The Journal of Systems and Software 224 (2025) 112364

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

A GUI-based Metamorphic Testing Technique for Detecting Authentication Vulnerabilities in Android Mobile Apps[☆]

Domenico Amalfitano[b,*], Misael Júnior[a], Anna Rita Fasolino[b], Marcio Delamaro[a]

[a] *Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, ICMC-USP, Brazil*
[b] *Department of Electrical Engineering and Information Technology, University of Naples Federico II, DIETI-UNINA, Italy*

ARTICLE INFO

Keywords:
Security testing
Metamorphic testing
Vulnerability testing
Mobile testing
Test automation
GUI-based testing

ABSTRACT

Context: The increasing use of mobile apps in daily life involves managing and sharing sensitive user information.
Problem: New vulnerabilities are frequently reported in bug tracking systems, highlighting the need for effective security testing processes for these applications.
Proposal: This study introduces a GUI-based Metamorphic Testing technique designed to detect five common real-world vulnerabilities related to username and password authentication methods in Android applications, as identified by OWASP.
Methods: We developed five Metamorphic Relationships to test for these vulnerabilities and implemented a Metamorphic Vulnerability Testing Environment to automate the technique. This environment facilitates the generation of *Source test case* and the automatic creation and execution of *Follow-up test case*.
Results: The technique was applied to 163 real-world Android applications, uncovering 159 vulnerabilities. Out of these, 108 apps exhibited at least one vulnerability. The vulnerabilities were validated through expert analysis conducted by three security professionals, who confirmed the issues by interacting directly with the app's graphical user interfaces (GUIs). Additionally, to assess the practical relevance of our approach, we engaged with 37 companies whose applications were identified as vulnerable. Nine companies confirmed the vulnerabilities, and 26 updated their apps to address the reported issues. Our findings also indicate a weak inverse correlation between user-perceived quality and vulnerabilities; even highly rated apps can harbor significant security flaws.

1. Introduction

Mobile devices have powerfully entered our daily life in a growing worldwide market where 7.516 billion smartphone users are expected by 2026 (O'Dea, 2020). Regarding mobile operating systems, Android is the most widely used, as it is installed on 70% of devices, while 27.7% are equipped with iOS (Laricchia, 2024). To understand the diffusion of mobile applications (a.k.a. apps) on Android devices, consider that over 109 billion apps were downloaded in 2023 from Google Play Store, the official app market for the Android operating system (Ceci, 2023). Mobile applications are integral to daily tasks, including entertainment (e.g., Netflix), games (e.g., Among Us), transport (e.g., Uber), socializing (e.g., Instagram, Facebook, Twitter, TikTok), education (e.g., Duolingo), and e-commerce (e.g., Amazon, eBay, Zalando).

The increase in the number of available apps is paralleled by the rise in vulnerabilities, as reported by Skybox Security in 2020 (Skybox Security, 2020): *"Vulnerabilities on mobile OSs increased by 50 percent, driven solely by Android flaws. This rise has come at the same time as home networks and personal devices increasingly intersect with corporate networks as a result of the move towards a mass, remote workforce. These trends should focus on the need for organizations to improve access controls and gain visibility of all ingress and egress points to their network infrastructure".* In such a scenario, mobile apps manage and share users' sensitive information (like family addresses, private contacts, phone numbers, emails, messages, and credit card credentials) that must be kept safe from malicious mobile attacks (Júnior et al., 2021).

Most of these attacks exploit vulnerabilities in mobile apps, such as unsecured sensitive data storage (OWASP, 2023c), unencrypted

☆ Editor: Yan Cai.
* Corresponding author.
E-mail addresses: domenico.amalfitano@unina.it (D. Amalfitano), misaeljr@usp.br (M. Júnior), fasolino@unina.it (A.R. Fasolino), delamaro@icmc.usp.br (M. Delamaro).

https://doi.org/10.1016/j.jss.2025.112364
Received 4 September 2024; Received in revised form 5 January 2025; Accepted 27 January 2025
Available online 7 February 2025

networking channels (OWASP, 2023b), and insecure authentication (OWASP, 2023a). A vulnerability is a specific type of fault related to security properties that can be exploited by an attacker (Felderer et al., 2016) and is a major cause of security and privacy breaches in commercial software. Vulnerabilities are often caused by software errors made by developers in their projects, as noted by Weir et al. (2020) from an analysis of bug tracking systems. While vulnerabilities are primarily introduced due to developers' lack of specific knowledge about software security issues (Ribeiro et al., 2018; Júnior et al., 2021), they are also difficult to detect for several reasons: *(i)* known testing techniques are complex and inefficient (Ribeiro et al., 2018; Weir et al., 2020; Júnior et al., 2021), *(ii)* apps are not tested against real-world known vulnerabilities (Júnior et al., 2021), and *(iii)* defining and implementing oracles to assess the absence of vulnerabilities is challenging (Ribeiro et al., 2018; Júnior et al., 2021).

Metamorphic Testing (MT), a testing technique that supports the creation of new test cases, is often adopted as a valid alternative to alleviate the oracle problem (Chen et al., 1998). The key element in its application is the definition of a set of Metamorphic Relationship (MR), i.e., relations derived from properties that must be satisfied by the Application Under Test (AUT) (Chen et al., 1998; Segura et al., 2016). Mai et al. (2020b,a) and Chaleshtari et al. (2023) successfully applied MT to find vulnerabilities in web systems. Heavily inspired by these results, we decided to define a MT technique for detecting vulnerabilities in Android mobile apps.

Among all security issues, authentication and authorization vulnerabilities rank second in the Open Web Application Security Project (OWASP) Top 10 vulnerabilities (OWASP, 2023d). We applied MT to detect authentication and authorization vulnerabilities in Android mobile apps related to username and password authentication methods. Our research direction is justified mainly by two reasons, namely, *(i)* username and password is one of the most common authentication methods provided by mobile apps (Lahav, 2016) and *(ii)* new vulnerabilities in username and password authentication methods are frequently discovered and reported by users and developers. Indeed, Common Weakness Enumeration (CWE) (CWE, 2023a) reported at least six types of vulnerabilities that affect those methods among the Top 25 Most Dangerous Software Weaknesses in 2021.[1] Moreover, in 2023, the OWASP (2023d) pointed out that among the "Top 10 Mobile Risks", three were related to those methods.[2]

Even though these vulnerabilities are generic and affect software applications regardless of their nature (desktop, mobile, or web), this paper presents a GUI-based MT technique for detecting weaknesses that may cause real vulnerabilities in username and password authentication methods in Android mobile apps. Our approach is based on deriving MRs from real-world vulnerabilities reported in OWASP and related weaknesses described in CWE. The MRs are defined as properties that must be satisfied by an Application Under Test (AUT) when it does not present a weakness described in CWE. A particular feature of our technique is that it defines MRs as AUT properties that can be checked through GUI-based testing.

To design the MT technique, we followed a process similar to the one adopted by Mai et al. (2020b,a) and Chaleshtari et al. (2023). While Chaleshtari et al. (2023) applied metamorphic testing to detect vulnerabilities in web systems using a comprehensive catalog of metamorphic relations (MRs), our work focuses on Android mobile apps and targets vulnerabilities that manifest during runtime interactions with graphical user interfaces (GUIs). Following this process, we defined five MRs formalizing the properties that a secure mobile app should provide to the user. These were implemented and tested using a capture-and-replay GUI-based technique supported by the Metamorphic Vulnerability Testing Environment we designed.

[1] See vulnerabilities ranked at (11, 14, 18, 20) in CWE (2021)

[2] See Insecure Authentication/Authorization (M3) and Insecure Communication (M5) in OWASP (2023d)

Static analysis approaches, such as the notable AUTHEXPLOIT (Ma et al., 2022), have also significantly advanced the detection of authentication flaws through dependency analysis and intermediate representations. While these methods excel in identifying vulnerabilities at the code level, our approach complements them by directly validating dynamic properties of Android apps during runtime, focusing on security flaws observable through GUI interactions.

The technique's effectiveness in detecting authentication vulnerabilities was tested on 163 popular Android apps, revealing that 66.3% (108/163) were affected by at least one vulnerability. This experimental process draws inspiration from the methodology employed by Ma et al. (2022), though with significant differences in scope and approach. While their study analyzed a larger dataset of 1200 Android applications using static analysis to detect authentication flaws at the code level, our approach leverages dynamic metamorphic testing to identify vulnerabilities that manifest during runtime through GUI interactions. Interestingly, one of our findings aligns with one of those of Ma et al. (2022), revealing that even highly rated and widely used apps are not immune to authentication vulnerabilities. Indeed, our analysis uncovered a weak inverse correlation between user-perceived quality and vulnerabilities: apps with higher ratings and downloads showed slightly fewer vulnerabilities, yet security risks persisted even in high-quality applications.

Results were validated through expert analysis, where three security professionals confirmed the identified vulnerabilities by interacting with the app's graphical user interfaces (GUIs). To assess the practical utility of our technique, we engaged with 37 companies whose applications were identified as vulnerable. Nine of these companies confirmed the reported vulnerabilities, and 26 updated their apps to address the issues, demonstrating real-world relevance.

The main contributions of this paper are as follows:

1 We introduce a novel, tool-supported approach for detecting vulnerabilities in Android mobile apps by combining MT with GUI-based testing through a capture-and-replay methodology. This approach is designed to be replicable and extendable, offering researchers and practitioners a practical framework for automated security testing.

2 We provide actionable insights into vulnerabilities emerging from GUI-level interactions by analyzing their prevalence and correlation with user-perceived quality metrics, such as app ratings and downloads. These findings help practitioners understand and address security risks specific to runtime and GUI behavior.

3 We validate our approach through a large-scale empirical study involving popular Android apps, demonstrating its effectiveness, and applicability to real-world scenarios. Additionally, we showcase the practical utility of our technique by presenting feedback from industry professionals, highlighting how several companies addressed the vulnerabilities reported by our method by updating their apps.

The paper is structured as follows: Section 2 introduces the theoretical background necessary to support this research; Section 3 presents the selected username and password authentication methods vulnerabilities and describes the process followed for the definition of the MRs for detecting them; Section 4 details the MT technique for the detection of selected vulnerabilities and the testing environment designed and implemented for automating the technique; Section 5 describes the structure of the experiment and provides a more in-depth discussion of the findings; Section 6 summarizes the threats to the validity of the study; Section 7 addresses some related work and the contributions to improvements in the state-of-the-art in security and MT on Android apps; finally, Section 8 is devoted to the conclusions and future research directions. Appendix includes a comprehensive list of acronyms used throughout the paper to aid the reader in understanding the terminology.

2. Background

This section introduces the reader to the key concepts and the terminology used in this study. The following sections provide background on metamorphic testing, security testing, and the username and password secure authentication method implemented by mobile apps.

2.1. Metamorphic testing

Metamorphic testing was introduced in 1998, in a technical report published by Chen et al. (1998), according to a survey conducted by Segura et al. (2016). Chen et al. (1998) claimed *"Metamorphic testing is a technique conceived to alleviate the oracle problem"*. Underlying the metamorphic testing technique is the idea it is often simpler to reason about existing relations between a program's outputs than to fully understand or formalize its input–output behavior. MT is a solution that can be highly automated towards easing both the oracles definition and execution in cases in which, in practical means, the correctness of outputs generated from valid input data is of difficult judgment (Weyuker, 1982). A classic application of MT involves testing a program designed to compute the sine function. This program can be effectively tested by leveraging the mathematical property of the sine function, which states that $sin(x)$ equals $sin(180 - x)$ for any numerical value of x. By randomly sampling values for x, each value is automatically paired with its corresponding $180 - x$, allowing verification of whether the mathematical property holds true for the sine function. This is an example of Metamorphic Testing (MT) consisting of (1) an input transformation that can be used to generate new test cases from existing test data and (2) an output relationship for comparing the outputs produced by a pair of test cases.

Segura et al. (2016) described the general MT process for testing a program `P` implementing a function `f` and an example of how the process can be applied (Segura and Zhou, 2018).

The generic MT process consists of the following four steps:

(i) identification of the properties of `f` and their representation by MRs;
(ii) generation or selection of a test suite, composed by *Source test cases (STC)*, I_1, which can be defined by traditional testing techniques such as random testing or model-based;
(iii) use of MRs for generating a new test suite, I_2, made by *Follow-up test cases (FUTC)*;
(iv) execution of I_1 and I_2, and check of whether respective outputs O_1 and O_2 violate or not MRs. If an MR is violated for a couple of base and follow-up test cases, then `P` contains errors.

