PoPL: Proof-of-Presence and Locality, or how to secure financial transactions on your smartphone.

YONAS LEGUESSE¹, CHRISTIAN COLOMBO¹, MARK VELLA¹, AND JULIO HERNANDEZ-CASTRO ²

¹Department of Computer Science, University of Malta, Malta [yonas.leguesse.05, mark.vella, christian.colombo]@um.edu.mt
²School of Computing, Cornwallis South, University of Kent, UK jch27@kent.ac.uk

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

This work is supported by the 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].

Financial apps offer various security measures, for exam-
ple, 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 [11] 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:-

- **PoPLar**: a Proof-of-Presence and locality (PoPL) countermeasure against device input record/replay threats on mobile apps, based on mobile device sensors and a dendrogram-centric challenge.
- We demonstrate PoPLar’s effectiveness to mitigate accessibility attacks on three popular Android crypto exchange apps, without the need for additional hardware tokens, while ensuring usability.
- 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

Two-factor (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. 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 prac-
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. PoPLar 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.

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 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 PoPLar, the concept of a sensor-based challenge is taken to a level where it can also withstand accessibility attacks.

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 ticality/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 PoPLar.

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. PoPLar 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.

FIGURE 1. Successful accessibility trojan attack on crypto app (left) and an unsuccessful run blocked by PoPLar (right).
tools are often used for debugging, testing, and automation. The `adb` 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 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 `adb` process when it drops its capabilities to those associated with the `AID_SHELL` group id.

On the other hand, `input` 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 `adb` 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
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 programmatical. 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`. The assumption is that the attack targets a non-rooted 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 AppStore 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 PHONEDB website [4] which maintains the world’s largest and most detailed mobile device database. The `device spec`s 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

1https://toolsbox.getevent.c
2adb/daemon/main.cpp, private/android_filesystem_config.h
3cmds/input/src/com/android/commands/input/Input.java
4If the phone is rooted, the attacker would be able to bypass any controls in place.
5https://phonedb.net

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/
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.

### 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 | 9   | USE_BIOMETRIC       |
| Altimeter    | 0.8 | 14  | -                    |

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 PoPLar mechanism has the widest support possible, so the top four sensors all lend themselves well for building a PoPL mechanism. 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:

a: 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.

b: 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.

c: 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.

d: 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.

The approach adopted by SenCAPTCHA 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-B). 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 walkthrough 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.

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.
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 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. PoPLar is also available as a Jitpack library, allowing app developers to

---

7https://developer.android.com/reference/java/security/SecureRandom
8https://github.com/PoPDroid/AndroidPoP/tree/main/app/release
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 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. EXPERIMENT SETUP

In order to demonstrate PoPLar’s effectiveness in securing financial apps, we chose the Android ones for the top three cryptocurrency exchanges: Binance (v1.21), Coinbase (v6.55.0) and Huobi (v6.1.1). Between them, these apps have registered 16M+ installs on Google Play. 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. 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 AccessibilityTreeNodeInfo 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 identifier, 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. 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 prototyped 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 sendevent. This approach is similar to the one used by RERAN and SMAShedD, and once again assumes the installation phase described in the latter. This attack was prototyped with AndroidViewClient 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
TABLE 2. Device input record/replay attacks & key phases and steps

| Phase       | Steps                                                                 |
|-------------|------------------------------------------------------------------------|
| 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     | [Launch app] context.startActivity(new Intent.setComponent("com.app")). → |
| White-list  | [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) → |
| Get 2FA code| [Add address to white-list] getWidget(root, id).performAction(ACTION_SET_TEXT) → |
| Withdrawal  | [Save settings: Search for next node to click + click + repeat] getWidget(root, id).performAction(ACTION_CLICK) |
| Overlay     | [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) |
| Installation| [Install native service] adb push binsrvc /data/local/tmp → adb shell chmod u+x /data/local/tmp/binrvc → adb shell /data/local/tmp/binrvc & |
|             | [Start app] adb install app.apk → [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 →* |

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. The inability of the record/replay attack to bypass the PoPL challenge would result in a failed attack, as depicted in Figure 11.

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, 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 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.

b: Usability.
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:

c: 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.

d: 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)”. 

e: 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.

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.

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 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, or face ID spoofing, 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.

