EdinBurg Peer Reviewed Journals and Books Publishers
Journal of Information and Technology
Vol. 5||Issue 5||pp 34-48||June||2025
Email: info@edinburgjournals.org||ISSN: 3080-9576

HIPAA Technical Safeguards: How Well Do MHealth Applications Comply?

Dr. KN Jonathan[1*]
University of Lay Adventists of Kigali
Corresponding Email: phialn1@gmail.com

Accepted: 05 June 2025 || Published: 19 June 2025

Abstract

The rise of mobile health technology, or mHealth apps, necessitates the protection of individual health details. These digital platforms enable users to save, share, and access their medical data, monitor health issues, and manage treatments easily. As mHealth app usage increases, it's critical to ensure that protected health information (PHI) is securely transmitted, received, created, and maintained per the Health Insurance Portability and Accountability Act (HIPAA) guidelines. Unfortunately, many mHealth app developers lack a comprehensive understanding of HIPAA security and privacy requirements. This gap presents an opportunity to develop an analytical framework that aids programmers in creating secure, HIPAA-compliant code and educates users about PHI security and privacy. The proposed plan involves creating a framework to build an integrated development environment (IDE) plugin for developers and a web-based interface for consumers. This will help developers identify and address HIPAA compliance issues during the development process and provide consumers with a tool to evaluate mHealth app privacy and security before use. The goal is to promote the development of secure, compliant mHealth apps that protect personal health information.

Keywords: *HIPAA, mHealth, Android Apps, Privacy & Security, IDE plugin*

How to Cite: Jonathan, K. N. (2025). HIPAA Technical Safeguards: How Well Do Mhealth Applications Comply? *Journal of Information and Technology, 5*(5), 34-48.

1. Introduction

mHealth apps collect, process, store, and transmit sensitive user data and health records, making security crucial. A U.S. study found that over half of doctors recommend medical apps to patients, with 58% having downloaded one. During emergencies like COVID-19, mHealth app usage increased, with patients using them for tasks like paying bills, scheduling appointments, and accessing medical records.

The rise in mHealth app usage raises concerns about data security and privacy. The Health Insurance Portability and Accountability Act (HIPAA) mandates that mHealth apps must implement safeguards to protect electronic health information. However, many developers lack understanding of these requirements, leading to potential security flaws. A survey showed that 84% of FDA-approved medical apps had security issues.

To address these concerns, mHealth apps must ensure secure data transmission and transparent data handling practices. With over 300,000 mHealth apps available and many healthcare providers experiencing data breaches, assessing HIPAA compliance is essential. Cyberattacks on healthcare institutions, especially during the COVID-19 pandemic, highlight the importance of strong security measures.

HIPAA requires entities to have procedures to detect, respond to, and recover from cyberattacks. Traditional security solutions are often inadequate, necessitating advanced methods. Many mHealth app developers are unfamiliar with HIPAA regulations, presenting an opportunity to develop tools to help them ensure compliance.

We propose "HIPAA Checker," a framework to analyze mHealth app source code for HIPAA compliance. Our study of 285 mHealth apps showed the effectiveness of HIPAA Checker in identifying security issues.

The goals of this research are to among others; Create a framework to evaluate HIPAA compliance in mHealth apps, Ensure secure data communication between third-party apps and health record systems, Identify risks and safety features, and finally Develop an IDE plugin to provide developers with feedback on security and privacy issues early in development.

1.1 Background

Administrative, Physical, and Technical security requirements are outlined in the Health Insurance Portability and Accountability Act (HIPAA). Administrative safeguards are rules and guidelines that control the choice, creation, application, and upkeep of security measures.

Reference	Rule Name	Technical Safeguards
164.312(a)(1)	Authorization	Implement technological policies and procedures to restrict access to individuals or software programs that have been given access privileges for electronic information systems that maintain EPHI.
164.312(a)(2)(i)	Unique Id	Assign a unique name or number to each patient in order to identify and monitor their identification.
164.312(a)(2)(ii)	Emergency EPHI Access	Create and use processes for acquiring required digitally protected health information in an emergency.
164.312(a)(2)(iii)	Automatic Session timeout	Implement software procedures that end a session after a certain period of inactivity.

164.312(a)(2)(iv)	EPHI Encryption and Decryption	Implement a system for encrypting and decrypting EPHI.
164.312(b)	EPHI Audit Control	Implement methods for recording and examining activities in information systems that use or include EPHI.
164.312(c)(1)	EPHI Data Integrity	Implement regulations and procedures to prevent unauthorized manipulation or destruction of EPHI.
164.312(c)(2)	EPHI Integrity Verification	Utilize technological tools to verify that electronically stored protected health information has not been tampered with or deleted without authorization.
164.312(d)	EPHI Authentication	Establish processes to confirm that the individual or organization requesting access to EPHI is who is being identified.
164.312(e)(1)	EPHI Transmission Security	Implement technological security measures to prevent unauthorized access to digitally protected health information that is being sent through a network of electronic communications.
164.312(e)(2)(i)	EPHI Transmission Integrity	Implement security measures to guarantee that electronically transmitted protected health information is not improperly altered up to disposal without being noticed.
164.312(e)(2)(ii)	Appropriate EPHI Encryption	Implement a mechanism to encrypt EPHI whenever deemed appropriate.