A well-known example of the application of such a process is the testing of a program `P` implementing a function f that calculates the length of the shortest path on a graph `G`, between an origin vertex `O` and a destination vertex `D`. Said `|SP(G, O, D)|` such distance, an MR for `f` can be defined as *if the origin and destination vertexes are exchanged, then the size of the shortest path must be the same, i.e.,* `|SP(G, O, D)| == |SP(G, D, O)|`. If the aim is to apply MT for verifying whether `f` performs correctly on very large graphs, then `|SP(G, O, D)|` can be defined as a source test case, `|SP(G, D, O)|` can be described as the Follow-up test case, and MR `|SP(G, O, D)| == |SP(G, D, O)|` can be checked on whether it is not violated for any couple of origin and destination vertexes of `G`. As an example, such vertexes could be selected through a random testing strategy.

2.2. Security testing

According to the *System and Software quality models defined in the ISO/IEC 25000 SQUARE series* (ISO, 2011), the security requirement, a.k.a. "security characteristic", represents *the degree to which a software product or a software system protects information and data so that users or other systems have access to data appropriate to their types and levels of authorization*. Security testing is the process of assessing whether the features of a software implementation are consistent with the expected security requirements (Felderer et al., 2016).

The literature reports two main security testing approaches, namely, *security functional testing* and *security vulnerability testing* (Tian-yang et al., 2010). The former aims to validate whether specified security requirements have been correctly implemented regarding both security properties and security mechanisms. On the other hand, security vulnerability testing aims to identify unintended vulnerabilities in AUT.

A vulnerability is an error in the system design, implementation, operation, or management (Tian-yang et al., 2010; Potter and McGraw, 2004). Attackers usually exploit vulnerabilities to perform improper actions in the system, e.g., collecting sensible users' data. Security vulnerability testing can identify vulnerabilities and prevent them from being exploited by malicious attackers (Potter and McGraw, 2004).

The literature describes two methodologies for security vulnerability testing: (i) simulating attacks or conducting penetration tests to compromise the system's security (Arkin et al., 2005), and (ii) identifying system risks and generating tests to uncover vulnerabilities associated with those risks (Potter and McGraw, 2004).

In this paper, we focus on the latter methodology in the context of *security vulnerability testing* of mobile apps implementing user and password authentication methods.

2.3. Secure username and password authentication in mobile apps

OAuth 2.0 (OAuth, 2012) is the de-facto standard authorization protocol designed to allow applications to access Protected Resources (PR) on behalf of users. It defines an abstract protocol flow (D. H, 2012) that client applications, including mobile apps, should follow to obtain access. Fig. 1 illustrates the typical process for client apps on Android Device.

Initially, the client app sends an Authentication Request (AR) to the user, who responds with an Authorization Grant (AG). Among the four grant types defined by OAuth 2.0, User and Password Credentials (UPC) is the most commonly implemented (Ma et al., 2019, 2020). Typically, the app presents a Sign-In Screen (SIS) where the user enters their username and password before proceeding. The client forwards the AG to an Authorization Server (Auth-Server), which validates the credentials. If successful, the client receives an Access Token (AT), which it uses to request PR from the Resource Server (Res-Server). A well-known example is the Netflix app, which requires user authorization through credential entry before granting access to its catalog. To ensure secure implementation, OWASP (OWASP, 2001) provides guidelines in the Mobile and Web Application Security Testing Guides (Mueller et al., 2023; Saad and Mitchell, 2019). Key recommendations include:
(i) SSL certificate verification: The app must verify SSL/TSL certificates to prevent Man-in-the-Middle (MITM) attacks (Wei and Wolf, 2017; Wang et al., 2020). Valid certificates, issued by trusted Certificate Authorities, ensure encryption of client–server communications.
(ii) Encrypted data transmission: OWASP advises using HTTPS Connection Channel (HTTPS-CC) instead of HTTP Connection Channel (HTTP-CC) to protect sensitive data from interception.
(iii) Access token management: Access token lifetime must be properly handled to prevent session hijacking (Mueller et al., 2023). Tokens should expire after logout and have limited lifetimes, with high-risk apps often using shorter expiration times (2–5 min).
(iv) Session verification: Access to sensitive functionalities should be restricted to authenticated users. Res-Server must verify user authentication with each PR request.

Non-compliance with these guidelines may introduce weaknesses that could lead to authentication vulnerabilities, potentially exploitable by malicious attackers. The following sections describe the activities undertaken to define and experiment with the MT technique for detecting authentication vulnerabilities in Android mobile apps.

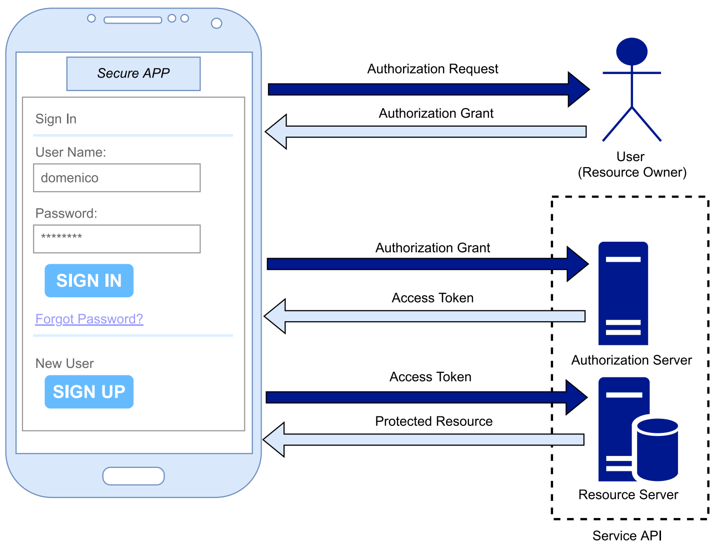


Fig. 1. Abstract protocol flow of user authorization and access to PR defined by OAuth 2.0.

Table 1
Weaknesses related to vulnerabilities due to improper design and implementation of username and password authentication methods.

CWE ID	CWE name	Weakness description	OWASP vulnerability category
CWE-295 (CWE, 2023c)	Improper Certificate Validation	A mobile app is potentially vulnerable if the Authentication Server accepts the user authentication request even if communication with the client is not established over an SSL/TSL secure channel.	M5: Insecure Communication (OWASP, 2023b)
CWE-613 (CWE, 2023f)	Insufficient Session Expiration	A mobile app is potentially vulnerable if the AT is not refreshed or destroyed after a long user's inactivity time.	M5: Insecure Communication (OWASP, 2023b)
CWE-384 (CWE, 2023e)	Session Fixation	A mobile app is potentially vulnerable if the AT is not destroyed after the user has signed out.	M5: Insecure Communication (OWASP, 2023b)
CWE-311 (CWE, 2023d)	Missing Encryption of Sensitive Data	A mobile app is potentially vulnerable if the Authentication Server accepts the user authentication request even if communication with the client is not established through an HTTPS-CC protocol.	M5: Insecure Communication (OWASP, 2023b)
CWE-288 (CWE, 2023b)	Authentication Bypass Using an Alternate Path or Channel	A mobile app is potentially vulnerable if it does not limit the access to functionalities and screens, collecting sensible data, for only authenticated and authorized users.	M3: Insecure Authentication/Authorization (OWASP, 2023a)

3. Metamorphic relationships for detecting authentication vulnerabilities

This section outlines the process we employed to define the MRs for detecting vulnerabilities associated with username and password authentication methods. It relied on the execution of two steps, i.e., *Vulnerabilities selection and weaknesses description* and *Metamorphic Relationships definition* that are described in the following.

3.1. Vulnerabilities selection and weaknesses description

In this step, we identified weaknesses corresponding to two categories of real-world vulnerabilities as documented by OWASP: *M5: Insecure Communication* (OWASP, 2023b) and *M3: Insecure Authentication/Authorization* (OWASP, 2023a). CWE defines weaknesses as errors that can result in vulnerabilities. As OWASP correlates each vulnerability with a set of weaknesses outlined in CWE, we manually examined weaknesses linked to the two aforementioned vulnerability categories. From these, we selected those that could arise if developers fail to adhere to the guidelines outlined in OAuth 2.0, as discussed in Section 2.3, for implementing a secure Abstract Protocol Flow.

Table 1 provides a summary of the selected weaknesses. For each weakness, it includes the CWE ID, CWE Name, a textual description, and the corresponding OWASP vulnerability category.

3.2. Metamorphic Relationships definition

In this step, we manually defined one or more MRs for each weakness listed in Table 1, presenting their corresponding IDs and formalizations in Table 2. While techniques for automatically generating MRs exist (Ayerdi et al., 2024), they are not yet applicable to GUI-based testing. Consequently, the MRs in this paper were manually specified using the template proposed by Segura et al. (2017a). This flexible template, inspired by widely adopted formats in software engineering, facilitates the structured description of MRs through both natural and formal language.

MR_1 - improper certificate validation (CWE-295). This MR verifies whether the Authorization Server rejects authentication when the device uses No Trusted SSL/TLS Digital Certificate or a Self Signed Digital Certificate for encryption. The authentication function $GUI = Auth(AndDev, UPC, AUT, certificate)$ returns the GUI when the Resource Owner authenticates through the Sign-In Screen of the Application Under Test using a specific certificate.

MR_2 - insufficient session expiration (CWE-613). This MR checks if the Access Token refreshes after one hour of inactivity. The token identifier at t_1 ($idAT(t_1)$) must differ from the one at $t_1 + \Delta$ ($idAT(t_1 + \Delta)$), where $\Delta \geq 60$ minutes. Functions $GUIInteract(AUT, RO, t_i, t_f)$ and $GUISleep(AUT, RO, t_i, t_f)$ indicate user interaction or inactivity during specific periods.

Table 2
Formalization of the five MRs introduced for each weakness listed in Table 1.

CWE ID	MR ID	MR Formalization
CWE-295	MR_1	In the domain of ***Android applications***
		The following metamorphic relation should hold MR_1
		If
		$GUI_v = Auth(AndDev, UPC, AUT, TSSLDC)$
		and
		$GUI_{nv} = Auth(AndDev, UPC, AUT, NTSSLDC)$ or
		$GUI_{nv} = Auth(AndDev, UPC, AUT, SSDC)$
		then $GUI_v \neq GUI_{nv}$
CWE-613	MR_2	In the domain of ***Android applications***
		assuming that the *RO* has authenticated at time t_0
		The following metamorphic relation should hold MR_2
		If
		$GUIInteract(AUT, RO, t_0, t_1)$ and
		$GUISleep(AUT, RO, t_1, t_1 + \Delta)$ $(\Delta > 60$ min)
		then $IdAT(t_1 + \Delta) \neq IdAT(t_1)$.
CWE-384	MR_3	In the domain of ***Android applications***
		assuming that the *RO* has authenticated at time t_0
		The following metamorphic relation should hold MR_3
		If
		$SignOut(AUT, RO, t_1)$ and
		$SignIn(AUT, RO, UPC, t_1 + \delta)$ $(\delta \approx 2$ min)
		then $IdAT(t_1 + \delta) \neq IdAT(t_1)$.
CWE-311	MR_4	In the domain of ***Android applications***
		The following metamorphic relation(s) should hold MR_4
		If
		$GUI_v = Auth(AndDev, UPC, AUT, HTTPS-CC)$ and
		$GUI_{nv} = Auth(AndDev, UPC, AUT, HTTP-CC)$
		then $GUI_v \neq GUI_{nv}$
CWE-288	MR_5	In the domain of ***Android applications***
		The following metamorphic relation should hold MR_5
		If
		$GUI_{RO-PR} = \{GUI : GUI\ accesses\ to\ PR\ of\ RO\}$ and
		$GUI_{NAAU} = \{GUI : GUI\ accessed\ by\ NAAU\}$
		then $GUI_{RO-PR} \bigcap GUI_{NAAU} = \emptyset$

MR_3 - session fixation (CWE-384). This MR ensures a new AT is generated after logout. The token identifier at logout ($idAT(t_1)$) must differ from the token at the next login ($idAT(t_2)$), where $\delta \approx 2$ minutes. Functions $SignIn(AUT, RO, UPC, t)$ and $SignOut(AUT, RO, t)$ represent user sign-in and sign-out actions.

MR_4 - missing encryption of sensitive data (CWE-311). This MR checks if the Authorization Server rejects authentication when HTTPS Connection Channel (HTTPS-CC) is not used for transmitting UPC. The function $GUI = Auth(AndDev, UPC, AUT, channel)$ returns the GUI when the Resource Owner authenticates through a specific channel.

