Easy As *Child's Play*: An Empirical Study on Age Verification of Adult-Oriented Android Apps

Yifan Yao, Shawn McCollum, Zhibo Sun, Yue Zhang*
Drexel University, Email: fanfannnmn@protonmail.ch, {sem445, zs384, yz899}@drexel.edu

Abstract

The rapid growth of mobile apps has provided convenience and entertainment, including adult-oriented apps for users 18 and older. Despite various strategies to prevent minors from accessing such content, the effectiveness of these measures remains uncertain. This paper investigates these mechanisms and proposes a novel detection solution: GUARD (Guarding Underage Access Restriction Detection). GUARD determines relevant components (e.g., those that can accept the user's age or birthdate) based on the spatial relationships of the components in a layout and tracks the data flows through taint analysis. Recognizing static analysis limitations, GUARD also dynamically interacts with apps to identify age-related input components, which are then used for precise taint analysis. Our analysis of 31,750 adult-only apps (out of 693,334 apps on Google Play) reveals that only 1,165 (3.67%) implement age verification, with the majority relying on the weakest method, the age gate (which simply asks users if they are over 18). Even apps with stronger age verification (e.g., document uploads, online ID verification) can be bypassed using simple methods like false IDs or fake documents. They can also be circumvented through accounts from services without age checks (e.g., OAuth abuse) or by exploiting regional differences via VPNs. This paper also proposes countermeasures to enhance the effectiveness of age verification methods, which received positive feedback from Google through our email exchanges.

1 Introduction

The rapid growth of mobile apps has brought significant convenience and entertainment to users worldwide, spanning various age groups and interests. Among these, adult-oriented apps (e.g., "*17+ apps*") occupy a unique and often controversial segment of the app market. These apps are specifically designed for adults, typically requiring users to be at least 18 years old due to their content or services (e.g., dating platforms, gambling). Those apps are deemed inappropriate or harmful for minors. Studies have shown that children who frequently view mature content are at a higher risk of developing anxiety and depression [6], and engaging in inappropriate conversations and actions [50]. One study from the National Center for Missing and Exploited Children even shows that a significant number of missing children were enticed online through adult-oriented apps [16].

To prevent children from accessing adult content, various strategies are implemented worldwide. For example, the Children's Internet Protection Act (CIPA) in the U.S. requires the implementation of internet safety policies [17], including *technology protection* measures to block or filter obscene content and content harmful to minors. In China, the government employs a combination of legal (e.g., Cybersecurity Law [15]) and technological measures to control adult content. Many Chinese apps require real-name registration for internet users, which helps enforce age restrictions. That is, users must provide their real names and national ID numbers when registering for those adult-oriented apps, with the information being cross-checked against government databases [28].

Despite the implementation of numerous policies and technical solutions to restrict minors' access to adult content, the adoption and effectiveness of these measures remains questionable. This paper aims to address the gap in our understanding of the ecosystem of the adult-oriented apps and enforcement effectiveness of age verification mechanisms in these apps. To that end, a tool that can effectively identify the age verification mechanism enforced by these adult-oriented apps is crucial. However, currently, there is no tool that can achieve such a goal to the best of our knowledge, and designing such a tool is subject to multiple challenges.

Specifically, unlike some resources (e.g., location) that can be directly accessed through Android APIs and readily identified by programming analysis tools, Android does not provide interfaces for retrieving the user's age. Consequently, apps may employ diverse methods for age verification. Directly searching for keywords like "age verification" or "birthday" often results in high false positives. Taint analysis

*Corresponding author: Yue Zhang.

offers a potential solution by tracing the data flow of collected inputs of interest (e.g., age) to determine if they undergo verification checks. However, determining and distinguishing between taint sources and sinks presents significant challenges. We cannot indiscriminately designate all UI components containing strings such as *"age verification"*, *"age"*, or *"birthday"* as either taint sources or taint sinks. For instance, these strings may appear in UI elements that serve as input boxes accepting user input (the desired UI components), text labels providing instructions for user interaction (e.g., *"please enter your age in the input box below"*), or dialogs displaying verification results (e.g., *"age verification fails"*). Dynamically loaded strings downloaded from servers further complicate static taint analysis efforts.

To address these challenges, we propose a tool named Guarding Underage Access Restriction Detection (GUARD), designed to identify age verification mechanisms within adult-oriented apps. GUARD first extracts and analyzes UI elements within the application to identify strings such as "age" and "birthday." Based on the spatial relationships between UI components, GUARD determines which components are relevant and which will serve as taint sources or sinks. For instance, a text label (`TextView`) containing these strings is likely instructing user input (e.g., "*please enter your age*") and is typically located near an input box (`EditText`) that accepts user input. This input box (not the label itself) should be treated as a taint source. Meanwhile, recognizing the limitations of static analysis in detecting strings downloaded from the internet, to extract their XML structure and collect possible components containing dynamically loaded strings of interest for taint analysis, ensuring a comprehensive assessment of age verification mechanisms.

By analyzing 31,750 adult-only apps (tagged by Google Play) out of 693,334 apps using our GUARD , we identified that only 1,165 (3.67%) adult-only apps have implemented age verification. We also have many interesting findings. For example, categories such as Finance (which involves gambling) and Business (which may sell alcohol and tobacco) show higher percentages of age verification, likely due to stringent regulations. The "17+ rating" by Google is often inaccurate, with some apps requiring users to be 21 years or older, and some setting the age limit as low as 16 years or below. Such lower limits may expose minors to adult content. Moreover, we found that some apps collect personal data (e.g., full names, dates of birth, and addresses) during the verification process to ensure they are interacting with a real person. However, this may raise privacy concerns, especially for children who might not understand the implications of sharing their data. Age gates (which directly ask users if they are over 17) are the most widely implemented method (31.84%), especially in less critical categories like Entertainment and Lifestyle, but are also the weakest, relying solely on self-declared ages. Conversely, Biometric verification is the least utilized (8.48%) due to its higher implementation costs, despite its robustness.

To make matters worse, we found that even for those apps that implement age verification, the verification can be bypassed "as easily as child's play." Our assumption is that the attacker possesses only minimal capabilities, mimicking the behavior of a child without advanced technical skills. Even under this scenario, six potential attacks exist, including simple methods such as inputting false IDs or uploading fake documents. Attackers can also gain access to these age-restricted apps through accounts from other services that do not enforce age verification (OAuth Trust-Chain Abuse) or by using a version of the app in other countries via VPNs (different regional regulations allow users to access less restricted versions of the same app). We also noticed that shopping apps had the highest vulnerability rate (91%), with 52% relying on age gate.

The contributions of the work are summarized as follows:

- We are the first to technically (not based on policy and regulation analysis) analyze and explore the inadequacy of age verification mechanisms in adult-oriented mobile apps, highlighting the risks posed to minors who might access inappropriate or harmful content.
- We propose a novel tool, GUARD , designed to identify age verification mechanisms within mobile applications. GUARD leverages static analysis to detect relevant UI components, while the dynamic behavior analyzer is employed exclusively to load runtime-dependent data. Once the necessary data is captured, static analysis is reapplied to complete the process.
- We conducted a comprehensive evaluation of adult-only apps, examining their categories (RQ1), download popularity (RQ2), app ratings (RQ3), and developers (RQ4). We also assessed whether these apps implemented age verification mechanisms (RQ6). For apps with age verification, we analyzed the types of verification methods used (RQ5) and the potential attacks that could bypass them. Our findings revealed that only 1,165 (3.67%) of the analyzed adult-only apps implemented age verification. Among those, only 8.48% adopted strict biometric authentication capable of defending against the attacks proposed.

2 Background

2.1 Adult-Oriented Apps and Regulations

Adult-oriented apps are applications designed specifically for adults due to their content, themes, or functionalities (e.g., sexual material, violent imagery). These apps are widely considered harmful to minors [6, 16, 50]. Consequently, various countries have developed regulations to prevent children from accessing adult content. To gain deeper insights, we conducted an analysis of regulations across different countries related to adult-oriented apps. In particular, the regulations were metic-

Regulation	Country	Year	Age	AV	PC	CF	P
Children's Internet Protection Act [17]	USA	2006	18+	✔	✗	✔	✔
JuSchG [49]	Germany	2003	18+	✔	✗	✔	✔
JMStV [30]	Germany	2023	18+	✔	✗	✔	✔
LCEN [23]	France	2023	15+	✔	✗	✔	✔
Age-Appropriate Design Code [46]	UK	2020	18+	✔	✗	✔	✔
Online Safety Act [21]	Australia	2021	18+	✔	✔	✔	✔
Cybersecurity Law [28]	China	2017	18+	✔	✗	✔	✔
Internet Safety Act [22]	Japan	2009	18+	✔	✔	✔	✔
Youth Protection Act [25]	South Korea	1997	19+	✔	✗	✔	✔
Information Technology Rules [47]	India	2021	18+	✔	✗	✔	✔
Online Harms Act [12]	Canada	1997	17+	✔	✔	✔	✔
Child and Adolescent Statute [37]	Brazil	1990	18+	✔	✗	✔	✔

Table 1: Regulations in various countries. Age verification (AV). Parental Controls (PC). Content Filtering (CF). Penalties (P).

Method	Simplicity	Reliability	Privacy	Scalability	Accuracy	Cost
Offline Age Verification						
Age Gate	★★★	★☆☆	★★★	★★★	★☆☆	★☆☆
Template-based Check	★★☆	★☆☆	★★★	★★★	★☆☆	★☆☆
Online Age Verification						
ID Number Verification	★★★	★★★	★☆☆	★★★	★★★	★★★
Credit Card Verification	★★★	★★☆	★★☆	★★★	★★☆	★★☆
Document (ID) Upload	★★☆	★★★	★☆☆	★★☆	★★★	★★☆
Biometric Verification	★★☆	★★★	★☆☆	★★☆	★★★	★★★

Table 2: Comparison of Age Verification Methods. High: ★★★. Medium: ★★☆. Low: ★☆☆

ulously sourced from authoritative official databases and reputable publications to ensure the credibility and comprehensiveness of the data. To further enhance the reliability of our analysis, translations were carefully conducted to maintain the accuracy and integrity of the original regulatory language. As shown in Table 1, all countries employ a multi-faceted approach that includes age verification, parental controls, content filtering, and criminal penalties. Some states in the U.S., such as Utah (Social Media Regulation Act) and Louisiana (Act 440), have implemented laws requiring robust age verification for access to certain types of content, particularly adult content [42]. These laws often require developers to verify age through government-issued IDs or equivalent methods. This demonstrates a global consensus on the importance of robust measures to protect minors from harmful online content.

Our findings revealed the following: (i) Most countries set the age restriction at 18, with South Korea at 19 and Canada focusing on children under 17. In France, 15 years old is the legal age of sexual consent, which aligns with broader societal norms regarding maturity and personal responsibility. This consistency often extends to regulations in related areas, such as exposure to online content. (ii) Japan and Australia complement regulatory measures with public awareness campaigns to promote online safety. (iii) Criminal penalties underscore the seriousness of non-compliance. For instance, the USA's Children's Internet Protection Act (CIPA) mandates internet safety measures in schools and libraries, with non-compliance risking the loss of millions in federal funding [17]. Similarly, in the UK, failure to implement age verification can result in fines of up to £250,000 or 5% of turnover [48].

