location verification tokens for payments," in *Proc. NDSS*, 2014, pp. 23–26.
[40] C. Wang, S. A. Anand, J. Liu, P. Walker, Y. Chen, and N. Saxena, "Defeating hidden audio channel attacks on voice assistants via audio-induced surface vibrations," in *Proc. 35th Annu. Comput. Secur. Appl. Conf.*, Dec. 2019, pp. 42–56.
[41] S. A. Anand and N. Saxena, "Coresident evil: Noisy vibrational pairing in the face of co-located acoustic eavesdropping," in *Proc. 10th ACM Conf. Secur. Privacy Wireless Mobile Netw.*, Jul. 2017, pp. 173–183.
[42] R. Spolaor, M. Monaro, P. Capuozzo, M. Baesso, M. Conti, L. Gamberini, and G. Sartori, "You are how you play: Authenticating mobile users via game playing," in *Proc. Int. Worskhop Commun. Secur. (WCS)*, Paris, France. Springer, 2017, pp. 79–96.
[43] W. Li, S. Luo, Z. Sun, Y. Xia, L. Lu, H. Chen, B. Zang, and H. Guan, "VButton: Practical attestation of user-driven operations in mobile apps," in *Proc. 16th Annu. Int. Conf. Mobile Syst., Appl., Services*, Jun. 2018, pp. 28–40.
[44] K. Ying, A. Ahlawat, B. Alsharifi, Y. Jiang, P. Thavai, and W. Du, "TruZ-Droid: Integrating TrustZone with mobile operating system," in *Proc. 16th Annu. Int. Conf. Mobile Syst., Appl., Services*, Jun. 2018, pp. 14–27.

YONAS LEGUESSE received the B.Sc. degree in mathematics and informatics and the M.Sc. degree (by research) in computer science & artificial intelligence from the University of Malta, in 2009 and 2018, respectively, where he is currently pursuing the Ph.D. degree in computer science. He has over 12 years experience in cybersecurity across various sectors, including national LEAs, EU agencies, the private sector, and academia. His interest in mobile security began during his time as an NIS expert at ENISA, the EU Cybersecurity Agency, where he was responsible for preparing and delivering mobile security and incident handling courses. His research interests include malware analysis, incident response, and memory forensics.

CHRISTIAN COLOMBO received the B.Sc., M.Sc., and Ph.D. degrees in computer science from the University of Malta, in 2007, 2009, and 2013, respectively. From 2008 to 2010, he worked as a Research Assistant on the nationally-funded project, Dependability and Error-Recovery in Security Intensive Financial Systems. Since 2010, he has been employed as an Academician with the Department of Computer Science, University of Malta. His research interests include runtime verification, software testing, compensating transactions, and domain-specific languages, with over 50 publications in these areas. He is currently focused on applying runtime verification in the area of cyber security through the funded projects: Secure Communication in the Quantum Era (NATO) and Lawful Evidence Collecting & Continuity Platform Development (Horizon 2020). He was a recipient of the MGSS Scholarship Scheme 2008.

MARK VELLA (Member, IEEE) received the M.Sc. degree in computer science from the University of Malta, and the Ph.D. degree in the area of computer systems security from the University of Strathclyde, U.K. He spent a number of years participating and leading enterprise application and integration projects before moving back to academia. He currently holds the position of a Senior Lecturer with the University of Malta. His initial research on developing intrusion detection techniques inspired by the workings of the human immune system, has today found home and immediate application within the context of using memory forensics for incident response, while keeping an eye on making computer systems less prone to security breaches. At University, he lectures and advises undergraduate and postgraduate students on topics of computer systems and security.

JULIO HERNANDEZ-CASTRO was with the University of Portsmouth, U.K., and Carlos III University, Spain. He is also affiliated with the Kent Cybersecurity Center. He is currently a Professor in computer security with the School of Computing, University of Kent. His research interests include wide, covering from RFID security to lightweight cryptography, including steganography and steganalysis and the design and analysis of CAPTCHAs. He has been a Predoctoral Marie Curie Fellow and also a Postdoctoral INRIA Fellow. He is currently the Vice-Chair of the EU COST Project CRYPTACUS. He receives research funding from Innovate U.K. Project aS, EPSRC Project 13375, and EU H2020 Project RAMSES.