Physical safeguards involve methods, guidelines, and practices designed to protect equipment from environmental threats and unauthorized access. In contrast, technical safeguards are policies and technology-based procedures that protect electronically protected health information (EPHI) from unauthorized access. Proposed methods for analyzing mHealth app source code focus on addressing technical safeguards issues. By ensuring compliance with

technical safeguards, other administrative and operational safeguards, such as tools for reviewing and monitoring security features, can be met, preventing issues like non-compliance with physical safeguards. For example, retrieving encrypted PHI from a lost or stolen cell phone would be very difficult.

The paper briefly compares "HIPAAChecker" with other tools available on the market, but a more in-depth comparison would help emphasize the unique advantages of "HIPAAChecker." Below is a table that outlines the features and limitations of various tools for assessing HIPAA compliance in mHealth apps:

Tool	HIPAA Focus	Security Features	Usability	Limitations
FindBugs	General	Identifies common security flaws in Java code	High, integrated with Android Studio	Does not specifically focus on HIPAA criteria
DexGuard	Limited	Focuses on code obfuscation and anti-tampering	High, with many customization options	Lacks HIPAA-specific security analysis
TrustKit	Limited	Provides SSL pinning to prevent man-in-the-middle attacks	Moderate, requires manual integration	Does not cover HIPAA compliance checks
HIPAAChecker	Comprehensive	Automated HIPAA compliance checks, source code analysis, audit logging, encryption verification	High, easy to integrate into Android Studio IDE	Focused on technical safeguards, may need manual audit for administrative safeguards.

HIPAAChecker Comprehensive Automated HIPAA compliance checks, source code analysis, audit logging, encryption verificationHigh, easy to integrate into Android Studio IDE Focused on technical safeguards, may need manual audit for administrative safeguards.

"HIPAAChecker" stands out for its comprehensive approach to analyzing HIPAA technical safeguards. It not only identifies standard security vulnerabilities but also focuses on HIPAA-specific concerns such as the protection of EPHI, audit control, and transmission security. Unlike other tools that primarily deal with general security threats, "HIPAAChecker" is designed to address the legal and compliance needs of the healthcare industry, making it uniquely valuable for mHealth app developers.

Interpretation of Figures in Textual Format

The analysis presented in the paper references several figures illustrating the results of HIPAA compliance testing. Table 2, which shows the percentage of apps that met HIPAA technical safeguards, reveals that only 40% of the tested apps implemented adequate transmission security. This means that more than half of the mHealth apps tested were at risk of leaking sensitive health information during data transmission. Similarly, Figure 9, which depicts the

percentage of code segments that complied with HIPAA regulations, highlights that approximately 45% of apps lacked sufficient audit controls to monitor access to EPHI.

Medical apps, compared to fitness apps, demonstrated higher levels of compliance across the board. Figure 10 shows that nearly 70% of medical apps adhered to HIPAA standards, as opposed to just 30% of fitness apps. This discrepancy underscores the importance of applying rigorous compliance standards to all types of mHealth apps, not just those directly related to medical care.

2. Related Work

To ensure the confidentiality, integrity, and availability of electronic health information, the HIPAA security rule, which took effect in April 2005, requires administrative, physical, and technical measures. mHealth apps need to be secured to protect patient and healthcare data. A recent study categorized security threats for mobile health apps into three levels: high (for monitoring, diagnosis, and care), medium (calculators, localizers, and alarms), and low (informative and educational apps). The American Health Information Management Association (AHIMA) advises on managing mobile health data breaches, including checking privacy settings, looking for certifications, using passwords and encryption, and avoiding texting sensitive health information.

Most vulnerabilities in mobile health apps should be addressed to mitigate security and privacy issues. Evaluating and testing source code according to HIPAA's security and privacy criteria is essential before allowing the use of these apps. A study showed that while only two of the top 20 mHealth apps required user authentication, 65% asked for personal information like name, address, email, and date of birth. Data breaches are a concern, especially for the 50% of apps storing data in the cloud and the 65% sharing data with third parties without user consent. Only 20% of apps informed users about data privacy and security measures.

Authors have suggested using static security analysis methods with the FindSecurityBugs IDE plugin for Android Studio, which helps developers secure their apps and reduce security risks. A review of available tools as of March 2023 found that no frameworks or tools specifically verify mHealth app security against HIPAA standards for EPHI. While tools like FindBugs, IntelliJ IDE, and Eclipse IDE help maintain and clean code, they do not focus on HIPAA security criteria. Recent mobile security tools like DexGuard and TrustKit do not address HIPAA compliance.