2.2 Age Verification in Adult-Oriented Apps

Currently, the most common technical solution to prevent minors from accessing adult content is age verification. As shown in Table 2, age verification methods can be divided into two categories:

(I) Offline Age Verification. Offline age verification methods operate without the need for internet connectivity, often leveraging local databases or algorithms to validate user input: *(i)* Age Gate, a straightforward approach where users are asked to input their date of birth or confirm they meet a minimum age requirement before accessing restricted content. Its simplicity and ease of implementation make it a widely adopted solution. *(ii)* Template-Based Age Verification, a method involves requesting users to provide a government-issued ID for more rigorous age verification. The app processes the provided information by matching it against locally stored templates or applying algorithms to confirm the user's age.

(II) Online Age Verification. Online age verification methods require internet connectivity to validate a user's age through various online resources and databases. Common approaches include: *(i)* ID Number Verification Services: Users input their ID numbers, which are then cross-referenced with government or credit records to verify their age. *(ii)* Credit Card Verification: This method leverages credit card information, as credit cards are generally issued only to adults, to confirm the user's age. *(iii)* Document (ID) Upload: Users are prompted to upload a scanned copy or photo of a government-issued ID. The system employs Optical Character Recognition (OCR) and additional verification techniques to authenticate the document and validate the user's age. *(iv)* Biometric Verification: This advanced method involves technologies such as facial recognition or fingerprint scanning. For example, users might take a selfie, which the system compares against the photo on their ID or a database to confirm their age.

3 Threat Model and Scope

Assumptions. In this paper, we explore the security practices of adult-oriented apps in scenarios where a child might bypass age verification. We assume: (1) The child lacks advanced technical skills such as reverse engineering; (2) The child can find and use easily accessible tools (e.g., VPNs, fake ID generators) to bypass age checks; and (3) The child may access a parent's photo ID or device but not their biometric data, such as fingerprints or facial recognition.

Scope. We limit our scope to Android apps available on

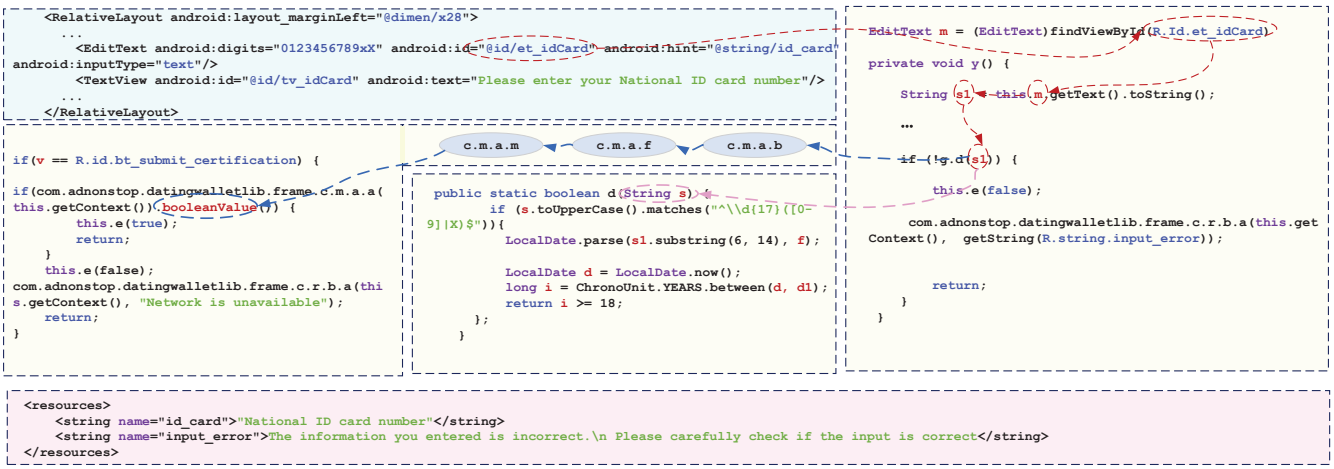

Figure 1: **The snippet from *21: Virtual Social App* shows fragments from `string.xml` (pink), the layout file (blue), and Java code (yellow).**

Google Play for the following reasons: First, Android is one of the most widely used mobile operating systems, and its open nature facilitates easier analysis of app data and usage patterns. Second, Google Play provides a clear categorization system, including a "17+" tag for adult-only apps. Third, Google Play is renowned for its strict vetting process. Our insight is that if apps on such a strictly protected market like Google Play are susceptible to age verification bypasses, the situation is likely to be even worse in other markets.

4 Example of Age Verification Analysis

Our objective is to conduct a comprehensive analysis of the age verification mechanisms employed by adult-oriented applications using program analysis. To achieve this, we will reverse engineer the adult-only apps to gain insights from the reverse engineering process. Particularly, in this section, we will focus on analyzing an app called *21: Virtual Social App* (`com.adnonstop.camera21`), a Chinese adult dating app that uses national ID numbers to verify user age. Chinese IDs contain birthdate information in an 18-digit format, where the 7th to 14th digits represent the birthdate (`yyyymmdd`). Users must enter their ID during registration. As shown in Figure 1, the code snippet defines attributes for an `EditText` element that collects the ID. The attribute `android:digits="0123456789xX"` restricts input to numbers and specific characters, ensuring the ID format is correct, while `android:id="@id/et_idCard"` provides a unique reference for use in Java code.

The code in the yellow boxes are the Java code. The code fragment `findViewById(R.Id.et_idCard)` finds the `EditText` element with the ID `et_idCard` in the layout and assigns it to the variable m. The code fragment `g.d(s1)` is a method that verifies the user's age, and if the check fails, the app displays an error message to the user, informing the user that the information entered is incorrect and prompting the user to check the input. The age verification is performed both online and offline. (*i*) The offline calculates the user's age by comparing the extracted birthdate with the current date. This is done by subtracting the birth year from the current year and adjusting for whether the current date is before or after the user's birthdate within the year. If the calculated age meets the minimum age requirement (i.e., 18 years or older), the system grants access to the application. If not, access is denied. (*ii*) In the online age verification process, the national ID number provided by the user is passed through multiple classes within the application. These classes create a network request (which includes the ID number) that is sent to the back-end server. Upon receiving the request, the back-end server performs the necessary verification checks to validate the ID number against a national database or another authoritative source. This may involve cross-referencing the ID number with stored records to confirm the user's age and other details. If the verification check fails, indicating that the user does not meet the age requirements, the server will refuse the user registration attempt and send an appropriate response back to the application. In this case, the failed cases (either failing the offline check or the online check) will present the user with a string, which is linked to the string whose id is `input_error` in the `strings.xml`.

5 GUARD Design

Based on the motivating example, identifying age verification requires processing UIs to recognize elements and tracking code flows to understand constraints and behaviors. As

shown in Figure 2, we introduce our tool, GUARD (Guarding Underage Access Restriction Detection), with three main components:

- **Static UI Processor (§5.1).** The UI processor identifies and records UI elements that collect specific user inputs (e.g., national ID, date of birth). It parses the APK to access layout files, mapping elements to inputs and saving only the IDs/types of elements directly involved in data collection.
- **Static Behavior Analyzer (§5.2).** The behavior analyzer identifies entry points and analyzes age verification. It uses the IDs and types from the UI processor to locate relevant Java files and applies taint analysis to check for age verification in the code.
- **Dynamic Behavior Tester (§5.3).** Static analysis falls short for app age verification, missing network interactions and dynamically loaded strings from remote backends. Our dynamic tester addresses this by loading app layouts in real-time, extracting the XML structure of the current interface, and searching for age verification keywords. Taint analysis further verifies age verification by tracing user input and its impact on app behavior.

GUARD was developed using Soot [43], with approximately 2,000 lines of custom Java code. We chose Soot over alternatives such as CodeQL [40] or FlowDroid [9] primarily due to our familiarity with Soot, having previously built several projects on it. Nevertheless, the core concepts underpinning GUARD are adaptable to other platforms like CodeQL or FlowDroid. Soot provides robust modeling of essential program dependencies, including control, data, alias, exception, and library dependencies. For our analysis, we primarily utilized control and data dependencies to implement our tool. Additionally, given the inherent complexity of Android apps, we use Soot's call graph to address these challenges.

5.1 Static UI Processor

The UI processor identifies UI elements that take the user's input of interest. It works through three main steps:

Step I: Processing Layout Files. The UI processor unpacks and parses the APK file to access layout files, which are XML files defining the app's UI and structure. These files contain elements like `TextView`, `EditText`, and `Button`, with attributes such as `android:text`, `android:hint`, and `android:contentDescription`, which may include age verification strings (e.g., *"age", "birth", "verify", "18","National ID number"*). For instance, in Figure 1, `android:text` is set to *Please enter your National ID card number* (See the example in §4)", containing the age-related string *National ID number*". Regular expressions are used to search these strings based on their format.

Step II: Processing String Resource File. The UI processor extracts the string resource file (`strings.xml`) because directly analyzing the layout files is insufficient; many Android apps use the string resource file to store text resources separately from the layout XML files. In the `strings.xml`, each string is assigned a unique ID (e.g., `@string/id_card`). These IDs are essential for maintaining a coherent link between the layout XML files and the strings. To link the reference IDs in the layout files to the actual strings during analysis, a straightforward solution is to traverse the XML tree to extract relevant UI elements and their attributes and then search for them in the `strings.xml` file. Once a string of interest (e.g., *"age", "birth"*) is identified, the ID and the corresponding UI element can be recorded for future reference. However, this process is time-consuming because there is not a one-to-one mapping, and many strings are repeatedly referenced. For example, a string "*The information you entered is incorrect.*" can be used multiple times in different contexts within the app. To save time, we first check all the strings in the `strings.xml` to find the strings of interest and then search for their IDs in the layout files. This significantly improves search performance, as each string is iterated only once in the entire search. The UI processor also extracts hard-coded strings and the corresponding IDs of the UI elements directly from the app layout files, as some developers favor this practice.

Type	Example
UI elements accept confirmation input	
├Confirmation	*I confirm that I am over 18 years old*
├Disclaimer	*Age verification is necessary to ensure compliance*
└Notice	*You must be at least 18 years old*
UI elements accept concrete input	
├Hint	*Enter your age **here***
└Format Explanation	*Date of birth (YYYY-MM-DD)*
UI elements instruct the input	
├Label	*Birthdate*
├Button Text	*Verify Age Submit*
└Prompt	*Please input the age **below***
UI elements display output	
├Fail	*Age verification failed*
└Success	*Age verified successfully.*

Table 3: Comparison of Possible Age Verification String.