MR_5 - authentication bypass (CWE-288). This MR ensures the AUT does not allow unauthenticated users to access PR GUIs. Two GUI sets are defined: GUI_{RO-PR} (accessible to authenticated users) and GUI_{NAAU} (accessible to unauthenticated users). This MR verifies that the two sets do not overlap.

3.2.1. Baseline comparison

To provide additional context, we compared our MRs with the ones proposed in Chaleshtari et al. (2023) for web systems, identifying overlaps and distinctions. While their approach emphasizes static and hybrid analyses for web traffic, our MRs target runtime GUI-based interactions in Android mobile apps. It is important to note that the relationship between the MRs used in this work and those proposed in Chaleshtari et al. (2023) is conceptual. Directly applying MRs from one domain, such as web systems, to another, like Android apps, is not feasible without adaptation. Each MR must be tailored to the specific characteristics and challenges of the target domain. Below, we specify whether each MR has a total, partial, or no overlap with the ones proposed in Chaleshtari et al. (2023).

MR_1. This relationship addresses SSL/TLS misconfigurations, such as trusting invalid certificates, which are directly aligned with MRs targeting outdated or improperly validated certificates (e.g., CWE-298, CWE-599) addressed in Chaleshtari et al. (2023). Our contribution extends these checks to runtime GUI interactions specific to Android apps. *Overlap:* ***Total***

MR_2. This relationship focuses on ensuring access tokens are refreshed after prolonged inactivity, aligning partially with the MRs, defined in Chaleshtari et al. (2023), addressing session expiration issues (e.g., CWE-613). The prior work concentrates on session-related vulnerabilities in web systems, whereas our rule captures mobile-specific usage scenarios. *Overlap:* ***Partial***

MR_3. This relationship ensures token regeneration upon re-authentication, addressing session fixation vulnerabilities. While similar MRs exist (e.g., CWE-384, CWE-610), they are tailored in Chaleshtari et al. (2023) for cookie-based authentication in web systems, whereas our approach targets token-based mechanisms in mobile apps. *Overlap:* ***Partial***

MR_4. This relationship checks for secure transmission of sensitive credentials, directly overlapping with MRs targeting unencrypted data transmission (e.g., CWE-312, CWE-315) proposed in Chaleshtari et al. (2023). The distinction lies in our focus on Android-specific communication channels. *Overlap:* ***Total***

MR_5. This relationship ensures that unauthenticated users cannot access restricted GUI elements. While Chaleshtari et al. (2023) includes MRs addressing authorization vulnerabilities (e.g., CWE-288, CWE-863), these do not explicitly target GUI-specific bypass scenarios in mobile apps. *Overlap:* ***None***

4. The proposed testing technique

This section describes the technique we defined to detect vulnerabilities in Android mobile apps. The proposed technique, illustrated in Fig. 2, integrates GUI-based testing, capture and replay testing, and metamorphic testing within a scenario-based framework.

The technique is GUI-based as test cases consist of user event sequences on the GUI, with test oracles implemented through GUI assertions. It is metamorphic because the Metamorphic Relationships in Section 3 automatically translate *Source test cases* into *Follow-up test cases*. Capture and replay testing is incorporated by capturing manual tester interactions for *STCs*, which are subsequently replayed to execute *FUTCs*.

The manual interaction follows a predefined scenario, facilitating the automated generation of executable *Follow-up* test cases. As shown in the figure, the technique consists of three main steps: **"App exploration"**, **"*Follow-up test cases generation*"**, and **"*Follow-up test cases execution*"**. The figure also highlights the required inputs, produced outputs, and the supporting tools for each step. A detailed description of each phase follows.

App Exploration. step generates a single *Source test case* through manual tester interaction with the GUI, covering a defined usage scenario, as shown in Fig. 3, which illustrates the Booking app. During interaction, user events and GUI descriptions are captured in the *STC*. GUIs are described by their widgets and attributes (Amalfitano et al., 2018). The scenario involves four key activities, essential for creating *Follow-up* test cases by applying MRs. These activities and their rationales are detailed below.

- Device setup preconditions - AUT launch: The Android Device must be configured by (1) installing a TSSLDC and (2) routing communication with Auth-Server and Res-Server through a controlled proxy. Finally, AUT is launched.
  Rationale: This ensures compliance with OAuth 2.0, encrypting traffic with TSSLDC and routing over HTTPS Connection Channel. The proxy enables monitoring of network traffic to detect vulnerabilities.

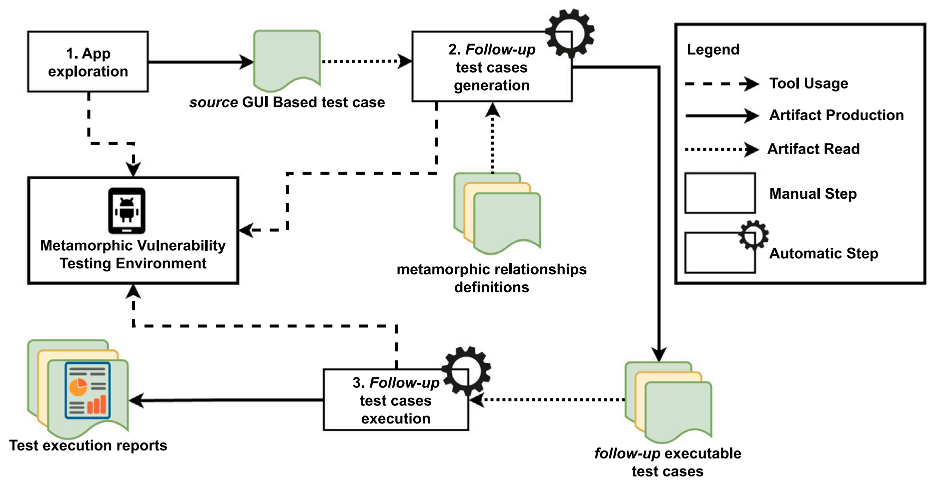

1. App exploration
source GUI Based test case
2. Follow-up test cases generation
Metamorphic Vulnerability Testing Environment
metamorphic relationships definitions
Test execution reports
3. Follow-up test cases execution
follow-up executable test cases
Legend
Tool Usage
Artifact Production
Artifact Read
Manual Step
Automatic Step

Fig. 2. The proposed metamorphic-based vulnerability testing technique.

- User authentication: The tester (Resource Owner) authenticates with valid User and Password Credentials through the Sign-In Screen of AUT, rendering a `Welcome` GUI.
  Rationale: Supports MR_1 and MR_4, verifying authentication failure if TSSLDC is missing or if credentials are not transmitted over HTTPS-CC.
- Access to Protected Resources: The RO navigates the GUI to access PR.
  Rationale: Essential for MR_2 and MR_5. MR_2 checks if Access Token refreshes periodically, while MR_5 ensures unauthorized users cannot access PR.
- User sign out - reauthentication: The RO signs out and re-authenticates.
  Rationale: Supports MR_3, ensuring Access Token is destroyed at sign-out and regenerated at re-authentication.

"Follow-up test case s generation". This step automatically generates *FUTCs* by applying the MRs from Section 3.2 to the GUI-based *STCs* produced in the previous phase. Six *FUTCs* are generated, despite the definition of only five MRs. Specifically, two test cases are derived from MR_1: one verifies authentication refusal when NTSSLDCs are installed, and the other when SSDCs are used. *Follow-up test cases* replay all or part of the four activities recorded during **App Exploration**. They can include commands to collect ATs or trigger random GUI events over a defined period. Preconditions may mirror those from **App Exploration** or differ to simulate vulnerabilities in specific configurations.

Table 3 outlines the design of each *FUTC*, describing preconditions, activity sequences, and test oracles for pass/fail determination. A detailed summary follows.

$FUTC_1$ Supports MR_1 by replaying the User authentication step with valid User and Password Credentials on a device equipped only with NTSSLDC. The resulting GUI is compared with that from **App Exploration**. A mismatch indicates authentication refusal.

$FUTC_2$ An alternative MR_1 test case, identical to $FUTC_1$, but uses SSDC instead of NTSSLDC as preconditions.

$FUTC_3$ Relates to MR_2. After User authentication, the initial AT (AT1) is collected. One hour of GUI navigation follows, after which AT2 is obtained. The test verifies that the two tokens differ, ensuring session expiration functions correctly.

$FUTC_4$ Designed for MR_3, this test case replicates the **App Exploration** process, with token collection after User Sign Out and subsequent User Authentication. The two tokens must differ, indicating session termination.

$FUTC_5$ Supports MR_4, mirroring $FUTC_1$ and $FUTC_2$ but redirecting traffic over HTTP Connection Channel to test for vulnerabilities caused by unencrypted communication.

$FUTC_6$ Applies MR_5. The test replays User Authentication with incorrect credentials, followed by the Access to PR step to verify that access to Protected Resources is denied.

"Follow-up test cases execution". This step simply executes the *follow-up* test cases, produced in the previous step, on the Android Device.

4.1. The metamorphic vulnerability testing environment

This section describes the *Metamorphic Vulnerability Testing Environment*, designed and implemented to support the execution of the proposed metamorphic-based vulnerability testing technique.

4.1.1. Architectural overview

Fig. 4 presents the architectural diagram of the testing environment, highlighting its core components. One key component is the *Android Device*, which can be either a *Real* or *Virtual Device* with the AUT preinstalled and preconditions set. The tester manually interacts with the device's GUI to execute the scenario shown in Fig. 3.

The *Capture* component records GUI interactions using Android's `getevent` (Android, 2023a) and `uiautomator` (Android, 2023b) tools. `getevent` streams kernel-level input events triggered during app exploration, detailing timestamps, device names, event types, and screen coordinates. `uiautomator` generates XML dumps describing GUI widgets and attributes. The *Capture* component produces a GUI-based *source* test case in `JUnit` (Gamma and Beck, 2017).

The *follow-up generator* creates a `JUnit` test suite from the *Source test case*, producing six *Follow-up test cases* that implement the testing logic. This component modifies initial preconditions by removing existing certificates and installing either NTSSLDC or SSDC. It translates user event sequences captured during App Exploration into a replayable format using a custom solution based on the `sendevent` tool, as detailed in Amalfitano et al. (2019).

Additionally, the generator integrates the Monkey tool to conduct 60 minutes of random GUI exploration (Android Developers, 2024). It converts GUI descriptions into oracles by embedding assertions directly into the test cases. Finally, it inserts commands to collect Access Tokens, which are critical for further analysis.

The *Android Device* is also responsible for executing the *Follow-up test cases* using the *Replay* component. This component employs the `sendevent` tool to simulate GUI events on the AUT and uses `uiau-`

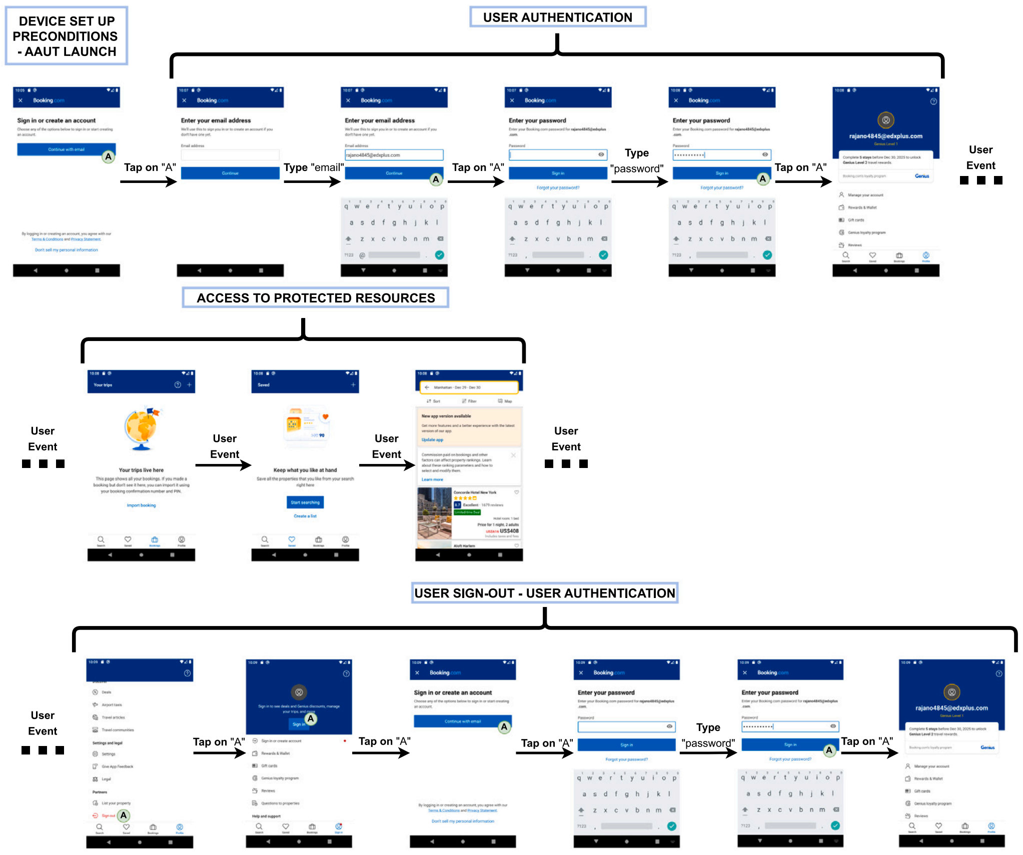


Fig. 3. Example of *source* GUI based test case scenario performed on the real Booking application.