3. HIPAA Checker Framework

This study introduces a framework for analyzing Android mHealth apps that handle patient data and adhere to HIPAA technical security standards. Unlike existing tools focused on Java-specific security, this framework automatically assesses app source code to identify common security and privacy issues in mHealth apps. The framework's architecture is shown in Figure 1, and its features are compared with other tools in Figure 2. The plugin tool helps mHealth developers detect and address HIPAA compliance issues before app release through Android Studio. Users can also submit APK files to check for security vulnerabilities. Figure 3 illustrates the meta-analysis process for web users and developers.

EdinBurg Peer Reviewed Journals and Books Publishers
Journal of Information and Technology
Vol. 5||Issue 5||pp 34-48||June||2025
Email: info@edinburgjournals.org||ISSN: 3080-9576

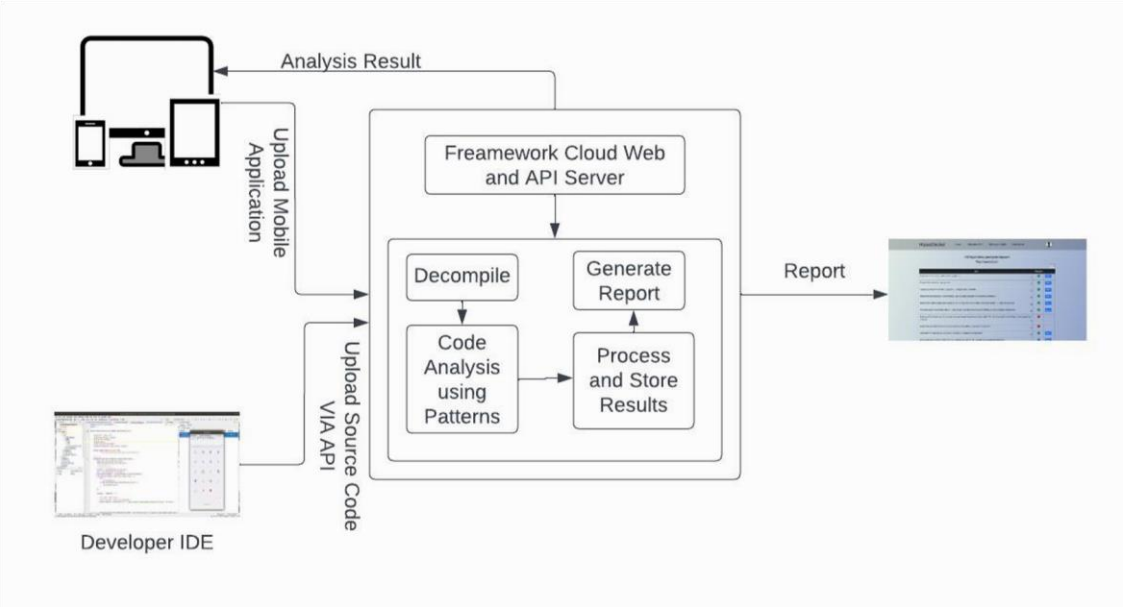

Figure 1: HIPAA checker Framework Architecture

The framework utilizes JADX [21] to decode manifest files and other resources and to decompile dex files into Java classes, as Android apps are usually written in Java, compiled into dex format, and run on the Android virtual machine. A source code analyzer, incorporating patterns shown in Table 2, is integrated into the framework's cloud server. This analyzer identifies specific HIPAA-related code vulnerabilities and generates reports for users. These reports highlight the exact lines of code that need to be revised to achieve HIPAA compliance.