Step III: Resolving Relationship. The UI processor utilizes UI elements and their IDs to establish their relationship with actual inputs (e.g., values related to age such as birthday). Interestingly, we notice that among those UI elements associated with strings related to age verification, some directly collect the inputs of interest, while others serve merely as components (e.g., labels) that instruct users to enter the input of interest into other UI elements. For example, as shown in Figure 1, the `EditText` UI element is the element that collects the input of interest, whereas `TextView` UI element, although it contains strings related to age verification, does not collect any input from the user. Instead, it instructs users to input their national ID into the input box.

As such, the types of these UI elements are crucial. As

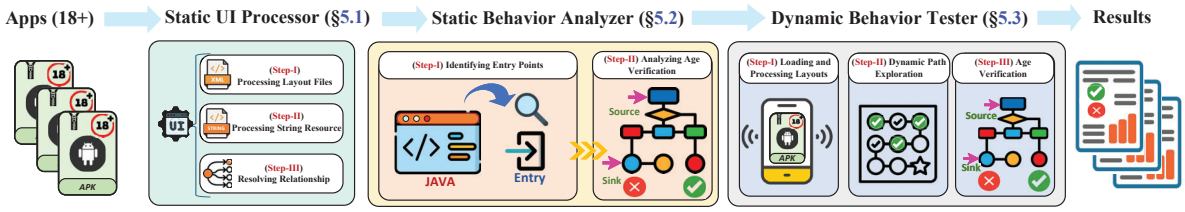

Figure 2: **Overview of GUARD**

illustrated in Table 3 , the UI elements can be categorized into four groups, each serving specific purposes. To accurately resolve the relationship between these strings and the actual input, it is essential to understand the context and meaning of these strings. However, this can be challenging. Different developers might use different strings or terminologies for similar purposes. For example, for the "*Hint*" message, one app might use "*Please enter your birthdate*" while another may simply use "*DOB*", requiring us to recognize various synonyms and patterns. One straightforward solution is to use a Large Language Model (LLM) [51, 53] or other Nature Language Processing (NLP) techniques to understand and process those natural language strings. However, running LLMs can be computationally intensive and may introduce performance bottlenecks. We conducted tests on a LLM (i.e., ChatGPT) and observed notable performance challenges primarily arising from communication and text generation overhead. Specifically, the process of generating detailed responses introduced significant latency, which was compounded by the model's reliance on iterative token-by-token text output. Additionally, the communication overhead between the application layer and the LLM framework, especially during high-frequency query scenarios, further impacted the overall performance. Given that each app may contain hundreds or thousands of strings across multiple files, the solution is not practical.

Our idea leverages the context of the strings within UI elements (we examine which UI elements contain age verification-related strings and the types of these UI elements) to infer their relationship with the actual input. For instance, if an age verification-related string is contained within the attribute of an `EditText` element, we can infer that this element serves as the input box for collecting age verification information. As shown in Table 4, we have demonstrated all the relationships between the UI elements and the corresponding recorded IDs. To determine the scope of the UI elements, we conducted an in-depth review of the Android developer documentation [20]. This systematic analysis aimed to identify all potential UI components that could be utilized within apps, with a particular focus on elements such as input fields, buttons, dropdowns, and other interactive components. To ensure a comprehensive understanding, we complemented this analysis by reverse engineering a diverse set of real-world Android applications spanning various categories and use cases. This reverse engineering process allowed us to extract UI-related code and layouts, providing practical insights into how these elements are implemented in real-world scenarios. Additionally, this approach enabled us to validate the completeness of our identified scope by cross-referencing documentation findings with actual app implementations, ensuring no significant UI element was overlooked.

We now discuss how we infer the relationship based on the types of UI elements and the types of strings:

- **UI Elements Accepting Confirmation Input.** Using a confirmation button (i.e., confirmation input) to implement age verification is a common practice in applications to ensure compliance with age-related regulations. This method requires the user to actively confirm that they meet the age requirement before accessing age-restricted content or services. In such methods, the `Button` or `ImageButton` contain age verification strings. Some developers may use `Switch Button` or `checkbox`. The IDs of these UI elements that can receive user inputs are of interest and will be recorded for future reference.
- **UI Elements Accepting Concrete Input.** Implementing age verification using an UI elements such as `EditText` or `DatePicker` allows users to enter their concrete birthdate, age, or national ID number, which is then validated to ensure they meet the required age threshold. In these cases, `EditText` will contain the necessary age verification strings (e.g., hint information). Some developers may utilize a `DatePicker` to enable users to select their birthdate. The IDs of these UI elements that receive user concrete inputs are crucial and will be recorded for future reference.
- **UI Elements Instructing Concrete Input.** There are some components that do not directly accept inputs from the user but instead instruct the user on how the app may accept inputs. For this type of component, we observe that they are usually nested with the actual UI elements that can receive user inputs. For example, as shown in Figure 1, the `TextView` with ID `tv_idCard` does not accept any user input, while the `EditText` with ID `et_idCard` does. Both of these UI elements are in the same layout view `RelativeLayout`. Therefore, our idea is to record the IDs of input elements that can receive input within the same

	TextView	ImageView	EditText	Button	ImageButton	CheckBox	RadioButton	Toggle	Switch	Spinner	DatePicker	Dialog	Toast
UI elements Acc. Confirmation Input	○	○	○	●	●	●	●	●	●	●	○	○	○
UI elements Acc. Concrete Input	○	○	●	○	○	○	○	○	○	○	●	○	○
UI elements Instructing Input	●	●	○	○	○	○	○	○	○	○	○	○	○
UI elements Displaying Outputs	○	○	○	○	○	○	○	○	○	○	○	●	●

Table 4: Relationship Between UI Elements and Recorded IDs. The highlighted one refers to the UI elements with ID recorded. ○ refers to the UI element does not contain the string of interest. ● refers to the UI element contains the string of interest.

layout view as the UI elements containing age verification strings, which instruct users on how to input. For instance, in the above example, the `EditText`'s ID `et_idCard` will be recorded, and `TextView`'s ID `tv_idCard` will not.

- **UI Elements Displaying Output.** We will also identify certain strings that may contain age verification messages, which are not directly associated with UI elements. For example, an application might specify a string such as `"Age verification failed"` to be displayed when the age verification process is unsuccessful. These strings may either not be linked to any UI elements (being used exclusively in the Java code) or may be associated with dialog-based views, such as `Toast` or `Dialog`. Only the IDs of these UI elements will be recorded for future reference.

5.2 Static Behavior Analyzer

The behavior analyzer works by first identifying entry points and then analyzing age verification. To be more specific:

Step I: Identifying Entry Points. To identify the relevant Java implementation, referred to as the entry point, we focus on Java files initializing UI elements tied to collected IDs. These elements are typically bound using the Android API `findViewById`, which links unique IDs to Java code. Instead of examining hundreds or thousands of Java files, we begin with the launching activity specified in the `AndroidManifest.xml`, as age verification logic is likely among the first activities a user encounters (Otherwise, it is easily bypassed). After identifying the offline age verification logic, we analyze its associated functions to check for an online verification process, as apps often combine offline checks with online requests. If an online implementation is found, we stop further analysis, as developers rarely implement multiple age verification methods in a single app. During this process, we also identify entry points for apps with hard-coded age verification strings, recording the corresponding UI elements for future reference.

Step II: Analyzing Age Verification. In this step, the behavior analyzer examines the implementations of the Java code from the entry points to statically determine whether the app has implemented age verification. One straightforward solution is to use code pattern matching, which involves searching for specific patterns in the code that indicate age verification logic. This can include searching for keywords or functions commonly associated with age checks. However, this method may miss complex or obfuscated implementations where age verification is not done using common patterns or keywords. Additionally, it does not provide insights into the flow of data or the actual logic used in verification. As shown in Figure 1 , in the online age verification process, the national ID number provided by the user is passed through multiple classes (e.g., from `c.m.a.b` to `c.m.a.f`) within the application. Such a case cannot be resolved by code pattern matching. Our idea is to use taint analysis to construct the relationship between the source and the sink and pinpoint the verification logic. This approach enables us to focus solely on the presence of conditional statements, irrespective of their specific implementation or underlying calculations. For instance, "18+ confirmation" apps must include conditional statements (e.g., the app terminates if the user confirms being under 18). In the absence of such conditional statements, age verification mechanisms are effectively nonexistent. Particularly, by marking sources, tracking data flow, identifying sinks, and analyzing control flow, we can comprehensively understand how age data is handled within the app.

- **Taint Sources.** Sources are the points in the code where user input related to age is introduced. These inputs need to be marked or "*tainted*" to track their flow through the application. Specifically, as shown in Table 4 (those highlighted in red), UI elements that accept user inputs for confirmation (e.g., `Button`) and those for concrete inputs (e.g., `EditText`, `DatePicker`) should be directly tainted. Additionally, for UI elements that guide concrete or confirmation inputs, we taint all UI elements capable of receiving input within the same layout view as these UI elements. For instance, in Figure 1, we taint the `EditText` widget with the ID `et_idCard`. This ensures that any input provided by the user in this field is tracked throughout the application.
- **Taint Sinks.** Sinks are the endpoints where tainted data (i.e., the age input) is used to make decisions or is presented to the users. Identifying sinks involves locating points in the code where age data influences the application's behavior. There are two possible scenarios to consider: (i) We mark the conditional statements that check if the age meets certain criteria, such as `i > = 18` as shown in Figure 1. Additionally, we mark API calls to online age verification services (e.g., `HttpURLConnection` or `OkHttpClient`). It is important to note that when the

check happens online, we cannot obtain the exact logic of the verification process. Consequently, we cannot be entirely certain that the input age is used specifically for age verification. (ii) We also mark the UI elements that display the verification results as sinks (e.g., "*You passed age verification*"). This is to reduce false positive. In offline verification, there are cases where an ID number includes the user's birthday. If we only consider conditional statements, the app might simply validate the ID number's format without checking the age or birthday. By treating those messages as sinks, we can filter out such cases.

5.3 Dynamic Behavior Tester

Static analysis alone is often insufficient for age verification in Android applications, primarily because it does not account for dynamic network interactions and can miss important elements downloaded from the backend. Many applications download strings and other resources, such as prompts for entering age verification information, from remote servers at runtime. If these strings (e.g., "*please enter your age*") are not embedded directly in the app's code but are instead retrieved via network requests, static analysis will fail to detect them.

Therefore, the dynamic behavior tester complements the static analyzer by addressing this critical limitation of static analysis: its inability to detect runtime-dependent data. By monitoring the execution of the application, the dynamic analyzer identifies data that is generated or retrieved during runtime, such as dynamically loaded information. Once this runtime data is identified, it is integrated back into the static analysis pipeline, enhancing static taint analysis by providing a more accurate and comprehensive understanding of data flows. While a dynamic-only taint analyzer could provide runtime insights, our static preprocessing (e.g., resolving relationships between UI components and data flows) ensures that dynamic analysis is more targeted and efficient. Also, dynamic taint analysis alone might fail to account for all paths or inputs because it requires user interaction simulations, which may not cover all use cases or edge conditions. GUARD mitigates this by combining dynamic observations with static code insights. At a high level, the dynamic analyzer works by first loading and processing the app layouts dynamically. Having collected all the IDs and types of UI elements of interest, taint analysis (which is similar to the static behavior analysis) is used to confirm the presence of age verification:

Step I: Loading and Processing Layouts. In this step, our behavior tester dynamically loads and processes app layouts by automatically installing and launching the application on a test device or emulator using `Appium` [8]. This setup mimics real-world usage, enabling interaction with the app as a user. Upon launch, the tester extracts the XML structure of the current interface, capturing UI elements, attributes, and their hierarchical relationships, including attributes such as `android:clickable` that indicate clickable elements.