Table 3
Design of the six test cases related to the defined Metamorphic Relationships.

MR	*Follow-up test case*	Preconditions	Test steps	Test oracle
MR_1	*$FUTC_1$*	Only NTSSLDCs are installed on the AndDev	**REPLAY** [User authentication (correct UPC)]	The resulting GUI has not be the one collected during the **App exploration** step, i.e., the authentication must be refused
	$FUTC_2$	Only SSDCs are installed on the AndDev	**REPLAY** [User authentication (correct UPC)]	The resulting GUI has not be the one collected during the **App exploration** step, i.e., the authentication must be refused
MR_2	*$FUTC_3$*	Same preconditions used in the **App exploration** step	**REPLAY** [User authentication (correct UPC)] - `Collect AT1` - 60 min of GUI Random Exploration - `Collect AT2`	`AT1.Id` ≠ `AT2.Id`
MR_3	*$FUTC_4$*	Same preconditions used in the **App exploration** step	**REPLAY** [User authentication (correct UPC)] - **REPLAY** [Access to PR] - **REPLAY** [User sign out] - `Collect AT1` - **REPLAY** [User authentication (correct UPC)] - `Collect AT2`	`AT1.Id` ≠ `AT2.Id`
MR_4	*$FUTC_5$*	The `mitmproxy` is configured to redirect the network traffic over a HTTP Connection Channel	**REPLAY** [User authentication (correct UPC)]	The resulting GUI has not be the one collected during the **App exploration** step, i.e., the authentication must be refused
MR_5	*$FUTC_6$*	Same preconditions used in the **App exploration** step	**REPLAY** [User authentication (incorrect UPC)] - **REPLAY** [Access to Protected Resources]	The resulting GUIs have not be the ones collected during the **App exploration** step, i.e., the Protected Resources are not reachable through the graphical user interface.

`tomator` to obtain the current GUI description in XML format. The XML description is evaluated against test oracles to verify whether the encountered GUI matches the expected result. Fig. 5 shows an example from the Booking app, where the left image displays a snapshot of the GUI indicating denied user authentication. This represents the expected outcome for MR_1 and MR_4, requiring a mismatch between the obtained GUI and the one recorded during app exploration (right image). The figure also includes an excerpt of the XML description for the left snapshot, highlighting the widget responsible for displaying the message *"Something went wrong"* during authentication. If the application is vulnerable under MR_1 or MR_4, successful user authentication would yield the same GUI as during app exploration, resulting in identical XML descriptions. This allows the oracle to detect the vulnerability. The ability to capture and replay events at the kernel level enables

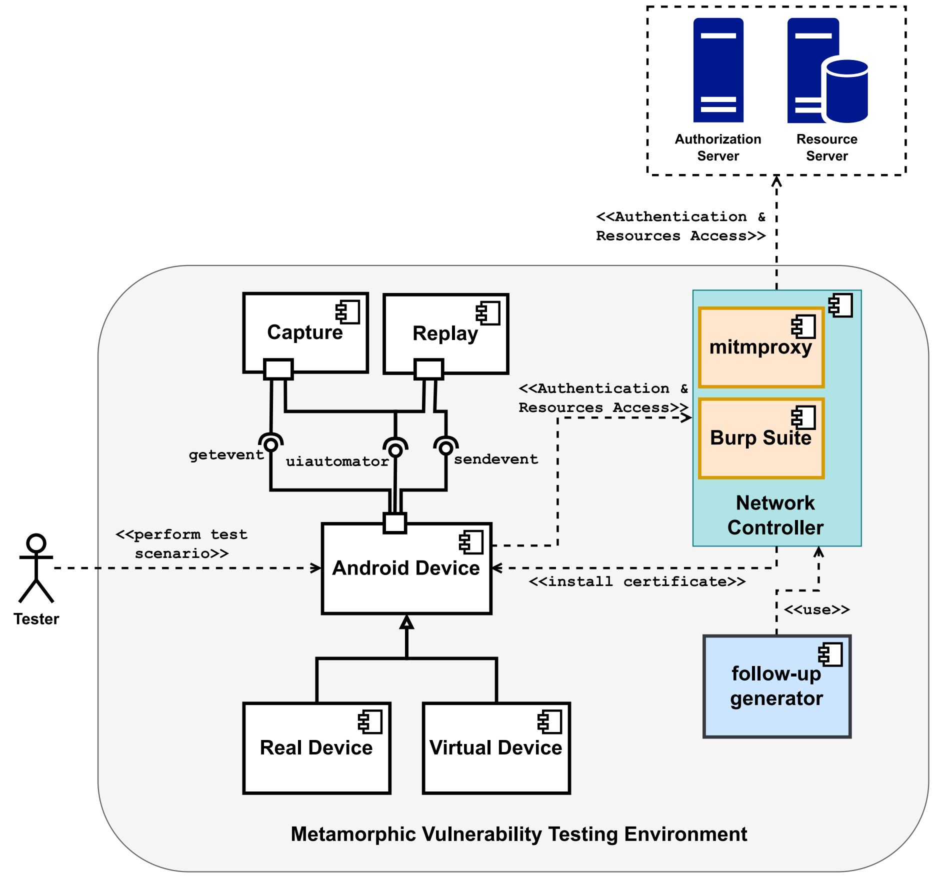
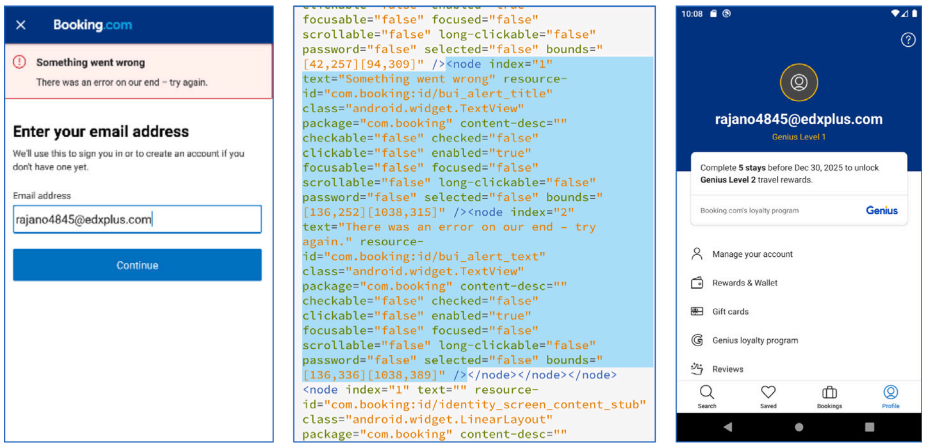


Fig. 4. Metamorphic vulnerability testing environment architecture.



```
focusable="false" focused="false"
scrollable="false" long-clickable="false"
password="false" selected="false" bounds="
[42,257][94,309]" /><node index="1"
text="Something went wrong" resource-
id="com.booking:id/bui_alert_title"
class="android.widget.TextView"
package="com.booking" content-desc=""
checkable="false" checked="false"
clickable="false" enabled="true"
focusable="false" focused="false"
scrollable="false" long-clickable="false"
password="false" selected="false" bounds="
[136,252][1038,315]" /><node index="2"
text="There was an error on our end – try
again." resource-
id="com.booking:id/bui_alert_text"
class="android.widget.TextView"
package="com.booking" content-desc=""
checkable="false" checked="false"
clickable="false" enabled="true"
focusable="false" focused="false"
scrollable="false" long-clickable="false"
password="false" selected="false" bounds="
[136,336][1038,389]" /></node></node></node>
<node index="1" text="" resource-
id="com.booking:id/identity_screen_content_stub"
class="android.widget.LinearLayout"
package="com.booking" content-desc=""
```

Fig. 5. Snapshot of the expected GUI (left), excerpt of the XML description of the expected GUI (middle), and snapshot of the GUI obtained during the app exploration step (right).

the application of the metamorphic testing technique, even for Android apps without source code access.

The *Network Controller* component provides full control over the network used by the AndDev to communicate with the authentication and resource servers of the AUT. This component utilizes two free tools, *Burp Suite* (PortSwigger, 2016) and *mitmproxy* (Cortesi et al., 2010), and performs several critical functions.

First, it generates and installs digital certificates, including NTSSLDC or TSSLDC, on the device. Second, it redirects network traffic through a controlled proxy, acting as an intermediary between the AUT and both Auth-Server and Res-Server. This allows for selecting the communication channel, such as switching between HTTP-CC and HTTPS-CC. Finally, the component monitors network traffic, enabling the interception and analysis of AT.

5. Experimental evaluation

Following a methodology similar to Ma et al. (2022), we conducted this study to detect authentication vulnerabilities through Metamorphic Testing at the graphical user interface level. This section describes the experiment we conducted to achieve three main goals: (1) to determine if the proposed MT technique can identify vulnerabilities in real Android mobile apps, (2) to characterize the prevalence of these vulnerabilities, and (3) to assess whether Android applications with

Table 4
Proposed metrics for answering the research questions.

Metric Name	Description
Vulnerabilities	Number of detected vulnerabilities per app; the metric is used to answer RQ_1, RQ_2, and RQ_3.
Downloads	Number of user downloads per app; the metric is combined with Metric4 to evaluate the user perceived quality for answering RQ_3.
Stars	Number of stars rated by the users per app; the metric is combined with Metric3 to evaluate the user perceived quality for answering RQ_3.

higher user-perceived quality are less vulnerable. Following the GQM (Goal, Question, Metric) approach (Cruzes et al., 2007), we formulated three research questions, each with its own rationale, to guide us in reaching these goals:

RQ_1 Is the testing technique able to detect vulnerabilities related to the username and password authentication method in real Android apps?
Rationale: this research question has the main aim to evaluate whether the proposed MT technique is suitable for detecting new vulnerabilities (i.e., that are not known a priori to the developers) in real mobile apps available on the official Google market store.

RQ_2 How do different MRs compare concerning their ability to detect vulnerabilities?
Rationale: supposing the answer to **RQ_1** is yes, this research question has twofold aims. On the one hand it has been proposed to understand the ability of each MR to detect real vulnerabilities. On the other hand, it could give an overview of the most common user authentication vulnerabilities.

RQ_3 How does vulnerabilities in Android apps vary with the user-perceived quality of the apps?
Rationale: This research question aims to explore whether Android apps with higher user-perceived quality, as indicated by higher ratings and more downloads, tend to have fewer vulnerabilities. The assumption is that apps with better user-perceived quality are more likely to undergo thorough testing, making them less prone to security issues.

To answer the research questions we defined the metrics reported in Table 4.

5.1. Object selection

To build the sample of AUTs, the objects of the experiment, we defined the inclusion (IC) and exclusion (EC) criteria summarized below. These criteria were intended to construct a sample from the most downloaded real Android apps utilizing username and password authentication methods. In this study, we did not use open-source applications, e.g. downloaded from F-Droid, since we were not currently interested in understanding the code errors causing vulnerabilities. Moreover, we believe our sample is significant since we tested our approach on the most downloaded real apps.

IC_1 The AUT can be freely downloaded from the official Brazilian or Italian Google Play Stores.

IC_2 The AUT belongs to one of the following three apps categories: *Food & Drink*, *Shopping*, and *Travel & Local*. We chose these categories because their apps typically use username and password authentication to grant access to the user's PR.

IC_3 The AUT is among the top 100 downloaded apps in its category. This criterion allowed us to build a sample composed of the most popular Android apps.

EC_1 The AUT does not adopt a user and password authentication method.

EC_2 The AUT fails during either the registration or sign-in process. In other words, it was not possible to complete the authentication process due to a failure of the selected Android app.

EC_3 We were unable to change the digital certificate of the AUT. It was needed to apply MR_1.