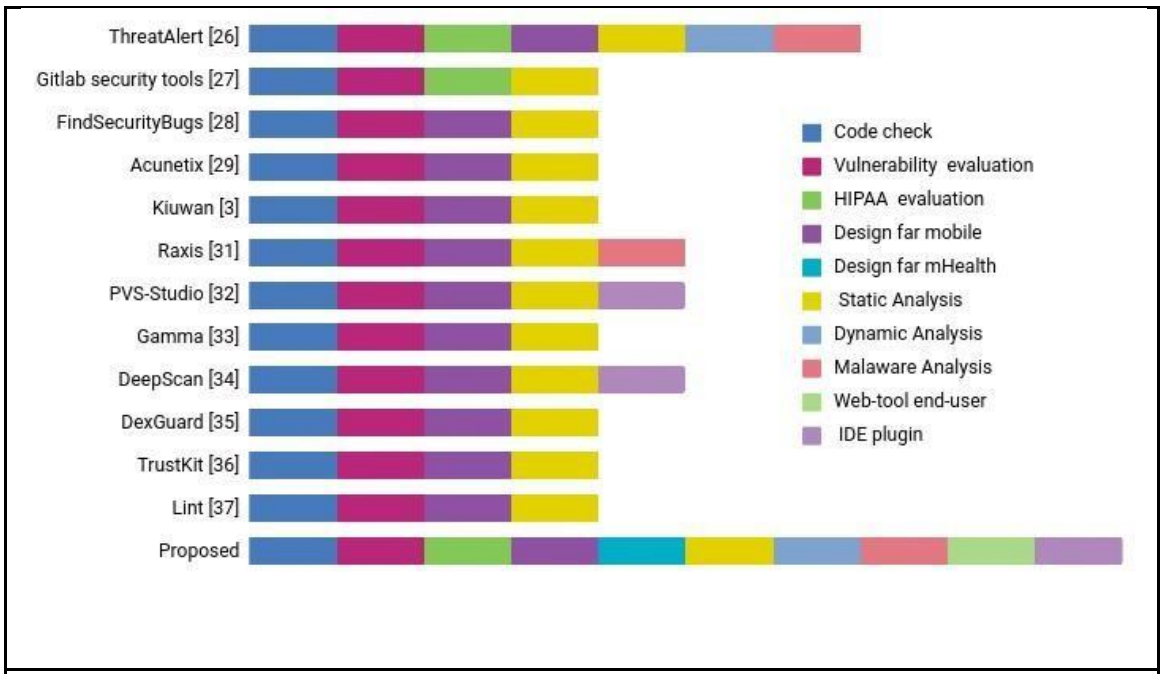

Figure 2: Comparison of the features of the proposed framework with those of similar products on the market

EdinBurg Peer Reviewed Journals and Books Publishers
Journal of Information and Technology
Vol. 5||Issue 5||pp 34-48||June||2025
Email: info@edinburgjournals.org||ISSN: 3080-9576

Overall, the proposed framework [55] will give mHealth app developers a methodical way to check that their applications meet HIPAA's technical security criteria, thereby improving the privacy and security of patient data.

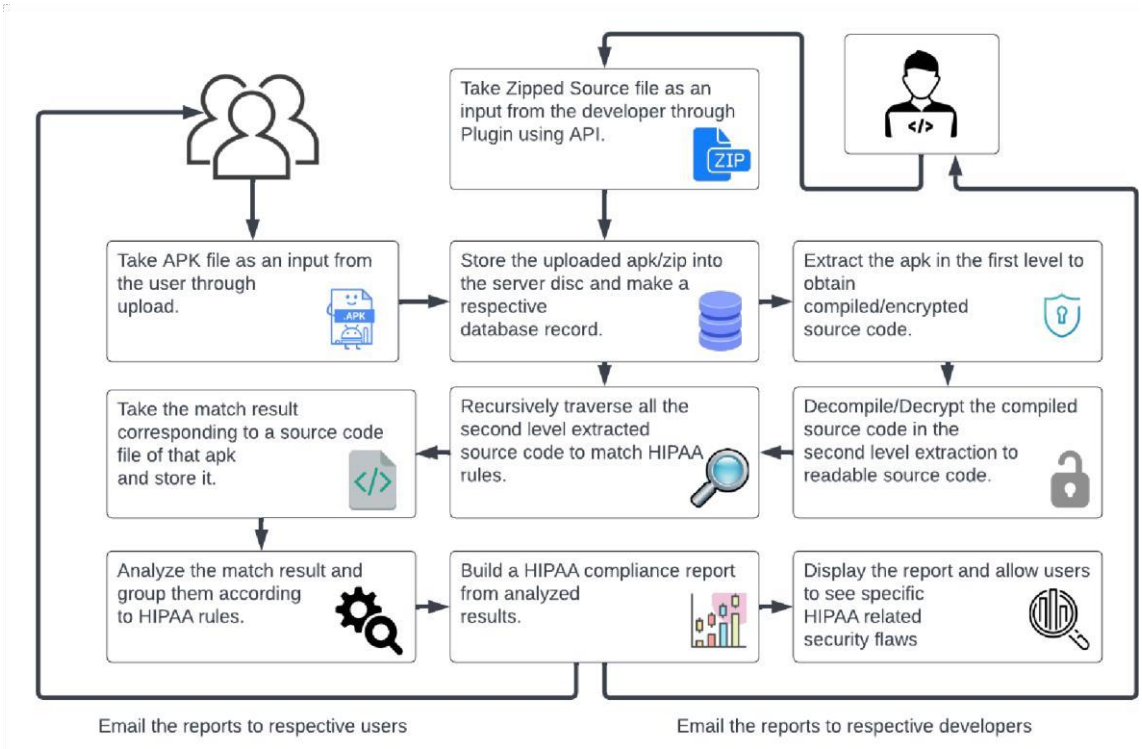

Figure 3: HIPAA Analysis Flow

EdinBurg Peer Reviewed Journals and Books Publishers
Journal of Information and Technology
Vol. 5||Issue 5||pp 34-48||June||2025
Email: info@edinburgjournals.org||ISSN: 3080-9576