Step II: Path Exploration. This step dynamically explores the paths and identifies clickable UI elements containing the strings of interest. Having collected those layout files in step I, their extracted XMLs are then analyzed for specific keywords related to age verification (e.g., "*enter your age*"). If found, relevant elements and contexts (e.g., IDs, types) are recorded. Otherwise, clickable elements on the current interface are identified to explore deeper navigation paths. This process is repeated iteratively, systematically navigating through the app's structure until all UI elements have been examined (except for those requiring specific inputs, such as a user-entered password, to access). That is, our UI exploration focuses solely on clickable UI elements and does not account for UI elements that require inputs (this is our limitation, as acknowledged in the limitations in section 8). Please refer to Appendix A for more details.

Step III: Age Verification. Having collected all the IDs and types of UI elements of interest, we again use taint analysis to confirm the presence of age verification. Based on the type of elements, we determine which UI elements should be marked as taint sources, and which UI elements should be marked as taint sinks (please also refer to Table 4 for more details). For example, we taint the UI element that directly collects the user's input. A dialog that displays messages such as "Age verification failed" is marked as a taint sink. This step is similar to the step III of Static Behavior Analyzer, and we will omit the details for conciseness.

6 Attacking Apps with Age Verification

We now discuss how the attacks can compromise the age verification methods introduced in §2.2. Table 5 illustrates a clear hierarchy in the effectiveness of age verification methods against various attacks:

(A1) False Age Self-Declaration. Age gate is a simple and widely used method for age verification but is easily bypassed by entering a false date of birth. For instance, a 15-year-old can claim to be 18 or older. Many apps, including popular ones like Reddit, are vulnerable to this issue. Reddit requires users to be 17+ to install the app and 13+ to create an account, hosting communities (subreddits), including those with mature content marked as "NSFW" (*Not Safe For Work*). However, users can bypass the age check by providing a fake birthdate.

(A2) Fake ID Number Self-Declaration. Users can bypass age verification by generating fake ID numbers if the app only checks their format (e.g., digits representing birthdates). For example, a Chinese national ID has 18 digits, with the 7th to 14th indicating the birthdate, and a fake ID number can be generated using tools such as the Random Chinese ID Number Generator [19]. Some apps, such as *Game for Peace*

and *Fantasy Westward Journey*, enhance verification by cross-referencing ID numbers and names with official databases. However, this method is still vulnerable to issues like data breaches and leaked ID numbers, which can be found online without accessing the dark web [1]. For example, we discovered some websites that released criminals' information, including their real ID numbers. Additionally, we found websites that published the ID numbers of award winners. For privacy reasons, we omit links to such websites.

(A3) OAuth Trust-Chain Abuse. OAuth (Open Authorization) is a protocol [18] that allows apps to securely access user information without sharing passwords. In an OAuth setup, a user can log into one application (the client) using their credentials from another application (the authorization server). This creates a trust chain between the two applications. This attack stems from inconsistent age verification across OAuth apps. If the authorization server does not enforce age verification, a user can use an OAuth flow to authenticate through an app without age restrictions, thereby gaining access to an app that does require age verification. For example, Dating Wallet, a dating app, requires users to be 17+ but allows login via QQ and Xiaomi accounts, which have no age restrictions. More details can be found in Appendix B.1.

(A4) Identifier Spoofing. Minors can bypass age verification systems by using authentic documents belonging to adults, such as national IDs. They often borrow IDs from family members or friends of legal age, with or without their knowledge. As mentioned earlier, many national IDs and personal documents are available online due to data breaches or careless publishing. For instance, the dating app *Peeks Social* only requires users to upload an ID photo for verification, making it vulnerable to such attacks (Details can be found in Appendix B.2). Interestingly, we have found that many online paid services provide adult verification with real person-assisted verification or adult account rental for some games [2,3]. Theoretically, only biometric-based age verification offers relatively strong protection. However, this method may not always be secure if the app fails to verify both the user's age and identity simultaneously (i.e., ensuring the adult user is genuine and not someone providing online assistance).

(A5) Age Verification Downgrade. Different regions may have varying regulations and guidelines concerning age restrictions for accessing the same app. These discrepancies are influenced by local laws, cultural norms, and regulatory bodies (as discussed in §2.1). For example, *League of Legends (LoL)*, a highly engaging game with strategic depth and social interaction, has strict regulations in some countries to combat gaming addiction among minors. The competitive nature of LoL can lead to toxic behavior, including harassment, aggressive language, and negative interactions [7]. In China, the government mandates real-name registration and imposes strict gameplay time limits for users under 18. However, in the U.S., although LoL is restricted to minors under 13, it does not have any in-app age verification, allowing minors to use the app freely. As a result, minors in China can use a VPN to access the international version of LoL, which does not have the same restrictions.

[1]Although these ID numbers are real, they do not belong to the users attempting to bypass the age verification. Therefore, we still consider them to be *fake ID numbers*

	Age Gate	Template-based Check	ID Number Verification	Credit Card Verification	Document Upload	Biometric Verification
(A1)	✔	✗	✗	✗	✗	✗
(A2)	✔	✔	✗	✗	✗	✗
(A3)	✔	✔	✔	✔	✔	✔
(A4)	✔	✔	✔	✔	✔	✗
(A5)	✔	✔	✔	✔	✔	✔
(A6)	✔	✔	✔	✔	✔	✔

Table 5: Summary of Attacks against different Age verification Methods

Category	Count	Pct (%)	Category	Count	Pct (%)
Entertainment	9,455	29.78	Health & Sports	1,028	3.24
Social	6,669	21.00	Finance	560	1.76
Lifestyle	4,239	13.35	Business	536	1.69
Shopping	2,647	8.33	Education	479	1.51
Food & Drink	2,644	8.32	Photography & Editors	466	1.47
News & Books	1,385	4.36	Travel & Local	370	1.17
Productivity	1,126	3.55	Other	146	0.46

Table 6: Distribution of apps w.r.t their categories

(A6) Tool-based Data Manipulation. Some adult apps have age verification mechanisms that can be easily bypassed using specific tools. These tools are straightforward to use, allowing minors to circumvent age restrictions with minimal effort. For instance, we encountered a tool named "*MT-Arknights* [1]", designed to bypass restrictions (e.g., age and time limits) enforced by *Arknights*, a game that involves mature themes such as disease, conflict, and ethical dilemmas. The principle behind this tool is to manipulate the data packets involved in the age verification process. This manipulation is achieved by establishing a MitM proxy between the server and the client device. The process requires the user to download and install the proxy's certificate on their device to facilitate the interception of data. Once the MitM proxy is in place, all requests sent to the server for age verification are intercepted by the proxy, which then alters these requests and their corresponding responses. More details can be found in Appendix B.3.

7 Evaluation

7.1 Experiment Setup

Dataset. We used the Androzoo [4] dataset to identify adult-oriented apps based on tags originally sourced from Google. By analyzing metadata from *693,334* Android apps (May

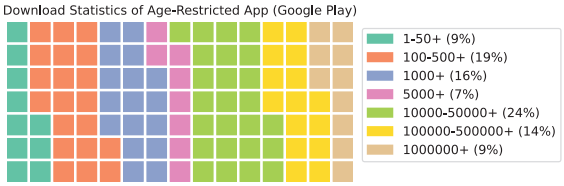

Figure 3: Distribution of download numbers for age-restricted apps

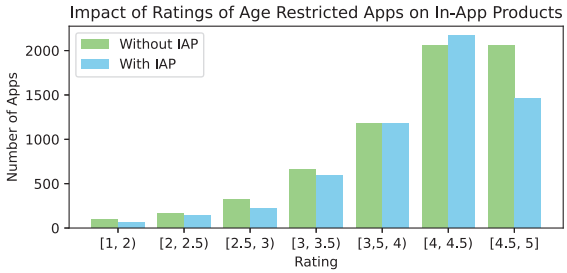

Figure 4: The relationship between app ratings and the presence of in-app products (IAP). Please note that some apps do not have any ratings.

2024 snapshot, covering 2017–2024), we identified *31,750* adult apps with diverse content, consuming 2,14 TB.

Environment. The experiments were conducted on a server running Ubuntu 22.04.4 LTS, equipped with 32 CPU cores and 64 GB of memory. The evaluation itself was performed on a desktop utilizing 8 concurrent processes.

FP and FN Analysis. We randomly sampled 100 apps categorized as having age verification mechanisms and 100 apps without any verification methods. To ensure accuracy, we manually verified the results. Each app was thoroughly analyzed using a combination of reverse engineering and dynamic testing to assess their age verification processes. As a result, we have identified three false positives, where the collected input was not used for verifying age. For example, *Equestrian Singles* simply asks the user to input a birthday to register an account, and this information is optional; even if the user does not input a birthday, the registration can still be successful. We also identified two false negatives, where the tool failed to identify obfuscated code and the data flow within the apps. We believe this limitation exists in all other taint analysis tools, as it is an inherent limitation of taint analysis.

7.2 Experiment Results

(Q1) What reasons contribute to the designation of apps as adult-only across different app categories?

To gain a better understanding of the landscape of adult-only apps and to investigate potential age verification mechanisms tailored to different reasons for designating an app as adult-only, we assigned each adult-only app to a suitable category based on its category tag in the Google Play Store, resulting in 14 distinct categories. As shown in Table 6, most apps fall into Entertainment, Social, and Lifestyle, comprising 64.13% of the total.

By manually analyzing a randomly selected sample of 50 apps in each category, we found distinct reasons for their adult-only designation. For example, Entertainment (29.78%) features explicit content, such as sexual and violent material. Social (21.00%) includes dating and adult chat apps, while Lifestyle (13.35%) involves mature themes. Interestingly, we found that the *Shopping* (8.33%) and *Food & Drink* (8.32%) categories may include apps selling restricted products such as alcohol and tobacco. *News & Books* and Productivity feature explicit literature, including joke books. *Health & Sports* cover adult health topics, and Finance and Business involve gambling or adult-oriented services. *Photography & Editors* enable explicit content creation, *Education* covers mature topics, *Travel & Local* offers adult travel services, and *Other* (0.46%) includes miscellaneous apps.

(Q2) Are these adult-only apps widely downloaded?