EC_4 It has not been possible to identify and analyze the AT after the authentication. This was necessary to apply MR_2 and MR_3.

EC_5 We could not redirect the network traffic over a HTTP-CC, as needed to evaluate the MR_4.

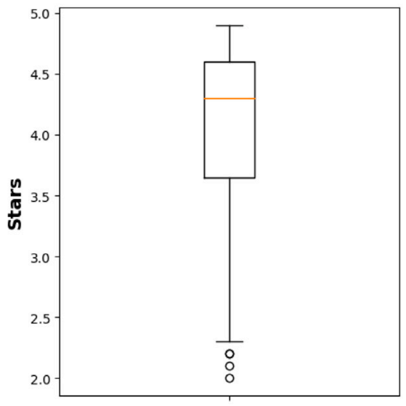
As the ICs were used solely as prerequisites to construct the initial dataset and not as elimination steps. The ECs were applied sequentially to filter out apps that did not meet the technical requirements necessary for the application of specific MRs. The order of application follows a logical progression, ensuring that fundamental prerequisites are addressed first to avoid unnecessary analysis of apps that fail critical steps. Table 5 summarizes the number of apps remaining after applying each EC. The breakdown is presented by app category (*Food & Drink*, *Shopping*, and *Travel & Local*) to provide a clear understanding of the selection process. The final sample consists of 163 apps, distributed as follows: 57 apps in *Food & Drink*, 65 apps in *Shopping*, and 41 apps in *Travel & Local*.

Fig. 6 graphically shows the characteristics of our sample by two box plots displaying the distributions of the rated stars, Fig. 6(a), and the number of downloads, Fig. 6(b), respectively. From these graphs, it is evident that more than half of the apps in the sample have been rated with more than 4.3 stars (users can rate the quality of an app from zero to five stars) and have more than 1,000,000 downloads. We defined three levels of *user-perceived quality* for an Android app: *High*, *Medium*, and *Low*. An app is classified as *High user-perceived quality* if its rating is greater than or equal to 4.3 stars and its downloads are greater than or equal to 1,000,000. Conversely, an app is classified as *Low user-perceived quality* if its rating is less than 4.3 stars and it has been downloaded fewer than 1,000,000 times. In all other cases, an app is classified as *Medium user-perceived quality*. According to this definition, in our sample, 46 apps were classified as *Low*, 46 as *Medium*, and 71 as *High* perceived quality.

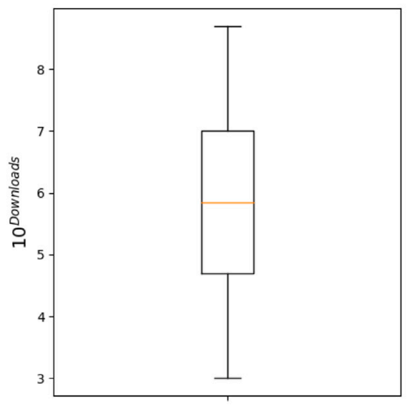
5.2. Experimental procedure

The experimental procedure we followed relies on the following steps that we executed for each app of the sample.

- *Registration of new user:* a new user account was created for testing the AUT.
- *Manual AAUT exploration:* the AUT was manually explored by one of the authors according to the scenario outlined in Fig. 3 to produce a GUI-based *Source test case*. This step was performed on an Android Virtual Device equipped with Android Q (ver. 10), one of the newest and the most diffused versions of Android at the time of the experiment.
- *Follow-up test cases generation and execution:* The GUI-based *Source test case* was automatically translated into a suite of executable *Follow-up test cases*. These test cases were executed on the same AVD used in the previous step to avoid failures due to differences in user interface characteristics. This choice does not affect the results, as the vulnerabilities we are considering do not depend on the hardware characteristics of the mobile device, particularly the device screen. Finally, details about the detected vulnerabilities were recorded in the *Test execution report*.
- *Vulnerabilities validation:* To validate the identified vulnerabilities, we consulted three security experts, each with over ten years of experience in the software security industry, including extensive work in mobile application security. Each expert independently analyzed the vulnerabilities detected by our approach. Since the source code of the applications was not available, the validation



(a) The distribution of the rated stars for the sample apps.



(b) The distribution of the downloads of the sample apps.

Fig. 6. Characterization of the sample regarding rated stars and number of downloads.

Table 5
Number of apps remaining after applying each Exclusion Criterion, by app category.

Exclusion Criterion (EC)	Food & Drink	Shopping	Travel & Local	Total remaining apps
Initial number of apps collected	70	75	60	205
EC_1: Fails during registration or login	65	70	55	190
EC_2: Does not use username/password authentication	60	68	50	178
EC_3: Unable to change the app's digital certificate	58	66	47	171
EC_4: Unable to analyze the access token	57	65	41	163

focused on reproducing the vulnerabilities by interacting with the app's graphical user interface (GUI). The experts used the same inputs and testing scenarios that triggered the vulnerabilities during the automated tests, observing the app's behavior to verify whether it violated the security properties defined in the metamorphic relations (MRs). All three experts confirmed the vulnerabilities by verifying that the app's behavior corresponded to specific weaknesses outlined in OWASP and CWE, and they assessed the potential impact, such as unauthorized access or data exposure. To mitigate threats to validity, the same tests were also executed on earlier Android versions, such as Android 6.0 (Marshmallow), Android 7.0 (Nougat), which were among the most used versions in Brazil at the time of the experiment, and Android 9.0 (Pie). We found that all vulnerabilities identified in Android 10 were also present in the earlier versions. While the work of Ma et al. (2022) has addressed the detection of authentication flaws in Android apps, we were unable to perform a direct comparison with existing tools. This is due to differences in scope and implementation, as our technique focuses on GUI-based metamorphic testing, whereas prior approaches primarily target protocol-level vulnerabilities or rely on static analysis. Despite the lack of direct baseline comparison, the ability of our technique to uncover vulnerabilities unknown to developers serves as a strong indication of its effectiveness.

5.3. Experimental results and answers to RQs

Table 6 shows the number of vulnerabilities detected and validated by each MR for each app category and Fig. 7 displays a clustering of our app sample in six groups based on the app vulnerability and the three levels of user-perceived quality. Each cluster indicates the number of apps contained and the corresponding percentage. An application is clustered as vulnerable if it exposes at least one vulnerability. The bubble charts in Figs. 8(a) and 8(b) show two detailed views of the clusters in Fig. 7. More precisely, each bubble in Fig. 8(a) reports the number of applications that have a given number of rated stars (x-axis) for which the number of vulnerabilities on the y-axis was found; analogously, Fig. 8(b) shows the distribution of the applications on the basis of their downloads (x-axis) and number of vulnerabilities (y-axis).

Table 6
Number of vulnerabilities detected by each MR for each app category.

Apps Category	MR_1	MR_2	MR_3	MR_4	MR_5	Total
Food & Drink	8	42	9	8	1	68
Shopping	4	43	13	4	0	64
Travel & Local	3	19	3	2	0	27
Total	15	104	25	14	1	159

5.3.1. Answer to RQ_1

Table 6 indicates that our testing technique successfully detected 159 vulnerabilities across the sampled apps, with at least one vulnerability identified for each MR. As shown in Fig. 8, 66.3% (108 out of 163) of the apps exposed at least one vulnerability. Moreover, Fig. 8 provides a breakdown: 71 of these 108 apps exposed a single vulnerability, 31 apps presented two vulnerabilities, 4 apps revealed three vulnerabilities, and lastly 2 apps exposed 4 and 5 vulnerabilities, respectively. Based on these findings, we can derive the following answer to RQ_1.

RQ_1 *Answer*: the proposed metamorphic-based testing technique is capable of detecting previously unknown vulnerabilities related to the username and password authentication method in real Android apps.

5.3.2. Answer to RQ_2

As reported in Table 6, most of the vulnerabilities were detected by MR_2 (104/159), followed by MR_3 (25/159). This indicates that weaknesses related to the handling of AT are the most widespread among

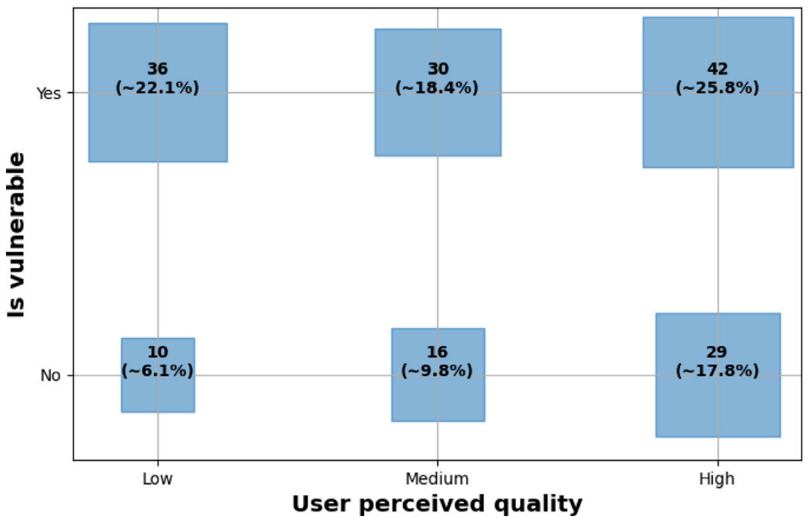


Fig. 7. Clustering of the sample by considering the vulnerability and the user perceived quality of the AUT.

Table 7
Spearman correlation coefficients.

	Downloads	Stars	User perceived quality
Vulnerabilities	−0.27	−0.13	−0.243
Is vulnerable	−0.225	−0.13	−0.193

those considered in this study. Fourteen vulnerabilities were detected by MR_4, showing that some apps still transmit sensitive data over an insecure HTTP channel. MR_1 identified 15/159 vulnerabilities related to the incorrect use of digital certificates, revealing that some apps in the sample enable authentication and access to protected data even on devices with untrusted certificates. Even though only one vulnerability was found by MR_5, it was particularly severe, as we observed that one application allowed unauthenticated users to read private chats. Given these findings, we can conclude the following answer to RQ_2:

> *RQ_2 Answer*: Certain MRs are more effective at detecting vulnerabilities than others, which suggests that some security weaknesses are much more widespread in mobile apps than others.

5.3.3. Answer to RQ_3

Figs. 7, 8(a), and 8(b) provide graphical representations of potential correlations between app vulnerabilities in the sample and three factors: user-perceived quality, rated stars, and the number of downloads. Since these plots do not visibly suggest strong correlations, we conducted a statistical analysis to confirm this observation.

To explore the relationship between these variables more rigorously, we evaluated Spearman's correlation coefficients, as shown in Table 7. Spearman's correlation was chosen for two main reasons: firstly, because it measures the strength and direction of the monotonic relationship between two ranked variables, making it appropriate for ordinal data such as user ratings and download counts. Secondly, this method is ideal when the data are not normally distributed, which was the case for our dataset. Unlike Pearson's correlation, Spearman does not assume a linear relationship and is less sensitive to outliers, providing a more robust analysis in this context.

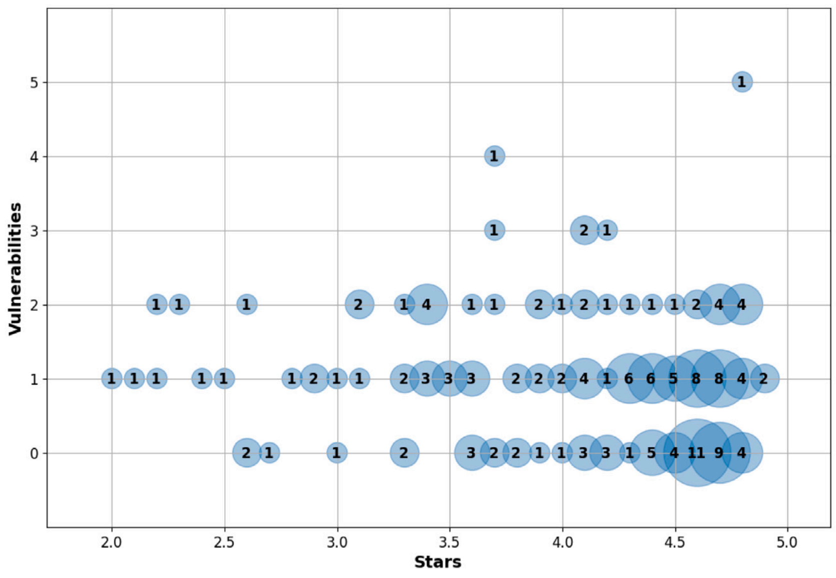
Table 7 presents the correlation coefficients calculated between the number of downloads, rated stars, and overall user-perceived quality in relation to the number of vulnerabilities and the presence or absence of vulnerabilities in the apps. The negative values of the correlation coefficients suggest an inverse relationship between these variables and the presence of vulnerabilities in the apps.