Figure 5 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

\[ \text{https://github.com/PoPDroid/AccAttacks} \]
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            |

Click Withdraw ➔ Steal Auth Codes ➔ Enter Stolen Codes & Confirm

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

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 two-factor authentication            | 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  |

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. 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.

D. 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.

a: Adding dependencies
Add the jitpack repository to the root build.gradle in the repositories list:

```
mvn {url 'https://jitpack.io'}
```

and add the dependency:

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

under dependencies.

b: 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).

c: 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., as returned by PoPLar 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).

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

// configure PoP challenge Depth
myint.putExtra("PoPDepth", 2);
// Launch PoP Intent
startActivityForResult(myint, LAUNCH_SECOND_ACTIVITY);
```
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 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.

c: 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, an authentication system using a wearable device that authenticates the user continuously 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, location, and vibration data. An interesting authentication mechanism even proposes authenticating users based on their cognitive skills in combination with sensor and touch interaction data collected while challenging the user to solve small games.

d: 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, 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.

V. CONCLUSIONS AND FUTURE WORKS

In this work, we addressed the threat of accessibility attacks, particularly its impact on current financial apps, putting victims at risk of financial loss. In this regard, we proposed 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] Jan-Erik Ekberg, Kari Kostiainen, and N Asokan. The untyped mapped trusted execution environments on mobile devices. IEEE Security & Privacy, 12(4):29–37, 2014.
[2] Lily Hay Newman. Flaws could have exposed cryptocurrency exchanges to hackers. [https://www.wired.com/story/cryptocurrency-exchanges-key-flaws-hackers/]. Accessed: 2021-01-27.
[3] Yonas Leguesse, Mark Vella, Christian Colombo, and Julio Hernandez-Castro. Reducing the forensic footprint with android accessibility attacks. In International Workshop on Security and Trust Management, pages 22–38. Springer, 2020.
[4] Google. Google developer training: Accessibility. [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]. Accessed: 2021-01-27.
[5] Yanick Fratantonio, Chenzhong Qian, Simon P Chung, and Wenke Lee. Cloak and dagger: from two permissions to complete control of the ui feedback loop. In 2017 IEEE Symposium on Security and Privacy (SP), pages 1041–1057. IEEE, 2017.
[6] Lukas Stefanko. Insidious Android malware gives up all malicious features but one to gain stealth. https://www.welivesecurity.com/2020/05/22/insidious-android-malware-gives-up-all-malicious-features-but-one-gain-stealth/, 2020. Accessed: 2021-01-27.
[7] Threat-Fabric. 2020 - Year of the RAT. [https://www.threatfabric.com/blogs/2020_year_of_the_rat.html]. Accessed: 2020-01-27.
[8] Yeojung Jing, Chengyu Song, Simon P Chung, Tielei Wang, and Wenke Lee. A11y attacks: Exploiting accessibility in operating systems. In Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, pages 103–115, 2014.
[9] Binance. How to resume the withdrawal function. [https://www.binance.com/in/support/articles/380038583951-How-to-Resume-the-Withdrawal-Function]. Accessed: 2021-01-27.
[10] Aleksandr Ometov, Sergey Bezzateev, Niko Mátkalo, Sergey Andreev, Tommi Mikkonen, and Yevgeni Koucheryavy. Multi-factor authentication: A survey. Cryptography, 2(1):1, 2018.
[11] Sanam Ghorbani Lyastani, Michael Schilling, Michaela Neumayr, Michael Backes, and Sven Bugiel. Is fido2 the kingslayer of user authentication? a comparative usability study of fido2 passwordless authentication. In 2020 IEEE Symposium on Security and Privacy (SP), pages 268–285. IEEE, 2020.