The data highlights a significant concern regarding the accessibility of adult-only apps. Apps with higher download counts and better user ratings are likely to attract a broader audience, including minors, as these metrics often reflect strong user engagement and perceived reliability. As illustrated in Figure 3, the popularity of these apps varies, with the largest segment (24%) falling in the *10,000–50,000+* downloads range. Notably, a smaller yet concerning proportion (9%) of adult-only apps surpass *1,000,000* downloads, demonstrating their widespread reach and potential risk of exposure to minors.

(Q3) What are the ratings of the adult-only apps?

We analyzed the ratings of these adult-only apps and found that a significant proportion of adult-only apps fall within the higher rating brackets of 4.0 to 5.0, as shown in Figure 4. This indicates that poorly rated adult-only apps are less common. Meanwhile, given that adult-only apps may require users to make in-app purchases (IAP) to access full functionality, which can significantly impact minors who may not be able to afford them, we conducted further analysis on the ratings concerning IAP requirements. Figure 4 shows that more than half of the adult-only apps do not require IAP, and there are more high-rated (4.0-5.0) no-IAP apps than IAP-required apps. This suggests that many high-rated adult-only apps are potentially accessible to minors.

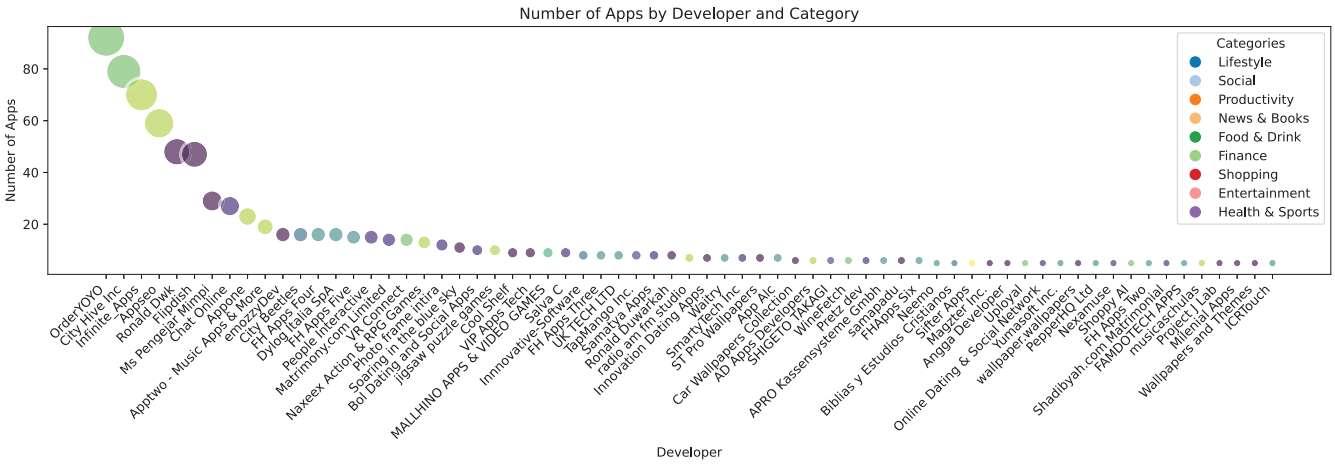

Figure 5: The distribution of adult-only apps across different developers and categories.

(Q4) Who are the developers of those apps?

Apps developed by the same entities may share similar age verification methods, which can help identify common strengths or weaknesses across different adult-only apps. To this end, we analyzed all app developers and found that many are already experienced in creating adult apps. For example, the developers, *OrderYOYO*, have developed more than 80 adult-only apps. Additionally, we noticed a high concentration of adult-only apps in the Finance and Shopping categories, particularly from developers like *OrderYOYO* and *City Hive Inc*. Further investigation revealed that these developers focus significantly on adult-oriented retail services, primarily selling products such as alcohol and tobacco. Figure 5 shows the developers who have created more than two adult-only apps. Interestingly, we observed that the same developers often exhibit consistent user interface designs and workflows across their apps. For instance, City Hive Inc. has developed numerous adult-oriented apps focused on retail services, such as alcohol and tobacco sales. To streamline development and ensure consistency, they frequently reuse components and templates throughout their product portfolio.

(Q5) How many of these apps have implemented age validation mechanisms?

As shown in Table 7, our results show that there are only 1,165 (3.75%) adult-only apps that have implemented the age verification. These results were determined by identifying apps that implemented UI elements that accept confirmation and concrete inputs, that instruct the user on types of concrete input, and that display output that contains age verification keywords. We also identified apps that implemented age verification through code pattern matching to identify app code that indicates age verification logic. The overall presence of age verification is relatively low across most categories and rating ranges. For example, apps with very high download numbers (*100,000+*) still show a low percentage of age verification, which is concerning given their wide reach and influence. Particularly, we found that certain categories "Finance" and "Business" show higher percentages of age verification apps across rating and download ranges. We infer that these apps target a more mature audience, typically adults engaged in financial activities (gambling) or business operations (selling alcohol). Meanwhile, they are often subject to stringent regulations to prevent fraud, money laundering, and to protect users from financial exploitation. These regulations often mandate robust age verification mechanisms to ensure that financial services are not accessible to minors. However, we do not observe any particular verification mechanism being used exclusively for a specific category or reason determining an app's adult-only status. Also, Apps with IAPs show slightly higher percentages of age verification, which might be due to the financial implications of ensuring that purchases are made by appropriate age groups.

Without effective age verification, there is no reliable way to ensure that users providing sensitive personal information are of the intended age group, and the minors' personal data could be collected, stored, or even misused without adequate safeguards. Such shortcomings amplify privacy concerns, as younger users are less likely to grasp the implications of sharing their data and are more susceptible to exploitation. Notably, the collection of personal information from minors is subject to stringent regulations designed to protect their privacy and safety such as federal law and New York's child privacy law. Therefore, as shown in Table 8, we also display the types of personal data collected by adult-only apps across various categories. This is achieved by checking the strings

	Ratings															Downloads														w/ IAP		
	[0.0,1.0)			[1.0,2.0)			[2.0,3.0)			[3.0,4.0)			[4.0,5.0)			0 - 100			100 - 1,000			1,000-10,000			10,000-100,000			100,000+					
Category	**V**	**#**	**%**	**V**	**#**	**%**	**V**	**#**	**%**	**V**	**#**	**%**	**V**	**#**	**%**	**V**	**#**	**%**	**V**	**#**	**%**	**V**	**#**	**%**	**V**	**#**	**%**	**V**	**#**	**%**	**V**	**#**	**%**
Lifestyle	0	9	0	2	59	3.39	7	288	2.43	43	1055	4.08	3	91	3.3	13	206	6.31	26	860	3.02	31	1350	2.3	44	1160	3.79	40	663	6.03	35	725	4.83
Other	0	1	0	0	1	0	2	21	9.52	0	28	0	1	2	50	2	8	25	2	16	12.5	0	42	0	1	43	2.33	1	37	2.7	1	36	2.78
Productivity	0	7	0	2	38	5.26	6	180	3.33	11	281	3.91	0	13	0	9	95	9.47	5	165	3.03	8	270	2.96	13	317	4.1	6	279	2.15	10	250	4
Travel & Local	0	7	0	0	13	0	0	24	0	1	55	1.82	0	6	0	2	43	4.65	1	64	1.56	2	124	1.61	4	97	4.12	2	42	4.76	3	100	3
Education	0	4	0	0	9	0	0	21	0	1	68	1.47	0	5	0	3	64	4.69	0	69	0	7	124	5.65	5	136	3.68	3	86	3.49	4	71	5.63
Finance	0	1	0	2	13	15.38	2	47	4.26	3	102	2.94	0	7	0	1	63	1.59	4	102	3.92	2	138	1.45	5	132	3.79	3	125	2.4	0	47	0
Business	1	4	25	0	5	0	4	37	10.81	4	49	8.16	0	5	0	5	104	4.81	2	156	1.28	5	148	3.38	5	87	5.75	1	41	2.44	3	19	15.79
Food & Drink	0	5	0	1	24	4.17	0	69	0	11	144	7.64	0	9	0	13	562	2.31	21	1016	2.07	27	646	4.18	22	284	7.75	20	136	14.71	3	59	5.08
News & Books	0	5	0	0	26	0	7	167	4.19	14	472	2.97	1	25	4	7	47	14.89	15	114	13.16	9	307	2.93	12	490	2.45	9	427	2.11	15	486	3.09
Health & Sports	1	8	12.5	2	30	6.67	3	118	2.54	7	278	2.52	0	13	0	6	69	8.7	8	152	5.26	8	265	3.02	12	294	4.08	7	248	2.82	13	365	3.56
Entertainment	1	44	2.27	6	223	2.69	57	1295	4.4	107	3133	3.42	5	98	5.1	33	600	5.5	77	1313	5.86	82	2044	4.01	74	2210	3.35	98	3289	2.98	131	3489	3.75
Social	0	53	0	16	351	4.56	37	1188	3.11	40	1289	3.1	2	51	3.92	18	361	4.99	51	772	6.61	44	1632	2.7	58	1948	2.98	59	1954	3.02	71	2913	2.44
Shopping	0	0	0	1	26	3.85	2	89	2.25	15	252	5.95	6	50	12	8	639	1.25	12	1144	1.05	26	457	5.69	26	256	10.16	22	151	14.57	1	10	10
Photography	0	4	0	1	22	4.55	2	66	3.03	6	156	3.85	0	5	0	1	33	3.03	10	47	21.28	4	85	4.71	1	130	0.77	2	171	1.17	3	101	2.97

Table 7: Distribution of age verification across app categories, ratings, downloads, and in-app purchases. "V" indicates whether the app has implemented age verification mechanisms, "#" represents the number of apps, and "%" shows the percentage. Due to incomplete metadata (e.g., download and ratings) in Androzoo, the total count adds up to less than the total number.

Category	**Fullname**	**Address**	**City**	**Country**
Entertainment	302	156	84	72
Social	170	79	47	32
Lifestyle	118	69	35	34
Food and Drink	95	53	19	34
Shopping	92	45	27	18
News and Books	40	16	9	7
Productivity	33	18	11	7
Health and Sports	26	12	5	7
Finance	17	5	4	1
Photography and Editors	15	8	4	4
Business	14	7	4	3
Education	13	7	3	4
Travel and Local	9	2	0	2
Other	4	2	2	0

Table 8: The types of personal data collected by adult apps across various categories

of UI components co-located in the same layouts. These apps require this information to ensure that the person interacting with them can provide the necessary details to prove they are a real person. It can be observed that entertainment apps have the highest data collection, with 302 instances of full names, 154 instances of dates of birth, and 156 instances of addresses. In contrast, business and finance apps collect minimal data compared to other categories. The extensive collection of personal data raises significant privacy concerns.

(Q6) What are the age verification mechanisms used across different categories?

Different age verification methods can be identified based on distinct code features. Age verification conducted locally is categorized as an age gate. Similarly, entering ID information with purely local validation constitutes a template-based check. If the ID is transmitted to a remote server for validation, it is classified as online ID verification. Credit card verification can be identified through specific keywords and the use of regular expressions, while document (ID) verification is detectable through file upload APIs. Biometric verification poses a greater challenge; however, we observed that many biometric-based age verification mechanisms share consistent code characteristics, such as leveraging Amazon [5] or Azure Face APIs [35], which include age range validation functionalities. This analysis allows us to generate a distribution of verification methods across various categories.