Specifically, the correlation between the number of vulnerabilities and the number of downloads is −0.27, indicating a weak negative correlation. suggests that apps with a higher number of downloads tend to have fewer vulnerabilities, though the relationship is not particularly strong. Similarly, the correlation between vulnerabilities and user-perceived quality is −0.243, and between vulnerabilities and star ratings is −0.13, both indicating weak inverse relationships.

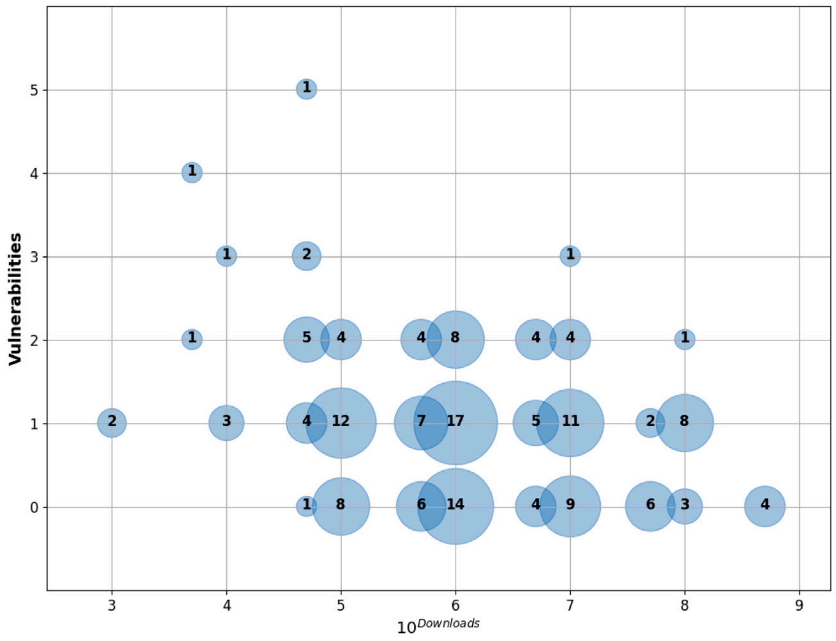
When examining whether an app is vulnerable or not, the correlations with the number of downloads (−0.225), star ratings (−0.13), and user-perceived quality (−0.193) also suggest weak negative associations. This implies that apps with higher user-perceived quality, as measured by star ratings and downloads, are somewhat less likely to contain vulnerabilities.

However, all the correlations are relatively weak, suggesting that while there is a tendency for higher-quality apps to be less vulnerable, this trend is not particularly strong. This could imply that other factors, beyond user-perceived quality and popularity, play a significant role in determining an app's security. Our findings align with those reported in Ma et al. (2022), where the authors analyzed correlations between app characteristics, such as popularity and categories, and the presence of authentication flaws. Similar to our results, they observed no strong correlations, concluding that app popularity and perceived quality are not definitive indicators of security. While their study focuses on static analysis and a broader set of characteristics, our approach specifically investigates GUI-based runtime vulnerabilities, offering complementary insights into how user-perceived quality relates to app security. Based on the statistical analysis and graphical data representations, the following conclusions can be drawn for RQ3:

> *RQ_3 Answer*: The analysis reveals a weak negative correlation between Android apps' user-perceived quality and vulnerability levels. Apps with higher ratings and more downloads tend to exhibit fewer vulnerabilities; however, the correlations identified are not sufficiently strong to make definitive claims about the influence of perceived quality on app security.



(a) Distribution of applications per vulnerabilities and rated stars.



(b) Distribution of applications per vulnerabilities and number of downloads.

Fig. 8. Distribution of vulnerabilities per user perceived quality characteristics.

5.3.4. Feedback from industries

To assess the practical utility of our technique, we engaged with 50 companies whose applications were identified as vulnerable. Thirteen companies were excluded from the process due to the lack of valid contact information or because the provided details (such as email addresses, online forms, or social media accounts) were invalid. Of the remaining 37 companies, 9 confirmed the reported vulnerabilities. Furthermore, 3 of these 9 companies requested consultations to gain more insights into the vulnerabilities and how to address them. Although the majority of the 37 companies did not respond, we observed that 26 of them updated their applications, effectively addressing and fixing the reported vulnerabilities.

6. Threats to validity

This section addresses potential threats to the validity of the study and discusses the measures taken to address these issues, as well as possible future mitigations to further improve the study's robustness.

External validity. A key threat to external validity is the sampling method used for the experiment. The applications were downloaded exclusively from the Brazilian and Italian official Google Play Stores and were limited to three specific categories. Although we attempted to mitigate this threat by including a diverse sample of apps in terms of categories and user-perceived quality, the findings may not generalize to other types of applications or markets. The experiment aimed to demonstrate that vulnerabilities could be exposed using our approach, but it does not guarantee that all applications will exhibit these vulnerabilities. To enhance external validity, future research should include a broader sample of apps from different international markets and additional categories.

Another potential threat to external validity is the use of a single version of Android. While Android 10.0 is one of the most recent and widely used versions, the study also tested previous Android versions (e.g., Android 6.0 and 7.0) to validate the results. To further address this threat, future experiments should include different Android versions and devices to ensure the technique's effectiveness across various platforms. Additionally, using only free applications instead of including open-source and paid apps may limit generalizability. Including a mix of open-source and paid applications could provide a more comprehensive assessment of the technique's applicability.

Internal validity. The technique primarily detects vulnerabilities related to self-implemented authentication methods, excluding other methods such as (i) third-party services (e.g., Google or Facebook), (ii) self-pinned certificates, and (iii) multi-step authentication methods. As a result, the presence of other types of vulnerabilities in the analyzed apps cannot be ruled out. To address this threat, future work should involve modifying the testing technique by incorporating additional MRs that account for these authentication methods and extending the Metamorphic Vulnerability Testing Environment to support their evaluation.

Conclusion validity. A threat to conclusion validity is related to the metric used for assessing user-perceived quality, which is based on the number of downloads and star ratings. Although these metrics provide objective and measurable indicators of quality, they do not capture user opinions directly. To mitigate this threat, future studies should involve real users to evaluate or confirm the user-perceived quality of the sampled apps.

Construct validity. The quality metrics used in this study, such as the number of downloads and star ratings, have limited relevance to authentication security. As evidenced by Table 7, no statistically significant relationships were found between these quality metrics and authentication security. To mitigate this threat, a variety of statistical tests were performed to explore potential correlations from different perspectives. This approach aimed to ensure that any subtle relationships between quality metrics and authentication security were thoroughly examined.

7. Related works

This section provides an overview of related work on key techniques for security testing of Android apps and the application of metamorphic testing to non-functional requirements, highlighting its relevance to mobile app security.

7.1. Security testing on Android apps

Numerous techniques have been proposed for security testing of Android apps, which can be classified into three main categories: *(i)* static, *(ii)* dynamic, and *(iii)* hybrid approaches. According to Felderer et al. (2016), static testing approaches analyze software development artifacts (e.g., requirements, design, or code) without executing them, dynamic approaches evaluate apps by observing their execution, and hybrid approaches combine both static and dynamic analyses.

Static techniques have been extensively applied to detect various security issues in Android applications, including sensitive data leaks (Zhang et al., 2021), Inter-Component Communication (ICC) vulnerabilities (Bagheri et al., 2021; Wang et al., 2023; Nirumand et al., 2024), Android WebView objects (El-Zawawy et al., 2021b), and Next-Intent Vulnerability (NIV) (El-Zawawy et al., 2021a). For example, FastDroid (Zhang et al., 2021) is a static analysis tool designed to detect data leaks in large-scale Android applications using taint analysis. In the context of ICC vulnerabilities, Bagheri et al. (2021) introduces FLAIR, a technique that efficiently analyzes incremental system changes by updating only the affected results, rather than reanalyzing the entire system, ensuring performance gains without compromising soundness or completeness. Similarly, Wang et al. (2023) presents IAFDroid, a framework that combines static analysis and taint analysis to detect collusion attacks between apps, identifying hidden inter-app communication channels that could compromise user privacy. Additionally, Nirumand et al. (2024) introduces VAnDroid3, a model-driven framework that identifies ICC vulnerabilities in both intra- and inter-app communications, creating unified security models and formal analysis to identify six categories of ICC vulnerabilities. Furthermore, static analysis tools like WebVSec (El-Zawawy et al., 2021b) and NIVD (El-Zawawy et al., 2021a) have also been proposed to address specific vulnerabilities. WebVSec detects new vulnerabilities in WebView objects with improved precision and efficiency compared to prior tools. NIVD, on the other hand, applies a type system to analyze smali code and identify sequences of calls to NIV-related APIs. Despite their scalability, static approaches often face challenges with high false-positive rates due to their reliance on approximations of runtime behaviors.

Dynamic testing techniques have been mainly adopted for the detection of memory leaks (Liang et al., 2018) and ICC vulnerabilities (El-Zawawy et al., 2022; Hu et al., 2024). Liang et al. (2018) proposed an analysis system called AppLance, which runs the app through its UI (User Interface), logs the execution data, and analyzes them to identify memory leaks. El-Zawawy et al. (2022) presents DLAIR, a dynamic analysis technique that enhances Intent resolution in Android to prevent security issues, such as sensitive information leaks. Unlike static taint analysis, DLAIR dynamically works across multiple apps simultaneously and can be integrated into the Android OS. Hu et al. (2024) introduces SIAT, a dynamic analysis technique designed to detect privacy-sensitive data leaks in ICCs by tracking data and control flows during runtime using taint tagging.

Hybrid methods combine static and dynamic analyses to leverage the strengths of both approaches. Notable examples include (Liu et al., 2018; Wang et al., 2020), which detect SSL/TLS vulnerabilities by combining static identification of vulnerable code with dynamic confirmation, and RONIN (Romdhana et al., 2023), which uses reinforcement learning for dynamically generating ICC exploits. Moreover, hybrid methods have been used to identify permission misuse, such as in Demissie and Ceccato (2020), which combines static and dynamic analysis to detect second-order permission re-delegation vulnerabilities in Android apps, where indirect permission flows between components are exploited for unauthorized access. Similarly, Natesan et al. (2020) proposed a hybrid approach for detecting data leaks in Android applications by combining static and dynamic analysis, where runtime permissions and logcat information are used to detect and categorize data leaks based on their probability and associated risk levels. Hybrid techniques aim to reduce false positives while maintaining scalability but often require significant computational resources.

Among these hybrid approaches, Ma et al. (2022) introduced AUTHEXPLOIT, a sophisticated tool designed to detect authentication flaws in Android apps. AUTHEXPLOIT employs a combination of static analysis and manual template definitions to identify vulnerabilities in password authentication protocols, such as insufficient SSL/TLS validation and plaintext password transmission. By leveraging intermediate representations, call graph construction, and dependency analysis, AUTHEXPLOIT achieves remarkable precision in pinpointing these flaws. Notably, their evaluation, conducted on a diverse dataset of 1,200 Android apps across multiple categories, demonstrated the tool's robustness and applicability to real-world scenarios, setting a high benchmark in the field of security testing. While AUTHEXPLOIT represents a significant advancement in the detection of authentication vulnerabilities, its focus is primarily on static dependencies and predefined templates. Consequently, it excels in identifying vulnerabilities related to code structure and implementation but does not address dynamic runtime behaviors or interactions with graphical user interfaces (GUIs). This limitation opens a complementary avenue for research, as certain vulnerabilities only emerge during runtime or through user interactions with the app's GUI. Our work builds on the foundation laid by Ma et al. (2022) by addressing this gap. Specifically, we propose a GUI-based metamorphic testing technique that evaluates dynamic properties of Android apps during runtime. Unlike static methods, our approach derives metamorphic relations (MRs) from real-world vulnerabilities reported in OWASP and CWE, enabling the detection of authentication flaws that manifest during GUI interactions. While AUTHEXPLOIT sets the standard for precision in static analysis, our technique complements it by focusing on runtime behaviors, offering a holistic view of app security.

7.2. *Metamorphic testing for non-functional requirements*

MT has been successfully applied in several application domains, such as Web (Andrade et al., 2019), Machine Learning, Concurrent Programs (Sun et al., 2023), and others. However, Segura et al. (2017b, 2018) discussed the hypothesis of its application as an effective and practical approach to detect issues related to NFRs. Therefore, studies have aimed to check its effectiveness of in detecting issues such as performance (Azimian et al., 2019; Johnston et al., 2019; Ayerdi et al., 2022) and security (Corradini et al., 2023; Mai et al., 2020b; Rahman and Izurieta, 2023; Chaleshtari et al., 2023).