Rule ID	Sub Rule ID	Detection Code Patterns
EPHI_encryption_decryption	EN-DE	➢ import java.util Base64
	AES	➢ import org.springframework.security.crypto ➢ import java.security.Security ➢ Cipher.getInstance("AES/ECB/ ➢ Cipher.getInstance("AES") ➢ Cipher.getInstance(AES_MODE ➢ new SecretKeySpec(keyBytes, "AES" ➢ Cipher.getInstance("AES/CBC/
	DES	➢ Cipher.getInstance(.*DES ➢ Cipher.getInstance(.*des
	RSA	➢ Cipher.getInstance("RSA
	BLOWFISH	➢ .getInstance(.*BLOWFISH
	RC	➢ .getInstance(.*RC2 ➢ .getInstance(.*rc4 ➢ .getInstance(.*RC4, .getInstance(.*rc2
	Message Digest	➢ MessageDigest import ➢ java.security.MessageDigest ➢ .getInstance(.*MD5 ➢ .getInstance(.*md5 ➢ DigestUtils.md5(➢ import org.apache.commons.codec.digest.Digest Utils;
	SHA	➢ .getInstance(.*SHA-1 ➢ .getInstance(.*SHA1 ➢ DigestUtils.sha(
	ECB	➢ Cipher.getInstance(\s*"\s*AES\/ECB

EdinBurg Peer Reviewed Journals and Books Publishers
Journal of Information and Technology
Vol. 5||Issue 5||pp 34-48||June||2025
Email: info@edinburgjournals.org||ISSN: 3080-9576

	HMAC	➢ import org.apache.commons.codec.digest.Hmac Algorithms import org.apache.commons.codec.digest.HmacUtils
EPHI_Transmission_integrity	TRANS-NET	➢ javax.net.ssl.TrustManager ➢ TrustManagerFactory.getInstance(
Appropriate_ EPHI_Encryption	DE	➢ android.util.Base64 ➢ .decodeToString ➢ .decode
	EN	➢ android.util.Base64 ➢ .encodeToString, .encode
	ENCRYPT	➢ io.realm.Realm ➢ .encryptionKey(
	Chiper	➢ Net.sqlcipher. ➢ AS encrypted KEY
Authorization	Authorization Control	➢ AuthorizationException
	Access Control	➢ IllegalAccessException
EPHI_Transmission_Security	API	➢ addRequestProperty(\"Authorization
	PKIX	➢ PKIXRevocationChecker
	TRANS-Data	➢ HttpsURLConnection new
Unique_Id	PK	➢ PRIMARY KEY
EPHI_authentication	FireBaseAuth	➢ FirebaseUser ➢ sendFirebasePropertyRegisteredUser ➢ FirebaseUserPropertiesSender ➢ Com.google.firebase\:firebase-auth ➢ FirebaseAuth
	aAuth	➢ android.accounts.AccountManager ➢ AccountManager.get(, .currentUser

EdinBurg Peer Reviewed Journals and Books Publishers
Journal of Information and Technology
Vol. 5||Issue 5||pp 34-48||June||2025
Email: info@edinburgjournals.org||ISSN: 3080-9576

Automatic_Session_Timeout	Inactivity	➢ public void onUserInteraction() ➢ .reset()
		➢ .clear() ➢ .commit()
EPHI_Audit_Control	Audit	➢ AppOpsManager.OnOpNotedCallback
EPHI_integrity_verification	authorization_exception_on_destroy	➢ AuthorizationException
	illegal_destruction_restriction	➢ IllegalAccessException
EPHI_data_integrity	authorization_exception	➢ AuthorizationException
	illegal_access	➢ IllegalAccessException
	user_authentication _oauth	➢ android.accounts.AccountManager ➢ AccountManager.get(
	user_authentication _firebase	➢ FirebaseUser ➢ sendFirebasePropertyRegisteredUser ➢ FirebaseUserPropertiesSender Com.google.firebase\:firebase-auth ➢ FirebaseAuth

Table 3: HIPAA rules-based meta-analysis techniques

Formal language is utilized to describe the vulnerability pattern that corresponds to HIPAA requirements instead of natural language, and the HIPAA technical safeguard patterns are formally described as Equation 1. The complete list of pattern-based matching for meta-analysis is described in Table 3.

H=(r, d, s, v, p)

Here,

“r” represents the rule reference

“d” represents the detection process

“s” represents sub-rules of HIPAA Technical Safeguard

“v” represents vulnerability information or vulnerability evidence “p” represents the patterns of HIPAA compliance.