It can be observed from Figure 6 that age gate emerges as the most implemented method (31.84%), especially in categories like Entertainment and Lifestyle. Biometric Verification (8.48%) are the least utilized methods, as shown by the lower bars across all categories. Despite the accuracy and robustness, the method involves higher costs, making it less appealing to developers. Interestingly, we also notice that Finance, Business and other categories that involving payments show the most consistent use of online ID and credit card checks, highlighting regulatory pressure. Age gate and template-based checks, being the weakest age verification methods, are commonly employed in less mission-critical apps, such as those in the entertainment, shopping, lifestyle, and social categories.

Interestingly, we found that the "17+ rating" by Google does not accurately reflect the age requirements set by many app developers. Despite being rated as 17+, 152 apps actually enforce an age limit of 21 years (based on the collected strings), which is higher than the rating suggests. We also found that 309 apps set the bar at 16 years. Lowering the age verification bar can result in younger users accessing content that may not be suitable for their age group, potentially exposing them to inappropriate or harmful material. Meanwhile, our analysis revealed that all 21+ apps relied solely on the weakest Age Gate verification, requiring only self-declared age. In contrast, 63% of 16+ apps used stronger methods like credit card or ID upload, likely due to stricter regulatory

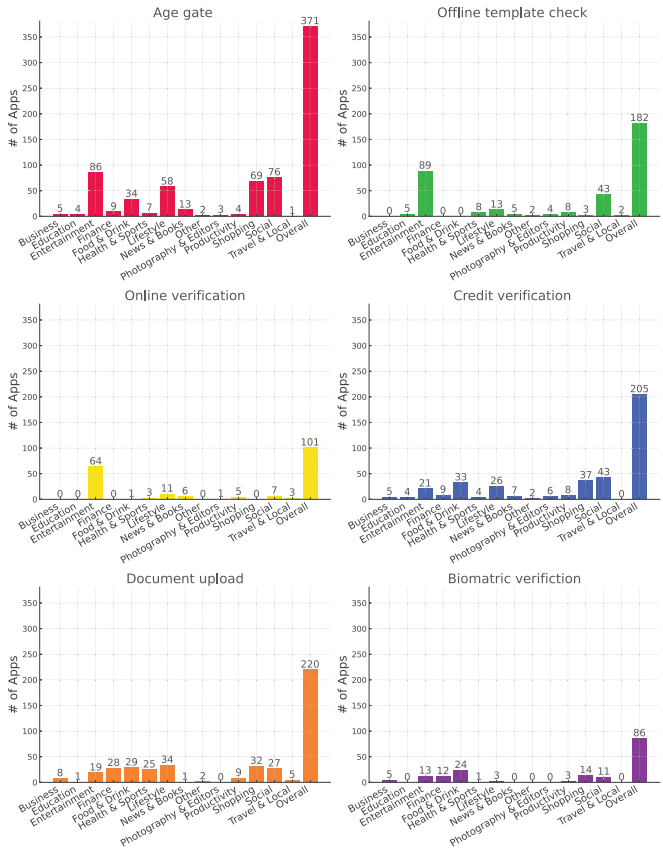

Figure 6: Distribution of age verification methods by category.

scrutiny for younger audiences.

> **(Q7)** Which types of discussed attacks are effective against these adult apps with age verification?

We theoretically analyzed which verification methods might be compromised by specific attack types (§6): age gates (31.84%) were conclusively vulnerable to A1, template-based checks to A2 (15.62%), and all other methods (e.g., online ID verification, accounting for 8.67%, Credit Card Verification, accounting for 17.59%, and document uploading, accounting for 18.88%) except strict biometric authentication (8.48%) to A4. To detect vulnerabilities to A3, we examined whether apps implemented OAuth (32.53%), as its APIs are publicly accessible, and determined that apps using OAuth could potentially be exploited by A3. However, we could not evaluate A5 and A6 due to their dependency on server-side implementations (e.g., whether login IP geo-restrictions are enforced), as probing servers is both unethical and beyond the scope of Android app analysis.

In terms of categories, as shown in Figure 7, Entertainment, and Social categories exhibit significantly higher numbers of

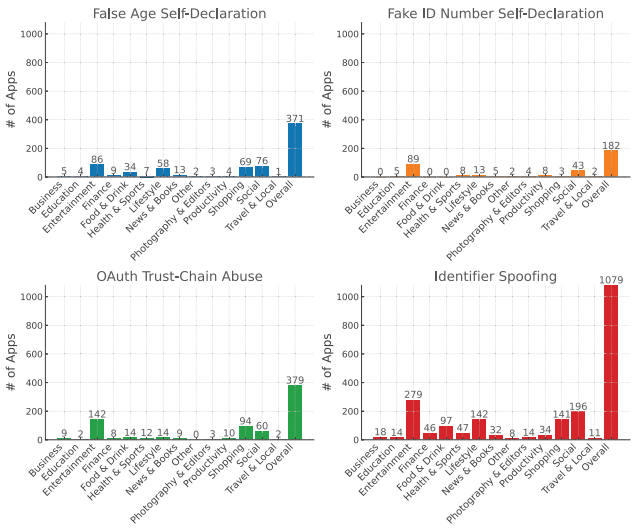

Figure 7: Distribution of attacks across categories.

apps affected by all attack types compared to others. The analysis highlights that the "Entertainment" category dominates across all attack types, with 142 cases of OAuth Trust-Chain Abuse and 279 cases of Identifier Spoofing, far outpacing other categories like "Food & Drink" (14 and 97 cases, respectively) and "Finance" (8 and 46 cases, respectively). In contrast, "Business" and "Education" exhibit significantly lower vulnerabilities, with occurrences ranging between 2 and 18 cases across all attacks Similarly, "Fake ID Number Self-Declaration" shows 89 cases in Entertainment, while "Finance" and "Food & Drink" report none. This suggests that attackers prioritize categories with larger user bases, such as "Entertainment," which accounts for over 60% of Identifier Spoofing cases and 70% of OAuth Trust-Chain Abuse cases. The high prevalence of attack vulnerabilities in the Entertainment category suggests a lack of robust age verification measures, likely due to their lower mission-critical nature and a preference for prioritizing user accessibility over stringent security. Finance and Business categories have relatively fewer apps vulnerable to attacks. These categories likely face stricter regulatory requirements, which compel developers to implement more robust age verification mechanisms, especially where financial transactions are involved.

8 Discussion

Mitigation. Mitigating vulnerabilities in age verification mechanisms for adult-oriented apps requires a multi-faceted approach that balances security, usability, and scalability. App developers can implement strategies such as Multi-Factor Verification (MFV), which combines methods like ID checks

with biometric authentication, and Real-Time Database Verification to authenticate IDs against government databases. Additional measures include IP Address Verification to block VPN-based attacks, Behavioral Analytics to detect age fraud through user interaction monitoring, and Ongoing Age Verification during critical interactions to maintain compliance. Developers should also enforce consistent age verification across all apps within an ecosystem and conduct regular security updates to address emerging threats. From Google's perspective, stricter developer guidelines, dynamic app testing to identify bypass vulnerabilities, and leveraging user feedback to address ineffective verification mechanisms are critical. Google has committed to adopting dynamic testing and user feedback measures, collaborating with stakeholders to enhance the integrity of age-restricted apps. Please refer to Appendix C for a detailed discussion.

Limitations. Our detection tool is not perfect and may be subject to limitations. First, the accuracy of detecting age verification mechanisms heavily relies on taint analysis, which might not be perfect and could miss certain data flows, especially in complex applications with obfuscated code. Second, our dynamic analysis may not explore all possible paths in some apps, particularly those that require valid input to access deeper or more varied paths. However, we argue that age verification checks are typically positioned early in the app flow and do not depend on extensive user input. These checks are unlikely to be deeply embedded within the app. Our reasoning is based on the nature of adult apps: if a user reaches deeper states without encountering an age verification check, the check becomes ineffective, as it implies the user—potentially a child—has already been actively using the app. This observation is supported by our FP/FN analysis. Furthermore, UI exploration is only necessary in rare cases where static analysis fails to identify age verification mechanisms. Third, since we can only collect the client-side code, the remote end of the app is treated as a black box in our analysis, meaning attacks involving the manipulation of back-end services may not be accurately detected. For example, Attack A5 is feasible because the back-end services do not enforce uniform age verification across different countries, and the feasibility of Attack A4 (OAuth abuse) depends on whether age verification mechanisms are consistent across different apps. While our results provide a useful reference, they may not be as definitive as those for A1–A3. However, this limitation applies universally to all tools analyzing back-end services, as actively probing servers and handling diverse traffic to ensure back-end functionality raises ethical concerns. Additionally, our static sampling and analysis of the apps revealed zero false positives, further supporting the reliability of our analysis. Finally, the tool is specifically designed for mobile applications, particularly Android apps, and may not be directly applicable to web-based platforms or iOS apps. However, we argue that the underlying principles remain the same.

9 Related Work

Age Verification. There have been several efforts focused on age verification mechanisms for adult-only apps, primarily from policy and regulatory perspectives [11, 26, 34, 36, 38, 39, 41, 55]. For example, one study reviewed the age verification mechanisms in 10 social apps (e.g., Instagram, TikTok, Skype, Facebook) [55]. The researchers manually opened these apps and checked their compliance with policies such as GDPR. However, this approach is insufficient, as technical solutions are necessary to better understand and protect minors. A few papers have focused on improving the performance of age verification mechanisms [14, 24, 27, 29]. These solutions include auditory perception-based methods [24], audio-visual-based methods [29], and facial 3D-based methods [27]. Unlike these efforts, our work is the first to systematically analyze Android adult apps on Google Play and investigate potential vulnerabilities.

Verification Mechanisms in Apps. There have also been efforts focused on the verification mechanisms enforced by Android [10, 13, 31–33, 44, 45, 52, 54]. For example, Hamid et al. [10] detected flaws in the permission verification system. VeriDroid [32] exposed potential defects in Android apps. Wenbo et al. [52] used program analysis to detect flaws in in-app payment verification. Haoyu et al. [45] identified security issues in the signature verification process of Android apps. ACID [33] verified Android API compatibility issues by selecting and executing relevant tests from an app's test suite. While previous works have focused on various verification mechanisms, our paper specifically addresses age verification mechanisms in adult apps.

10 Conclusion

Our paper has delved into the enforcement of age verification mechanisms within adult-only apps and introduced GUARD as a novel solution. Our comprehensive analysis of 31,750 adult-only apps on Google Play revealed a startlingly low implementation rate of age verification, with only 3.67% incorporating such mechanisms. Furthermore, even these mechanisms can be easily bypassed with minimal technical skills, posing significant risks. Our work underscores the inadequacies of current age verification methods in adult-oriented apps and highlights the urgent need for more robust countermeasures.