[12] Manar Mohamed, Babins Shrestha, and Nitesh Saxena. Smashed: Sniffting and manipulating android sensor data. In Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy, pages 152–159, 2016.
[13] Aijaz A Shaikh and Heikki Karjaluo. Mobile banking adoption: A literature review. Telematics and informatics, 32(1):129–142, 2015.

[14] Samuel Haag. Five mega exchanges hold 10% of Bitcoin’s entire supply. https://coingecko.com/news/five-mega-exchanges-hold-10-of-bitcoin-s-entire-supply/2020. Accessed: 2021-01-27.

[15] Ken Reese, Trevor Smith, Jonathan Dutson, Jonathan Armknecht, Jacob Cameron, and Kent Seaman. A usability study of five two-factor authentication methods. In Fifteenth Symposium on Usable Privacy and Security ([SOPS] 2019), 2019.

[16] Jeff Yan and Ahmad Salah El Ahmad. Usability of captchas or usability hijacking issues in captcha design. In Proceedings of the 4th symposium on Usable privacy and security, pages 44–52, 2008.

[17] Fadi Aloul, Syed Zahidi, and Wassim El-Hajj. Two factor authentication using mobile phones. In 2009 IEEE/ACS International Conference on Computer Systems and Applications, pages 641–644. IEEE, 2009.

[18] Stefan Sundin. 2fa qr code generator. https://stefansundin.github.io/2fa-qrcode/2016. Accessed: 2021-01-27.

[19] Kaj Riel, Eleni Philippou, Emiliano De Cristofaro, and M Angela Sasse. “they brought in the horrible key ring thing!” analysing the usability of two-factor authentication in uk online banking. arXiv preprint arXiv:1501.04434, 2015.

[20] Bin B Zhu, Jeff Yan, Guanbo Bao, Maowei Yang, and Ning Xu. Captcha as graphical passwords—a new security primitive based on hard on problems. IEEE transactions on information forensics and security, 9(6):891–904, 2014.

[21] Gerardo Reynaga and Sonia Chiasson. The usability of captchas on smartphones. In 2013 International Conference on Security and Cryptography (SECRYPT), pages 1–8. IEEE, 2013.

[22] Yunhe Feng, Qing Cao, Hairong Qi, and Scott Ruoti. SenCAPTCHA: A mobile-first CAPTCHA using orientation sensors. Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies, 4(2):1–26, 2020.

[23] Thomas Hupperich, Katharina Krombolz, and Thorsten Holz. Sensor captchas: On the usability of instrumenting hardware sensors to prove liveliness. In International Conference on Trust and Trustworthy Computing, pages 40–59. Springer, 2016.

[24] Yimin Chen, Jingchao Sun, Rui Zhang, and Yanchao Zhang. Your song your way: Rhythm-based two-factor authentication for multi-touch mobile devices. In 2015 IEEE conference on computer communications (INFOCOM), pages 2686–2694. IEEE, 2015.

[25] Alejandro Acien, Aythami Morales, Julian Fierez, Ruben Vera-Rodriguez, and Oscar Delgado-Mohatar. BeCAPTCHA: Behavioral bot detection using touchscreen and mobile sensors benchmarked on humidib. Engineering Applications of Artificial Intelligence, 98:104058, 2016.

[26] Saur Nejr Alotaibi, Steven Furnell, and Nathan Clarke. A novel transparent user authentication approach for mobile applications. Information Security Journal: A Global Perspective, 27(5-6):292–305, 2018.

[27] Kevin Sun. Google play apps drop anubis, use motion-based evasion. https://www.trendmicro.com/en_us/research/19/a/google-play-apps-drop-anubis-banking-malware-use-motion-based-evasion-tactics.html, 2019. Accessed: 2021-01-28.

[28] Onas Leguesse, Mark Vella, and Joshua Ellul. AndroNeo: Hardening Android malware sandboxes by predicting evasion heuristics. In IFIP International Conference on Information Security Theory and Practice, pages 140–152. Springer, 2017.

[29] Riccardo Spolaor, Myriam Monaro, Pasquale Capuozzo, Marco Baesso, Mauro Conti, Luciano Gambirani, and Giuseppe Sartori. You are how you play: Authenticating mobile users via game playing. In International Workshop on Communication Security, pages 79–96. Springer, 2017.

[30] Yonas Leguesse, Mark Vella, and Joshua Ellul. AndroNeo: Hardening Android malware sandboxes by predicting evasion heuristics. In IFIP International Conference on Information Security Theory and Practice, pages 140–152. Springer, 2017.

[31] Ethimios Alespis and Constantinos Pataskis. Trapped by the UI: The android droid case. In International Symposium on Retract in Attacks, Intrusions, and Defenses, pages 334–354. Springer, 2017.