4. Framework Development

We developed a web-based platform for conducting HIPAA compliance experiments. We used agile methodology to create the HIPAA Checker application, which allowed us to create it quickly and with the most up-to-date technologies. We carefully selected the technologies we used to ensure that the web application and plugin were dependable, efficient, and secure. We chose Ruby as the backend programming language for the web application and Ruby on Rails as the web application framework. Ruby on Rails is well-known for its ability to speed up the development process and enable developers to write clean, maintainable code. For the cloud server, we also used Amazon Web Services (AWS) Elastic Compute Cloud (EC2), which provides scalable and reliable cloud computing resources. We used PostgreSQL, a powerful and dependable open-source database management system, for database management. To manage changes to the source code, the popular version control system Git was used.

We used HTTPS and two-factor authentication (2FA) for additional security to ensure secure communication. While 2FA adds an extra layer of security to user authentication by requiring a second form of authentication in addition to the password, HTTPS encrypts data in transit to prevent unauthorized access. User authentication and authorization were handled by Devise, a versatile and adaptable authentication solution. To ensure the application's quality, we used RSpec and Capybara for unit and integration testing, which allowed us to find and address problems early in the development process. The application was automated for deployment using Capistrano and Docker, which offered a reliable and effective way to do so with zero downtime. On the front end of the web application, we used Bootstrap for responsive design, creating a user-friendly and aesthetically pleasing interface. With the aid of Webpack, we were able to effectively manage and bundle our JavaScript and CSS files. ActionCable was used for real-time websockets, enabling the web application to react to user actions instantly. Additionally, we developed RESTful APIs with token-based authentication and API authentication using JSON Web Tokens (JWT) for third-party integration. We used Jbuilder in the API to efficiently serialize data so that it could be used in the API. Our team integrated the plugin with Android Studio using the IntelliJ Platform SDK. Git was also used for version control, and RESTful APIs were used to communicate with the plugin while JSON Web Tokens (JWT) were used for authentication and authorization. We tested the plugin in various scenarios using an Android emulator or a physical Android device, and we used OkHttp to make HTTP requests.

5. Testing and Evaluation

We conducted a thorough investigation to assess the potential risk of HIPAA violations by downloading 285 mHealth apps from the Google Play Store and Github's Medical and Health & Fitness categories. The apps were chosen based on specific features and functionalities related to the storage, processing, management, and transfer of Electronic Protected Health Information (EPHI). We also considered the privacy policies, terms of service, and collection techniques of the selected apps from various geographical locations.

EdinBurg Peer Reviewed Journals and Books Publishers
Journal of Information and Technology
Vol. 5||Issue 5||pp 34-48||June||2025
Email: info@edinburgjournals.org||ISSN: 3080-9576

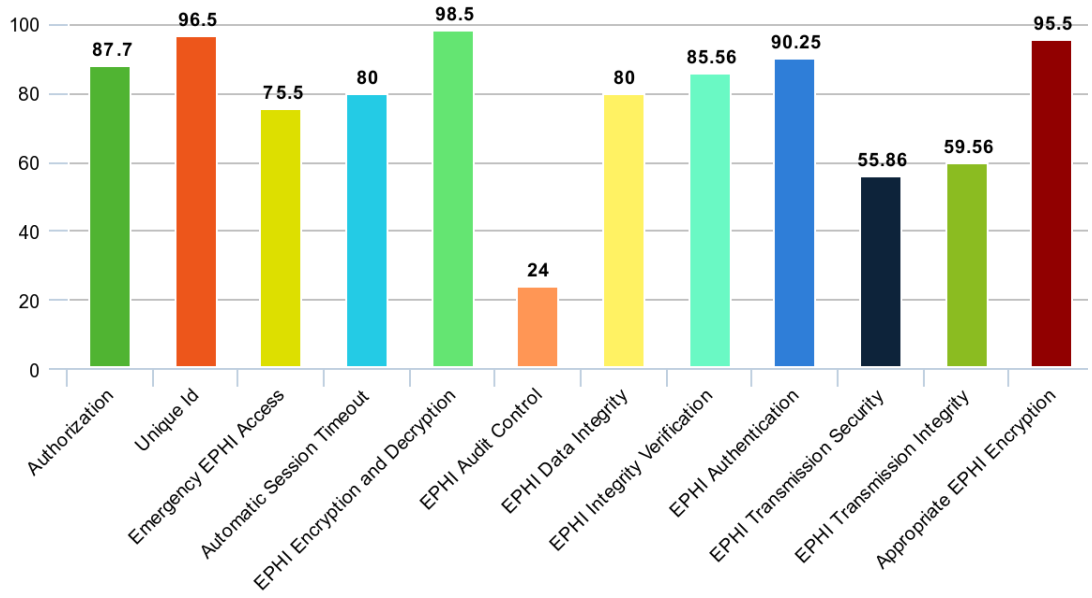

Figure 4: Percentage of apps that met the HIPAA technical safeguards

Our primary goal was to assess the potential risk of HIPAA violations and determine whether the selected mHealth apps adhere to the act's standards and regulations. To ensure a thorough analysis, we chose both top-rated downloaded apps (10M+ downloads) and low-rated apps (100+ downloads). The Google Play Store applications were tested using our developed web application, and the Github open-source repositories were tested using our developed Integrated Development Environment (IDE) plugin.