Ethics Considerations

It's crucial to address ethical concerns meticulously to ensure that our research adheres to ethical standards and does not inadvertently cause harm or violate privacy. First, we did not use researchers' personal data for dynamic analysis,

as our dynamic analysis operates without requiring any inputs, However, personal data was used to test the verification mechanisms. Importantly, in our study, we only use our own personal data to test the verification mechanisms and do not use or store any personal data from other users (although as discussed, other users information can be easily accessed on the internet). To be more specific, when dynamically testing potential attacks, we analyzed two scenarios involving the input of ID numbers or other sensitive information. The first scenario is the false ID declaration attack, where a randomly generated false ID is used. The second scenario involves ID spoofing, which requires the input of a real ID. For this, we used our own personal data. We ensured that all team members reviewed, understood, and consented to the use of their data, in alignment with the privacy policies of the platforms involved. We have completed the required Responsible Conduct of Research training and other related certifications through CITI. We uphold the highest ethical standards when testing these apps and take every precaution to ensure that no one is harmed during our research. Meanwhile, all sensitive information, such as ID numbers and ID card photos, is securely stored and anonymized to prevent any potential leaks. Second, ethical guidelines require that no harm comes to the users or systems being tested. We strictly limit our testing to our own accounts and avoid any actions that could be construed as hacking or attacking the apps. Finally, we reported our findings to Google Play, as we believe Google should take responsibility for ensuring that apps comply with age verification requirements, given that the standards are set by Google. Google acknowledged our findings and expressed gratitude for our efforts. They noted that the specific implementation of age verification measures can vary based on the type of content and the developer's chosen approach, making it challenging to verify age authentication comprehensively. Google stated their plans to review the app, test whether its age verification can be bypassed, and rely on user feedback to identify complaints regarding age-inappropriate content or ineffective age verification mechanisms. We will continue contacting and working with Google to minimize the harm.

Open Science Policy

We have made our code publicly available in a repository https://zenodo.org/records/14688696.

References

[1] 24kkkkkkk. Mt-arknights. https://github.com/24kkkkkk/MT-Arknights/tree/master, 2024. Accessed: 2024-06-04.

[2] Alibaba. Taobao. https://www.taobao.com. Accessed: 2024-06-03.

[3] Alibaba. Xianyu. https://2.taobao.com. Accessed: 2024-06-03.

[4] Kevin Allix, Tegawendé F Bissyandé, Jacques Klein, and Yves Le Traon. Androzoo: Collecting millions of android apps for the research community. In *Proceedings of the 13th international conference on mining software repositories*, pages 468–471, 2016.

[5] Amazon Web Services. Amazon rekognition. https://aws.amazon.com/rekognition/. Accessed: 2024-12-31.

[6] Amnesty International. Global: Tiktok's 'for you' feed risks pushing children and young people towards harmful mental health content. https://amnesty.org/en/latest/news/2023/11/tiktok-risks-pushing-children-towards-harmful-content/, 2024. Accessed: 2024-05-30.

[7] AppInChina. China implements government system for real-name login. https://appinchina.co/blog/china-implements-government-system-for-real-name-login/, 2021. Accessed: 2024-06-03.

[8] Appium. Appium: Mobile app automation made awesome. http://appium.io. Accessed: 2024-06-02.

[9] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. *ACM sigplan notices*, 49(6):259–269, 2014.

[10] Hamid Bagheri, Eunsuk Kang, Sam Malek, and Daniel Jackson. Detection of design flaws in the android permission protocol through bounded verification. In *FM 2015: Formal Methods: 20th International Symposium, Oslo, Norway, June 24-26, 2015, Proceedings 20*, pages 73–89. Springer, 2015.

[11] Scott Brennen and Matt Perault. Keeping kids safe online: How should policymakers approach age verification? *The Center for Growth and Opportunity*, 2023.

[12] Public Safety Canada. Online child sexual exploitation: Booklet for parents and caregivers of kids aged 10-17. https://www.canada.ca/en/public-safety-canada/campaigns/online-child-sexual-exploitation/booklet-for-parents-and-caregivers-of-kids-aged-10-17.html. Accessed: 2024-05-31.

[13] Yongliang Chen, Ruoqin Tang, Chaoshun Zuo, Xiaokuan Zhang, Lei Xue, Xiapu Luo, and Qingchuan Zhao. Attention! your copied data is under monitoring: A systematic study of clipboard usage in android apps.

In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*, pages 1–13, 2024.

[14] Sahar Dammak, Hazar Mliki, and Emna Fendri. Face age verification for access control application. In *Fourteenth International Conference on Machine Vision (ICMV 2021)*, volume 12084, pages 157–163. SPIE, 2022.

[15] DigiChina. Translation: Cybersecurity law of the people's republic of china (effective june 1, 2017). https://digichina.stanford.edu/work/translation-cybersecurity-law-of-the-peoples-republic-of-china/, 2017. Accessed: 2024-05-30.

[16] Emma Henderson Vaughan. One third of missing children enticed online are recovered in a different state: New analysis. https://www.missingkids.org/blog/2024/online-enticement-new-analysis-blog, 2024. [Online; accessed 30-May-2024].

[17] Federal Communications Commission. Children's internet protection act (cipa). https://www.fcc.gov/consumers/guides/childrens-internet-protection-act. Accessed: 2024-05-30.

[18] Daniel Fett, Ralf Küsters, and Guido Schmitz. A comprehensive formal security analysis of oauth 2.0. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 1204–1215, 2016.

[19] Getnewidentity.Com. Random chinese id number generator. https://www.getnewidentity.com/chinese-id-number-generator.php. Accessed: 2024-06-03.

[20] Google. Android developer documentation. https://developer.android.com/docs, 2024. Accessed: 2024-12-29.

[21] Australian Government. Online safety act 2021. https://www.legislation.gov.au/C2021A00076/latest/text, 2021. Accessed: 2024-05-31.

[22] ICLG. Cybersecurity laws and regulations japan 2024. https://iclg.com/practice-areas/cybersecurity-laws-and-regulations/japan. Accessed: 2024-05-30.

[23] ICLG. Digital business laws and regulations france 2024. https://iclg.com/practice-areas/digital-business-laws-and-regulations/france. Accessed: 2024-05-30.

[24] Muhammad Ilyas, Alice Othmani, Régis Fournier, and Amine Nait-Ali. Auditory perception based anti-spoofing system for human age verification. *Electronics*, 8(11):1313, 2019.

[25] Korean Legislation Research Institute. Youth protection act. https://elaw.klri.re.kr/eng_service/lawViewTitle.do?hseq=8716. Accessed: 2024-05-31.

[26] Ahmad Jamaludin. Age verification regulation in social media platform usage: Preventive measures against online child sexual violence. *Jurnal Bina Mulia Hukum*, 8(2):276–295, 2024.

[27] Marie Jandová, Marek Daňko, and Petra Urbanová. Age verification using random forests on facial 3d landmarks. *Forensic Science International*, 318:110612, 2021.

[28] Ben Jiang. Chinese social media platforms weibo, wechat and douyin's real-name authentication rule for influencers to benefit operations amid tightened online regulation, analysts say. https://www.scmp.com/tech/policy/article/3239986/chinese-social-media-platforms-weibo-wechat-and-douyins-real-name-authentication-rule-influencers, 2023. Accessed: 2024-05-30.

[29] Pavel Korshunov and Sébastien Marcel. Face anthropometry aware audio-visual age verification. In *Proceedings of the 30th ACM International Conference on Multimedia*, pages 5944–5951, 2022.

[30] IT Media Law. Interstate treaty on the protection of minors in the media (jmstv). https://itmedialaw.com/en/wissensdatenbank/interstate-treaty-on-the-protection-of-minors-in-the-media-jmstv/. Accessed: 2024-05-30.

[31] Chongqing Lei, Zhen Ling, Yue Zhang, Kai Dong, Kaizheng Liu, Junzhou Luo, and Xinwen Fu. Do not give a dog bread every time he wags his tail: Stealing passwords through content queries (CONQUER) attacks. In *30th Annual Network and Distributed System Security Symposium, NDSS 2023, San Diego, California, USA, February 27 - March 3, 2023*. The Internet Society, 2023.

[32] Yepang Liu and Chang Xu. Veridroid: Automating android application verification. In *Proceedings of the 2013 Middleware Doctoral Symposium*, pages 1–6, 2013.

[33] Tarek Mahmud, Meiru Che, and Guowei Yang. Detecting android api compatibility issues with api differences. *IEEE Transactions on Software Engineering*, 2023.

[34] Christine Marsden. Age-verification laws in the era of digital privacy. *Nat'l Sec. LJ*, 10:210, 2023.

[35] Microsoft Corporation. Azure face api. https://azure.microsoft.com/en-us/services/cognitive-services/face/. Accessed: 2024-12-31.

[36] Clare Morell and John Ehrett. Age verification: Policy ideas for states. https://eppc.org/publication/age-verification-policy-ideas-for-states/. Accessed: 2024-06-03.

[37] International Association of Privacy Professionals (IAPP). How brazil regulates children's privacy and what to expect under the new data protection law. https://iapp.org/news/a/how-brazil-regulates-childrens-privacy-and-what-to-expect-under-the-new-data-protection-law/. Accessed: 2024-05-31.

[38] Liliana Pasquale and Paola Zippo. A review of age verification mechanism for 10 social media apps. *CyberSafe Ireland*, May 2020.

[39] Liliana Pasquale, Paola Zippo, Cliona Curley, Brian O'Neill, and Marina Mongiello. Digital age of consent and age verification: Can they protect children? *IEEE Software*, 39(3):50–57, 2020.

[40] Semmle. Codeql: Querying code as data. https://codeql.github.com/, 2024. Accessed: 2024-12-24.

[41] Xiangbo Shu, Guo-Sen Xie, Zechao Li, and Jinhui Tang. Age progression: Current technologies and applications. *Neurocomputing*, 208:249–261, 2016.

[42] Veratad Technologies. Louisiana act 440 age verification faqs: Staying compliant. https://veratad.com/blog/louisiana-act-440-age-verification-faqs-staying-compliant, 2024. Accessed: 2024-12-21.

[43] Raja Vallée-Rai, Phong Co, Etienne Gagnon, Laurie Hendren, Patrick Lam, and Vijay Sundaresan. Soot: A java bytecode optimization framework. In *CASCON First Decade High Impact Papers*, pages 214–224. 2010.

[44] Chao Wang, Yue Zhang, and Zhiqiang Lin. Uncovering and exploiting hidden apis in mobile super apps. In Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda, editors, *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, CCS 2023, Copenhagen, Denmark, November 26-30, 2023*, pages 2471–2485. ACM, 2023.

[45] Haoyu Wang, Hongxuan Liu, Xusheng Xiao, Guozhu Meng, and Yao Guo. Characterizing android app signing issues. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 280–292. IEEE, 2019.

[46] Wikipedia. Children's code. https://en.wikipedia.org/wiki/Children%27s_Code. Accessed: 2024-05-30.