Regarding performance testing, Johnston et al. (2019) applied MT to Adobe Experience Platform Launch Tag Manager software for identifying performance anomalies. They created test pages that implemented Adobe tags as well as test pages that implemented the tags through Experience Platform Launch, expecting both pages would be functionally identical, but their page-load performance would probably be different.

Azimian et al. (2019) identified known bugs and hot-spots energy (i.e., performance issues) on Android apps using MT. They designed MRs based on known bugs and hot-spots energy and then checked if the MRs had been violated. More recently, Ayerdi et al. (2022) proposed an MR pattern called PV (Performance Variation) for the identification of failures in CPRs (Cyber-physical systems), which are systems that integrate software with physical processes. The proposed PV eases the identification of performance MRs in CPSs, alleviating the test oracle problem.

In the context of security testing, Mai et al. (2020b) proposed an approach based on MT concepts to evaluate the security of Web systems. The authors adopted the technique proposed by Huang et al. (2003), according to which an intentionally invalid input (i.e., *source input*) and a valid input (i.e., *follow-up input*) are generated for each MR. They first selected a set of known vulnerabilities that usually occur on Web systems, then designed a set of 22 system-agnostic MRs to detect them. Finally, security properties were automatically captured from the Web system for checking whether the MRs had been violated.

Rahman and Izurieta (2023) extended (Mai et al., 2020b) study by introducing a substantial expansion of the MRs catalog. They meticulously developed 54 additional system-agnostic MRs, thereby amplifying the original collection from 22 to a comprehensive total of 76. They employed automated testing procedures within two well-known web systems, namely, Jenkins and Joomla, to showcase the practical efficacy of the catalog. Remarkably, the augmented approach proved its ability to detect a substantial 85% of vulnerabilities in those systems.

Chaleshtari et al. (2023) employed MT to evaluate banking software functionalities based on their inherent properties. They devised a comprehensive set of 11 MRs meticulously categorized into three primary groups aligned with fundamental banking operations, namely, Deposit, Withdrawal, and Transfer. These MRs focus on vulnerabilities specific to web systems and business processes. While their work emphasizes static and hybrid analyses, our approach differs by targeting vulnerabilities that emerge during runtime interactions with graphical user interfaces (GUIs) in Android mobile apps. This distinction allows us to address authentication and authorization issues unique to mobile platforms, which are less explored in the existing literature.

8. Conclusions and future work

This study introduced a GUI-based Metamorphic Testing technique designed to identify vulnerabilities in Android applications resulting from improper username and password authentication methods, particularly those that disregard OAuth 2.0 guidelines. Our work differentiates from prior studies in two key aspects. First, unlike existing approaches that primarily focus on detecting authentication vulnerabilities through static code analysis and automated orchestration (Chaleshtari et al., 2023), our method operates at the GUI level during runtime, enabling the identification of flaws that manifest through user-driven interactions. Second, while other metamorphic testing techniques have been successfully applied to web systems by manipulating HTTP requests and URLs (Ma et al., 2022), our technique extends this paradigm to the mobile domain, addressing vulnerabilities in Android applications through GUI-based testing. This shift allows us to detect flaws that may not be observable through web-focused or purely static approaches. Building on prior studies that adapted MT for web applications (Mai et al., 2020b,a; Chaleshtari et al., 2023) this tool-supported technique to detect five prevalent and representative real-world vulnerabilities associated with username and password authentication, as identified by OWASP. Our approach involved designing five MRs and implementing a Metamorphic Vulnerability Testing Environment to evaluate them. We applied this technique to 163 widely used Android applications from the Brazilian and Italian markets. The results were significant: 159 vulnerabilities unknown to developers were uncovered. Of these, 108 applications were found to have at least one vulnerability, demonstrating the technique's effectiveness. Three security experts independently reviewed the identified vulnerabilities and confirmed their existence by interacting with the apps' graphical user interfaces, validating the effectiveness of our technique in real-world scenarios. In addition to the technical evaluation, we sought feedback from industry professionals to assess the practical utility of our technique. We engaged with 50 companies whose applications were identified as vulnerable. Among the 37 companies that we successfully contacted, 9 confirmed the vulnerabilities, and 3 of them requested consultations for further guidance. Furthermore, 26 companies updated their applications, effectively addressing and fixing the reported vulnerabilities. This real-world feedback highlights the practical relevance and impact of our approach in identifying and mitigating security risks in mobile applications. An interesting finding from our analysis is that no strong correlation was found between higher perceived quality, more downloads, or better ratings and fewer vulnerabilities. This suggests that other factors may play a more significant role in the presence of vulnerabilities in mobile applications. Overall, our work contributes to the security testing field by providing an effective and practical technique for detecting authentication vulnerabilities in Android apps. We believe that the demonstrated ability to uncover vulnerabilities unknown to developers, combined with the real-world feedback from industry partners, validates the utility and impact of the proposed method.

Future research will focus on expanding the scope and applicability of our GUI-based metamorphic testing approach to further enhance its effectiveness in detecting authentication vulnerabilities across a broader range of mobile applications and environments. Building on the feedback from the reviewers, we outline several key directions.

First, we plan to extend our study to application categories that handle highly sensitive information, such as finance, communication, and social networking. This will allow us to evaluate the applicability and robustness of our approach in high-stakes environments.

Another important direction is the investigation of vulnerabilities beyond username and password methods, encompassing multi-factor authentication (MFA), biometric systems, and third-party login services (e.g., Google and Facebook OAuth). This broader analysis aims to address evolving authentication practices and provide a more comprehensive security framework.

Future work will also involve empirical evaluations that compare the detection effectiveness of our metamorphic relations (MRs) with existing methods. In particular, we will assess overlapping MRs, such as improper certificate validation and missing encryption, to provide quantitative insights into their relative performance.

Additionally, we recognize the need to evaluate the scalability of our GUI-based approach compared to traditional static and decompilation methods. Although our technique currently requires manual intervention, including user registration and GUI interaction, we plan to formally assess its scalability by applying it to larger and more complex applications. This will help identify bottlenecks and improve automation where feasible.

To enhance the transparency and reliability of our findings, we will incorporate open-source Android applications in future studies. By leveraging community feedback on platforms like GitHub, we aim to cross-reference vulnerabilities detected by our technique with known issues and patches in public repositories.

Another avenue of future work involves analyzing paid applications to investigate whether they exhibit different vulnerability patterns compared to free applications. This comparison will help assess whether the pricing model influences the level of security investment and overall vulnerability presence.

Finally, we plan to adapt and extend our GUI-based metamorphic testing approach to desktop and web applications. This cross-platform expansion will explore how metamorphic testing can detect vulnerabilities across different user interfaces and environments, broadening the applicability of our method beyond the mobile ecosystem.

CRediT authorship contribution statement

Domenico Amalfitano: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Supervision, Methodology, Formal analysis, Data curation, Conceptualization. **Misael Júnior:** Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Formal analysis, Data curation, Conceptualization. **Anna Rita Fasolino:** Writing – review & editing, Writing – original draft, Supervision. **Marcio Delamaro:** Writing – review & editing, Writing – original draft, Validation, Supervision, Investigation, Conceptualization.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Marcio Delamaro reports financial support was provided by Conselho Nacional de Desenvolvimento Científico e Tecnológico. Misael Junior reports financial support was provided by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

Prof. Marcio Delamaro is partially funded by the Conselho Nacional de Desenvolvimento Científico e Tecnológico – Brazil (CNPq), Brazil, process number 308636/2021-0

Dr. Misael Júnior is partially financed by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001.

Appendix. List of acronyms

Acronyms

AG	Authorization Grant
AndDev	Android Device
AR	Authentication Request
AT	Access Token
AUT	Application Under Test
Auth-Server	Authorization Server
CWE	Common Weakness Enumeration
FUTC	Follow-up test case
HTTP-CC	HTTP Connection Channel
HTTPS-CC	HTTPS Connection Channel
MITM	Man-in-the-Middle
MR	Metamorphic Relationship
MT	Metamorphic Testing
NAAU	Not Authenticated and Not Authorized User
NTSSLDC	No Trusted SSL/TLS Digital Certificate
OWASP	Open Web Application Security Project
PR	Protected Resources
Res-Server	Resource Server
RO	Resource Owner
SIS	Sign-In Screen
SSDC	Self Signed Digital Certificate
SSL	Secure Sockets Layer
STC	Source test case
TSL	Transport Security Layer
TSSLDC	Trusted SSL/TLS Digital Certificate
UPC	User and Password Credentials

Data availability

Data will be made available on request.

References

Amalfitano, Domenico, Riccio, Vincenzo, Amatucci, Nicola, Simone, Vincenzo De, Fasolino, Anna Rita, 2019. Combining automated GUI exploration of android apps with capture and replay through machine learning. Inf. Softw. Technol. 105, 95–116. http://dx.doi.org/10.1016/j.infsof.2018.08.007, URL https://www.sciencedirect.com/science/article/pii/S0950584918301708.

Amalfitano, Domenico, Riccio, Vincenzo, Paiva, Ana C.R., Fasolino, Anna Rita, 2018. Why does the orientation change mess up my android application? From GUI failures to code faults. Softw. Test. Verif. Reliab. 28 (1), e1654. http://dx.doi.org/10.1002/stvr.1654, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/stvr.1654. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/stvr.1654. e1654 stvr.1654.

Andrade, Stevão A., Santos, Italo, Junior, Claudinei Brito, Junior, Misael C., Souza, Simone R. S., Delamaro, Márcio E., 2019. On applying metamorphic testing: An empirical study on academic search engines. In: Proceedings of the 4th International Workshop on Metamorphic Testing. MET, pp. 9–16.

Android, 2023a. The getevent tool. https://source.android.com/docs/core/interaction/input/getevent. [Online; Accessed 19 December 2023].

Android, 2023b. Uiautomator. https://stuff.mit.edu/afs/sipb/project/android/docs/tools/help/uiautomator/index.html. [Online; accessed 19 December 2023].

Android Developers, 2024. Monkey | android developers. URL https://developer.android.com/studio/test/other-testing-tools/monkey?hl=en. (Accessed 03 July 2024).

Arkin, Brad, Stender, Scott, McGraw, Gary, 2005. Software penetration testing. IEEE Secur. Priv. 3 (1), 84–87.

Ayerdi, Jon, Terragni, Valerio, Jahangirova, Gunel, Arrieta, Aitor, Tonella, Paolo, 2024. GenMorph: Automatically generating metamorphic relations via genetic programming. IEEE Trans. Softw. Eng. 50 (7), 1888–1900. http://dx.doi.org/10.1109/TSE.2024.3407840.

Ayerdi, Jon, Valle, Pablo, Segura, Sergio, Arrieta, Aitor, Sagardui, Goiuria, Arratibel, Maite, 2022. Performance-driven metamorphic testing of cyber-physical systems. Trans. Reliab..

Azimian, Farzaneh, Faghih, Fathiyeh, Kargahi, Mehdi, Mirdehghan, SM Mahdi, 2019. Energy metamorphic testing for android applications. In: Proceedings of the 30th International Symposium on Personal, Indoor and Mobile Radio Communications. PIMRC, pp. 1–6.

Bagheri, Hamid, Wang, Jianghao, Aerts, Jarod, Ghorbani, Negar, Malek, Sam, 2021. Flair: efficient analysis of android inter-component vulnerabilities in response to incremental changes. Empir. Softw. Eng. 26, 1–37.

Ceci, L., 2023. Annual number of app downloads from the google play store. https://www.statista.com/statistics/734332/google-play-app-installs-per-year/. [Online; Accessed 31 March 2024].

Chaleshtari, Nazanin Bayati, Pastore, Fabrizio, Goknil, Arda, Briand, Lionel C., 2023. Metamorphic testing for web system security. IEEE Trans. Softw. Eng. 49 (6), 3430–3471. http://dx.doi.org/10.1109/TSE.2023.3256322.

Chen, Tsong Y., Cheung, Shing C., Yiu, Shiu M., 1998. Metamorphic Testing: a New Approach for Generating Next Test Cases. Technical Report, Technical Report HKUST-CS98-01, Department of Computer Science, Hong Kong University of Science and Technology, Hong Kong.