[32] Sen Chen, Ting Su, Lingfan Guo, Guozhu Meng, Minhui Xue, Yang Liu, and Lihua Xu. Are mobile banking apps secure? what can be improved? In Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pages 797–802. 2018.

[33] Lorenzo Gomez, Iulian Neamtiu, Tanzirul Azim, and Todd Millstein. Reran: Timing-and-touch-sensitive record and replay for android. In 2013 35th International Conference on Software Engineering (ICSE), pages 72–81. IEEE, 2013.

[34] Xianbo Wang, Xikang Chen, Ronghui Yang, Shangcheng Shi, and Wing Cheong Lau. Fingerprint-jacking: Practical fingerprint authorization hijacking in android apps.

[35] Zifong Yu, Xiaobai Li, Xuesong Niu, Jingang Shi, and Guoying Zhao. Face anti-spoofing with human material perception. In European Conference on Computer Vision, pages 557–575. Springer, 2020.

[36] Archit Taneja, Aakrati Tayal, Aakash Malhotta, Anush Sankaran, Mayank Vatsa, and Rieha Singh. Fingerprint spoofing in mobile devices: a preliminary study. In 2016 IEEE 9th International Conference on Biometrics Theory, Applications and Systems (BTAS), pages 1–7. IEEE, 2016.

[37] Fatmeh H Alqahtani and Fawaz A Alsuailim. Is image-based captcha secure against attacks based on machine learning? an experimental study. Computers & Security, 88:101635, 2020.

[38] Wên Xu and Yuhin Fu. Own your android! yet another universal root. In 9th {USENIX} Workshop on Offensive Technologies ({WOOT} 15), 2015.

[39] Prakash Shrestha and Nitesh Saxena. Hacksw: biometric-free non-stop web authentication in an emerging world of wearables. In Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks, pages 13–24, 2020.

[40] Franc Pk Park, Chimmay Gangakbedkar, and Patrick Traynor. Leveraging cellular infrastructure to improve fraud prevention. In 2009 Annual Computer Security Applications Conference, pages 350–359. IEEE, 2009.

[41] Claudio Marforio, Nikolaos Karapanos, Claudio Soriente, Kari Kostiainen, and Sofjan Capkun. Smartphones as practical and secure location verification tokens for payments. In NDSS, volume 14, pages 23–26, 2014.

[42] Ken Reese, Trevor Smith, Jonathan Dutson, Jonathan Armknecht, Jacob Cameron, and Kent Seaman. A usability study of five two-factor authentication methods. In Fifteenth Symposium on Usable Privacy and Security ([SOPS] 2019), 2019.

[43] Fatmeh H Alqahtani and Fawaz A Alsuailim. Is image-based captcha secure against attacks based on machine learning? an experimental study. Computers & Security, 88:101635, 2020.

[44] Archit Taneja, Aakrati Tayal, Aakash Malhotta, Anush Sankaran, Mayank Vatsa, and Rieha Singh. Fingerprint spoofing in mobile devices: a preliminary study. In 2016 IEEE 9th International Conference on Biometrics Theory, Applications and Systems (BTAS), pages 1–7. IEEE, 2016.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/
Yonas Leguesse is a computer science Ph.D. student at the University of Malta. He received his B.Sc. in Mathematics and Informatics, and M.Sc. (By Research) in Computer Science & Artificial Intelligence from the University of Malta in 2009 and 2018 respectively. 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 his 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 academic within the Department of Computer Science at the 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). Dr. Christian Colombo was a recipient of the MGSS Scholarship Scheme 2008.

Mark Vella currently holds the position of Senior Lecturer at the University of Malta. He obtained an M.Sc in Computer Science from the University of Malta and spent a number of years participating and leading enterprise application and integration projects before moving back to academia. He pursued a research doctorate in the area of computer systems security at the University of Strathclyde (UK). 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 of computer security with the School of Computing, University of Kent. His research interests are wide, covering from RFID security to lightweight cryptography, including steganography and steganalysis and the design and analysis of CAPTCHAs. He has been a Pre-Doctoral Marie Curie Fellow and also a Post-Doctoral INRIA Fellow. He is currently the Vice-Chair of the EU COST Project CRYPTACUS. He receives research funding from InnovateUK Project aS, EPSRC Project 13375, and EU H2020 Project RAMSES.