[47] Wikipedia. Information technology rules, 2021. https://en.wikipedia.org/wiki/Information_Technology_Rules,_2021. Accessed: 2024-05-31.

[48] Wikipedia. Proposed uk internet age verification system. https://en.wikipedia.org/wiki/Proposed_UK_Internet_age_verification_system. Accessed: 2024-05-31.

[49] Wikipedia. Protection of young persons act (germany). https://en.wikipedia.org/wiki/Protection_of_Young_Persons_Act_(Germany), 2003. Accessed: 2024-05-30.

[50] World Health Organization. One in six school-aged children experiences cyberbullying: New who/europe study. https://www.who.int/europe/news/item/27-03-2024-one-in-six-school-aged-children-experiences-cyberbullying--finds-new-who-europe-study, 2024. Accessed: 2024-05-30.

[51] Biwei Yan, Kun Li, Minghui Xu, Yueyan Dong, Yue Zhang, Zhaochun Ren, and Xiuzhen Cheng. On protecting the data privacy of large language models (llms): A survey. *arXiv preprint arXiv:2403.05156*, 2024.

[52] Wenbo Yang, Yuanyuan Zhang, Juanru Li, Hui Liu, Qing Wang, Yueheng Zhang, and Dawu Gu. Show me the money! finding flawed implementations of third-party in-app payment in android apps. In *NDSS*, 2017.

[53] Yifan Yao, Jinhao Duan, Kaidi Xu, Yuanfang Cai, Zhibo Sun, and Yue Zhang. A survey on large language model (llm) security and privacy: The good, the bad, and the ugly. *High-Confidence Computing*, page 100211, 2024.

[54] Yue Zhang, Yuqing Yang, and Zhiqiang Lin. Don't leak your keys: Understanding, measuring, and exploiting the appsecret leaks in mini-programs. In Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda, editors, *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, CCS 2023, Copenhagen, Denmark, November 26-30, 2023*, pages 2411–2425. ACM, 2023.

[55] Paola Zippo and Liliana Pasquale. 2019 technical report: A review of age verification mechanism for 10 social media apps. *Technological University Dublin*, 2019.

A Algorithm for Dynamic Age Verification

As shown in Algorithm 1, the dynamic age verification analysis systematically explores Android applications to identify age verification mechanisms by simulating user interactions and analyzing UI components. It begins by launching the app and extracting the XML structure of the initial UI, which contains details about UI elements such as buttons, text fields,

and labels. The algorithm iteratively examines each element, searching for keywords such as *"age," "birthdate," or "verify"* that suggest relevance to age verification. When a relevant element is found, its identifier and type are recorded for further analysis, and the current UI analysis terminates. For elements with click functionality, the algorithm simulates clicks to navigate to new screens, recursively analyzing each newly loaded interface. This dynamic and recursive process ensures thorough exploration of all navigable paths within the app, uncovering hidden or dynamically loaded components.

Algorithm 1 Dynamic Age Verification Analysis

procedure ANALYZEAPP(A)
  LAUNCH(A)
  ANALYZEAPPUI($A \rightarrow UI_{1st}$)
end procedure
function ANALYZEAPPUI(UI)
  $XML \leftarrow$ ExtractCurrentInterface(UI)
  for each $ele \in XML$ **do**
    if SEARCHKEYWORDS(ele) **then**
      RECORD($ele \rightarrow ID$, $ele \rightarrow type$)
      return
    end if
    $clickables \leftarrow$ findClickables(ele)
    for each $c \in clickables$ **do**
      $UI_i \leftarrow$ CLICK(c)
      ANALYZEAPPUI(UI_i)
    end for
  end for
end function

B Code Snippet

B.1 Code Snippet of *Dating Wallet*

This code snippet is part of a login selection UI for an Android application, supporting third-party login methods such as Xiaomi and QQ accounts. It sets UI texts to guide the user in selecting a login method, dynamically assigns IDs to views using their `hashCode`, and applies layout parameters for proper sizing. Resource utilities are used to fetch assets or configurations for the respective login methods. Additionally, the `a()` method includes a dynamic analysis or proxy-checking mechanism, likely part of a framework for hotfixes or method interception, which skips execution if certain conditions are met. This implementation connects to the provided OAuth security concerns, where inconsistent age verification mechanisms can be exploited.

B.2 Code Snippet of *Peeks Social*

As shown in Figure 10, *Peeks Social*, a dating app, requires users to upload a photo of their ID for age verification. The

```
 this.d.setText("Please select your login method");
LinearLayout.LayoutParams linearLayout$LayoutParams0 = new
↪ LinearLayout.LayoutParams(-2, -2);
miTextView1.setId(miTextView1.hashCode());
this.e.setText("Log in with Xiaomi Account");
ResourceUtils.c(this.getContext(),
↪ "mio_login_third_account_mi_pure")
miTextView2.setId(miTextView2.hashCode());
this.e.setText("Log in with QQ Account");
ResourceUtils.c(this.getContext(),
↪ "mio_login_third_account_qq_pure")
...
public void a() {
    if(PatchProxy.proxy(new Object[0], this,
    ↪ LoginSelectLoginAccount.changeQuickRedirect, false,
    ↪ 0x206, new Class[0], Void.TYPE).isSupported) {
        return;}}
```

Figure 8: The code snippet of app *Dating Wallet*

app extracts information such as the user's name and zip code from the photo to verify their age. However, this method has vulnerabilities. For instance, the app's reliance on photo uploads and basic data extraction makes it susceptible to attacks. Minors could potentially manipulate or forge ID photos to bypass the verification process, compromising the app's security and effectiveness in preventing underage access.

```
this.n.takePicture(null, null, this.u);
if(bundle0 != null) {
     Intent intent0 = new Intent(this,
     ↪ WithdrawStepAlmost.class);
     intent0.putExtra("firstname",
     ↪ bundle0.getString("firstname"));
           intent0.putExtra("lastname",
           ↪ bundle0.getString("lastname"));
     intent0.putExtra("zipcode",
     ↪ bundle0.getString("zipcode"));
     ...

     if(this.s != null) {
     intent0.putExtra("imagepath", this.s);}
    }
    //Upload all the infomation and Photo ID
    if(!this.B) {
        l.a().a(this, "Uploading...", 60000L, false,
        ↪ 15027);
        com.peeks.app.mobile.f.c.a().r()
        .a(com.peeks.app.mobile.f.c.a().i()
        .getUser_id(), this.r, this.p.getCode(),
        ↪ this.q.getAbsolutePath(), null, this.t);
            return;
        }
```

Figure 9: The code snippet of app *Peeks Social*

B.3 Code Snippet of *MT-Arknights*

The provided code snippet illustrates a function, request, within the *MT-Arknights* tool, designed to bypass age verification and payment restrictions in the game Arknights through a MitM proxy. The function intercepts HTTP requests between the client and server, targeting specific endpoints related to age verification and payment processes. It iterates through a list (`fklist`) of targeted URLs, checks if the request host

```
def request(self, flow: HTTPFlow):
    for cgi in self.fklist:
        if cgi in flow.request.url and "biligame.net" in
        ↪ flow.request.host and not self.FirstLogin:
            ttime = time.strftime("%Y-%m-%d %H:%M:%S",
            ↪ time.localtime())
            print("[%s]Intercepted age verification request:
            ↪ %s" %
                  (ttime, flow.request.url))
            flow.kill()
    if "api/client/can_pay" in flow.request.url and
    ↪ "biligame.net" in flow.request.host:
        ttime = time.strftime("%Y-%m-%d %H:%M:%S",
        ↪ time.localtime())
        print("[%s]Modified payment restriction" %
              (ttime, flow.request.url)))
```

Figure 10: The code snippet of attacking tool *MT-Arknights*

is "`biligame.net`," and if the user is not logging in for the first time, it terminates the age verification request using `flow.kill()` to prevent it from reaching the server. Additionally, it identifies requests involving payment restrictions (e.g., containing "`api/client/can_pay`") and modifies them to bypass these limitations, logging all actions for reference. This approach aligns with how MitM tools operate: intercepting, logging, and altering network traffic to exploit insufficiently secured communication protocols. By terminating or modifying critical requests, the tool effectively circumvents server-side checks, enabling minors to bypass age restrictions or payment barriers with minimal effort.

C Mitigation

For app developers, addressing the vulnerabilities identified in age verification mechanisms for adult-oriented apps requires the adoption of robust strategies to enhance security while maintaining usability. As outlined in Table 9, the following mitigation strategies can be implemented either individually or in combination to create a tailored approach that balances these critical aspects. Particularly, we advocate for a tailored approach rather than imposing a one-size-fits-all countermeasure, recognizing that different applications have distinct security requirements.

(M1) Multi-Factor Verification (MFV): We can implement MFV to add an additional layer of security by combining multiple verification methods. For example, an ID check can be paired with biometric authentication, such as fingerprint or facial recognition, to enhance the reliability of age verification processes.

(M2) Real-Time Database Verification: We should verify user-provided information against real-time government databases to ensure the authenticity of IDs and other documents. This helps in detecting fake IDs.

(M3) IP Address Verification: We can use geolocation services to detect and block IP addresses associated with VPN services. This can defend against attacks such as age verification downgrade.

	(M1)	(M2)	(M3)	(M4)	(M5)	(M6)	(M7)
(A1) False Age	✔	✔	✗	✗	✗	✗	✗
(A2) False ID	✔	✔	✗	✗	✗	✗	✗
(A3) OAuth Abuse	✗	✗	✗	✗	✗	✔	✗
(A4) ID Spoofing	✗	✗	✗	✔	✔	✗	✗
(A5) Age Downgrade	✗	✗	✔	✗	✗	✗	✗
(A6) Tool Based	✗	✗	✗	✗	✗	✗	✔

Table 9: Summary of Mitigation

(M4) Behavioral Analytics: We can implement behavioral analytics to monitor user interactions and detect anomalies that might indicate age fraud or misuse of the verification process.

(M5) Ongoing Age Verification: We should continuously verify the user's age at regular intervals or during critical interactions (e.g., whenever users access age-restricted content) to ensure ongoing compliance.

(M6) Consistent Enforcement across Apps: We should ensure consistent enforcement of age verification across all applications within the OAuth ecosystem to prevent bypassing through third-party logins.

(M7) Regular Security Updates: We should conduct regular security audits and updates to address new vulnerabilities and threats, and to ensure that the age verification of apps are up-to-date with the latest security standards and technologies.

From Google's perspective, mitigating the challenges of age verification in apps requires robust measures that balance security, usability, and scalability. First, Google can establish stricter guidelines for developers to ensure the proper implementation of age verification mechanisms. Second, they can introduce dynamic app testing during the review process to simulate real-world attempts to bypass these mechanisms, removing apps that fail to meet the required standards. Lastly, they can rely on user feedback to identify issues related to age-inappropriate content or ineffective verification, taking prompt action based on these reports. Following our discussion with Google, they have committed to adopting the second and third solutions to address this issue, and we will collaborate with them to minimize potential harm.