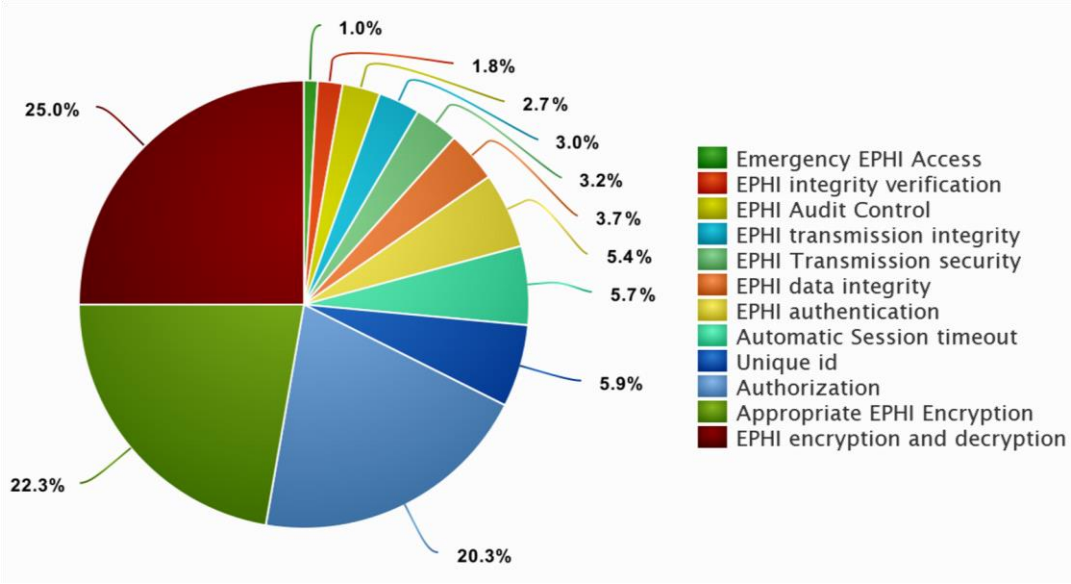

Figure 5: Percentage of code segments detected in all downloaded apps that comply with HIPAA regulations

Our investigation revealed that a significant percentage of the downloaded apps lacked appropriate audit control or transmission security measures, potentially resulting in HIPAA

violations. We discovered, in particular, that nearly half of the mHealth apps lacked appropriate audit controls to track and monitor user access to EPHI. Furthermore, approximately 40% of the apps lacked appropriate transmission security measures to protect EPHI from unauthorized access while in transit. The majority of the apps, on the other hand, included adequate authorization, unique user identification, and encryption/decryption mechanisms for EPHI. We discovered that more than 90% of the apps used unique user identification to authenticate user access to EPHI and that more than 80% of the apps used EPHI encryption/decryption mechanisms to protect sensitive health information.

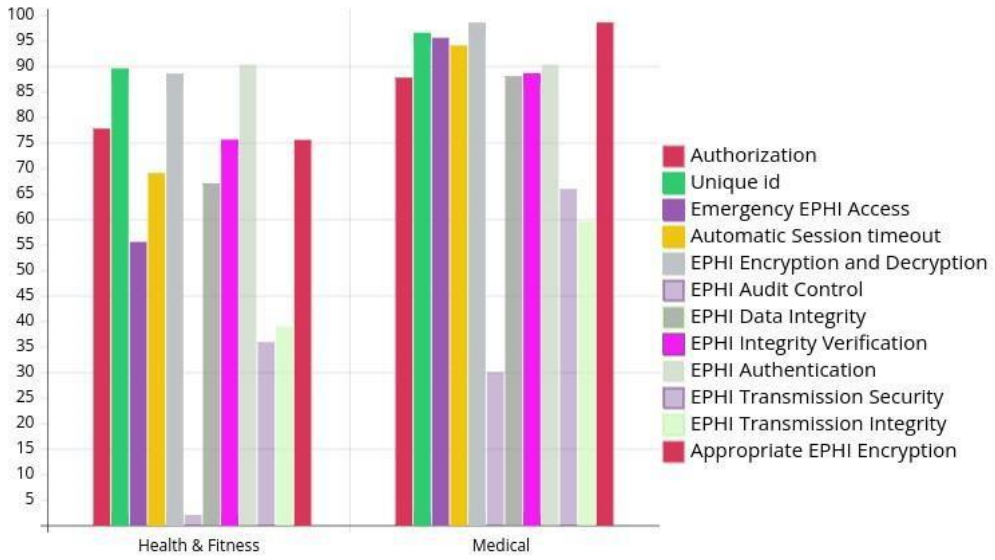

Figure 6: Percentage of apps that met the HIPAA technical safeguards with respect to apps categories

Our findings are presented graphically, with Figures 8, 9, and 10 illustrating the findings of our investigation. Furthermore, we discovered that medical applications were more HIPAA-compliant than health and fitness apps. This could be due to the nature of medical apps that handle sensitive patient information, as well as the requirement for more stringent security measures to comply with HIPAA regulations. Our investigation sheds light on the potential risks of HIPAA violations posed by mHealth apps. According to our findings, many mHealth apps lack appropriate audit control or transmission security measures, which could lead to HIPAA violations.