Corradini, Davide, Pasqua, Michele, Ceccato, Mariano, 2023. Automated black-box testing of mass assignment vulnerabilities in restful APIs. In: Proceedings of the 45th International Conference on Software Engineering. ICSE.

Cortesi, Aldo, Hils, Maximilian, Kriechbaumer, Thomas, contributors, 2010. mitmproxy: A free and open source interactive HTTPS proxy. URL https://mitmproxy.org/. [Version 10.1].

Cruzes, D., Mendonca, M., Basili, V., Shull, F., Jino, M., 2007. Extracting information from experimental software engineering papers. In: Proceedings of the 26th International Conference of the Chilean Society of Computer Science. SCCC, pp. 105–114. http://dx.doi.org/10.1109/SCCC.2007.11.

CWE, 2021. CWE top 25 most dangerous software weaknesses. https://cwe.mitre.org/top25/archive/2021/2021_cwe_top25.html. [Online; Accessed 03 October 2023].

CWE, 2023a. Common weakness enumeration. https://cwe.mitre.org/. [Online; Accessed 03 October 2023].

CWE, 2023b. CWE-288: Authentication bypass using an alternate path or channel. https://cwe.mitre.org/data/definitions/288.html. [Online; Accessed 25 December 2023].

CWE, 2023c. CWE-295: Improper certificate validation. https://cwe.mitre.org/data/definitions/295.html. [Online; Accessed 25 December 2023].

CWE, 2023d. CWE-311: Missing encryption of sensitive data.

CWE, 2023e. CWE-384: Session fixation. https://cwe.mitre.org/data/definitions/384.html. [Online; Accessed 25 December 2023].

CWE, 2023f. CWE-613: Insufficient session expiration. https://cwe.mitre.org/data/definitions/613.html. [Online; Accessed 25 December 2023].

D. H, Ed, 2012. Mobile app authentication architectures. https://mobile-security.gitbook.io/mobile-security-testing-guide/general-mobile-app-testing-guide/0x04e-testing-authentication-and-session-management. [Online; Accessed 18 December 2023].

Demissie, Biniam Fisseha, Ceccato, Mariano, 2020. Security testing of second order permission re-delegation vulnerabilities in android apps. In: Proceedings of the 7th International Conference on Mobile Software Engineering and Systems. pp. 1–11.

El-Zawawy, Mohamed A., Faruki, Parvez, Conti, Mauro, 2022. Formal model for inter-component communication and its security in android. Computing 104 (8), 1839–1865.

El-Zawawy, Mohamed A., Losiouk, Eleonora, Conti, Mauro, 2021a. Do not let next-intent vulnerability be your next nightmare: type system-based approach to detect it in android apps. Int. J. Inf. Secur. 20 (1), 39–58.

El-Zawawy, Mohamed A., Losiouk, Eleonora, Conti, Mauro, 2021b. Vulnerabilities in android webview objects: Still not the end! Comput. Secur. 109 (1), 1–10.

Felderer, Michael, Büchler, Matthias, Johns, Martin, Brucker, Achim D, Breu, Ruth, Pretschner, Alexander, 2016. Security testing: A survey. In: Advances in Computers. vol. 101, Elsevier, pp. 1–51.

Gamma, Erich, Beck, Kent, 2017. JUnit. URL https://junit.org/. [Online; Accessed 06 February 2024].

Hu, Yupeng, Kuang, Wenxin, Zhe, Jin, Li, Wenjia, Li, Keqin, Zhang, Jiliang, Hu, Qiao, 2024. SIAT: A systematic inter-component communication real-time analysis technique for detecting data leak threats on android. J. Comput. Secur. 32 (3), 291–317.

Huang, Yao-Wen, Huang, Shih-Kun, Lin, Tsung-Po, Tsai, Chung-Hung, 2003. Web application security assessment by fault injection and behavior monitoring. In: Proceedings of the 12th International Conference on World Wide Web. pp. 148–159.

ISO, 2011. ISO/IEC 25010:2011, Systems and Software Engineering – Systems and Software Quality Requirements and Evaluation (SQuaRE) – System and Software Quality Models. ISO, Geneva, Switzerland.

Johnston, Owen, Jarman, Darryl, Berry, Jeffrey, Zhou, Zhi Quan, Chen, Tsong Yueh, 2019. Metamorphic relations for detection of performance anomalies. In: Proceedings of the 4th International Workshop on Metamorphic Testing. MET, IEEE, pp. 63–69.

Júnior, Misael C, Amalfitano, Domenico, Garcés, Lina, Fasolino, Anna Rita, Andrade, Stevão A, Delamaro, Márcio, 2021. Dynamic testing techniques of non-functional requirements in mobile apps: A systematic mapping study. ACM Comput. Surv. 1 (1), 1–36.

Lahav, E.H., 2016. The oauth 2.0 authorization framework. https://datatracker.ietf.org/doc/html/rfc6749#page-8. [Online; Accessed 18 December 2024].

Laricchia, Federica, 2024. Mobile operating systems' market share worldwide. https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/. [Online; Accessed 29 August 2024].

Liang, Hongliang, Wang, Yudong, Yang, Tianqi, Yu, Yue, 2018. AppLance: A lightweight approach to detect privacy leak for packed applications. In: Proceedings of the Nordic Conference on Secure IT Systems. Springer, pp. 54–70.

Liu, Yang, Zuo, Chaoshun, Zhang, Zonghua, Guo, Shanqing, Xu, Xinshun, 2018. An automatically vetting mechanism for SSL error-handling vulnerability in android hybrid web apps. World Wide Web 21 (1), 127–150.

Ma, Siqi, Bertino, Elisa, Nepal, Surya, Li, Juanru, Ostry, Diethelm, Deng, Robert H, Jha, Sanjay, 2019. Finding flaws from password authentication code in android apps. In: European Symposium on Research in Computer Security. Springer, pp. 619–637.

Ma, Siqi, Li, Juanru, Nepal, Surya, Ostry, Diethelm, Lo, David, Jha, Sanjay Kumar, Deng, Robert H., Bertino, Elisa, 2022. Orchestration or automation: Authentication flaw detection in android apps. IEEE Trans. Dependable Secur. Comput. 19 (4), 2165–2178. http://dx.doi.org/10.1109/TDSC.2021.3050188.

Ma, Siqi, Liu, Yang, Nepal, Surya, 2020. Are android apps being protected well against attacks? IEEE Wirel. Commun. 27 (3), 66–71.

Mai, Phu X., Goknil, Arda, Pastore, Fabrizio, Briand, Lionel C., 2020a. SMRL: A metamorphic security testing tool for web systems. In: Proceedings of the 42nd International Conference on Software Engineering: Companion Proceedings. ICSE-Companion, pp. 9–12.

Mai, Phu X., Pastore, Fabrizio, Goknil, Arda, Briand, Lionel, 2020b. Metamorphic security testing for web systems. In: Proceedings of the 13th International Conference on Software Testing, Validation and Verification. ICST, pp. 186–197. http://dx.doi.org/10.1109/ICST46399.2020.00028.

Mueller, B., Schleier, S., Willemsen, J., Holguera, C., 2023. Mobile Application Security Testing Guide (MASTG). OWASP.

Natesan, Shriya, Gupta, Megha Rajeev, Iyer, Lakshmi Natesan, Sharma, Deepak, 2020. Detection of data leaks from android applications. In: Proceedings of the 2th International Conference on Inventive Research in Computing Applications. ICIRCA, pp. 326–332.

Nirumand, Atefeh, Zamani, Bahman, Ladani, Behrouz Tork, 2024. A comprehensive framework for inter-app ICC security analysis of android apps. Autom. Softw. Eng. 31 (2), 1–61.

OAuth, 2012. Oauth 2.0.

O'Dea, Simon, 2020. Smartphone users worldwide 2016–2021. https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/. [Online; Accessed 10 January 2020].

OWASP, 2001. Open worldwide application security project (OWASP). https://owasp.org/. [Online; Accessed 22 December 2024].

OWASP, 2023a. M3: Insecure authentication/authorization. https://owasp.org/www-project-mobile-top-10/2023-risks/m3-insecure-authentication-authorization.html. [Online; Accessed 22 December 2024].

OWASP, 2023b. M5: Insecure communication. https://owasp.org/www-project-mobile-top-10/2023-risks/m5-insecure-communication.html. [Online; Accessed 22 December 2024].

OWASP, 2023c. M9: Insecure data storage. https://owasp.org/www-project-mobile-top-10/2023-risks/m9-insecure-data-storage.html. [Online; Accessed 22 December 2024].

OWASP, 2023d. OWASP mobile top 10. https://owasp.org/www-project-mobile-top-10/. [Online; Accessed 22 December 2024].

PortSwigger, 2016. Burp suite. https://portswigger.net/burp. [Online; Accessed 19 December 2023].

Potter, Bruce, McGraw, Gary, 2004. Software security testing. IEEE Secur. Priv. 2 (5), 81–85.

Rahman, Karishma, Izurieta, Clemente, 2023. An approach to testing banking software using metamorphic relations. In: Proceedings of the 24th International Conference on Information Reuse and Integration for Data Science. IRI, pp. 173–178.

Ribeiro, Victor Vidigal, Cruzes, Daniela Soares, Travassos, Guilherme Horta, 2018. A perception of the practice of software security and performance verification. In: Proceedings of the 25th Australasian Software Engineering Conference. ASWEC, pp. 71–80.

Romdhana, Andrea, Merlo, Alessio, Ceccato, Mariano, Tonella, Paolo, 2023. Assessing the security of inter-app communications in android through reinforcement learning. Comput. Secur. 131, 1–13.

Saad, E., Mitchell, R., 2019. Web Security Testing Guide (WSTG). OWASP.

Segura, Sergio, Durán, Amador, Troya, Javier, Cortés, Antonio Ruiz, 2017a. A template-based approach to describing metamorphic relations. In: Proceedings of the 2nd International Workshop on Metamorphic Testing. MET, IEEE, pp. 3–9.

Segura, Sergio, Fraser, Gordon, Sanchez, Ana B., Ruiz-Cortés, Antonio, 2016. A survey on metamorphic testing. IEEE Trans. Softw. Eng. 805–824.

Segura, Sergio, Troya, J., Durán, A., Ruiz-Cortés, A., 2017b. Performance metamorphic testing: motivation and challenges. In: Proceedings of the 39th International Conference on Software Engineering: New Ideas and Emerging Technologies Results Track. ICSE-NIER, pp. 7–10.

Segura, Sergio, Troya Castilla, Javier, Durán Toro, Amador, Ruiz Cortés, Antonio, 2018. Performance metamorphic testing: A proof of concept. Inf. Softw. Technol..

Segura, Sergio, Zhou, Zhi Quan, 2018. Metamorphic testing 20 years later: A hands-on introduction. In: Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings. pp. 538–539.

Skybox Security, 2020. Vulnerability and threat trends mid-year report 2020. https://www.skyboxsecurity.com/resources/report/vulnerability-threat-trends-mid-year-report-2020/. [Online; Accessed 03 October 2023].

Sun, Chang-Ai, Dai, Hepeng, Geng, Ning, Liu, Huai, Chen, Tsong Yueh, Wu, Peng, Cai, Yan, Wang, Jinqiu, 2023. An interleaving guided metamorphic testing approach for concurrent programs. ACM Trans. Softw. Eng. Methodol. 33 (1), 1–21.

Tian-yang, Gu, Yin-Sheng, Shi, You-yuan, Fang, 2010. Research on software security testing. World Acad. Sci. Eng. Technol. 70, 647–651.

Wang, Yingjie, Xu, Guangquan, Liu, Xing, Mao, Weixuan, Si, Chengxiang, Pedrycz, Witold, Wang, Wei, 2020. Identifying vulnerabilities of SSL/TLS certificate verification in android apps with static and dynamic analysis. J. Syst. Softw. 167, 110609.

Wang, Bin, Yang, Chao, Ma, Jianfeng, 2023. IAFDroid: Demystifying collusion attacks in android ecosystem via precise inter-app analysis. IEEE Trans. Inf. Forensics Secur..

Wei, Xuetao, Wolf, Michael, 2017. A survey on HTTPS implementation by android apps: issues and countermeasures. Appl. Comput. Inform. 13 (2), 101–117.

Weir, Charles, Rashid, Awais, Noble, James, 2020. Challenging software developers: dialectic as a foundation for security assurance techniques. J. Cybersecur. 6 (1), 1–16.

Weyuker, Elaine J., 1982. On testing non-testable programs. Comput. J. 465–470.

Zhang, Jie, Tian, Cong, Duan, Zhenhua, 2021. An efficient approach for taint analysis of android applications. Comput. Secur. 104, 102161.