6. Conclusion

The use of mHealth applications is widespread, yet many of them have security and privacy flaws and don't adhere to HIPAA Technical Safeguard requirements. Our developed HIPAA Checker can solve this issue by spotting the absence of technical security measures in both released and under-development applications. By guaranteeing that applications are in compliance with HIPAA regulations, this framework seeks to increase the trust of application users. Using our tool, developers can find and fix any security or privacy flaws in their applications. By incorporating HIPAA safety measures, the healthcare and fitness industry can improve the security of sensitive EPHI and build trust between patients and healthcare providers.

7. Recommendations

It is evident that there is a great need for advancements in the security and privacy procedures of healthcare apps. Our analysis report shows that the main hazards of these applications are lack of audit control, unsecured information sharing with third-party APIs, and unauthorized access to critical resources. We propose the following actions: Recommendations for Application Users: Before sharing sensitive health information, thoroughly read the application's review and privacy policy. Look for signs of HIPAA compliance, such as adherence to privacy standards and adequate security measures. Before sharing personal health information, use tools to validate HIPAA compliance. This can assist guarantee that your data is handled correctly and securely. Examine and critique programs in order to educate others and aid researchers and developers in developing the app successfully. Giving input may assist in identifying and addressing any security issues, as well as improving overall user privacy. Recommendations for Application Developers: Implement audit measures to allow for a full investigation of all incidents. This can aid in identifying and addressing any security flaws before they are exploited. When integrating external APIs, employ SSL to ensure safe data delivery. Use suitable access control mechanisms to guarantee that sensitive EPHI is only accessed by authorized personnel. This can aid in the prevention of unwanted access and the protection of user privacy. Use cutting-edge encryption and decryption technology to protect sensitive information from unauthorized access and ensure data security. To safeguard sensitive information from unwanted access and assure data security, use cutting-edge encryption and decryption technologies. Users and developers may collaborate to enhance the security and privacy standards of healthcare applications by adopting these guidelines. It is critical that all stakeholders engaged in the development and usage of healthcare apps take responsibility for guaranteeing the safety and security of personal health information.

References

1. M. R. Mia, H. Shahriar, M. Valero, N. Sakib, B. Saha, M. A. Barek, M. J. H. Faruk, B. Goodman, R. A. Khan, and S. I. Ahamed, “A comparative study on HIPAA technical safeguards assessment of android mhealth applications,” Smart Health, vol. 26, p.100349, 2022.
2. B. Pieper, “An overview of the HIPAA security rule, part ii: Standards and specifications.” Optometry (St. Louis, Mo.), vol. 75, no. 11, pp. 728–730, 2004.
3. F. Zubaydi, A. Saleh, F. Aloul, and A. Sagahyroon, “Security of mobile health (mHealth) systems,” in 2015 IEEE 15th International Conference on bioinformatics and bioengineering (BIBE). IEEE, 2015, pp. 1–5.
4. E. P. Morera, I. de la Torre D´ıez, B. Garcia-Zapirain, M. L´opezCoronado, and J. Arambarri, “Security recommendations for mhealth apps: Elaboration of a developer’s guide,” Journal of medical systems, vol. 40, pp. 1–13, 2016.
5. B. Pieper, “An overview of the HIPAA security rule, part ii: Standards and specifications.” Optometry (St. Louis, Mo.), vol. 75, no. 11, pp. 728–730, 2004.
6. H. Kharrazi, R. Chisholm, D. VanNasdale, and B. Thompson, “Mobile personal health records: an evaluation of features and functionality,” International Journal of Medical Informatics, vol. 81, no. 9, pp. 579– 593, 2012.

7. R. Adhikari, D. Richards, and K. Scott, "Security and privacy issues related to the use of mobile health apps." ACIS, 2014. [8] find sec bugs. (2023, Apr.) Find security bugs. [Online]. Available: https://find-sec-bugs.github.io/
8. Eclipse. (2023, Apr.) Eclipse ide. [Online]. Available: https://www.eclipse.org/ide/
9. guardsquare. (2023, Apr.) Dexguard. [Online]. Available: https://www.guardsquare.com/en/blog/dexguard-vs-proguard
10. L. Li, T. F. Bissyand´e, M. Papadakis, S. Rasthofer, A. Bartel, D. Octeau, J. Klein, and L. Traon, "Static analysis of Android apps: A systematic literature review," Information and Software Technology, vol. 88, pp. 67–95, 2017.
11. TrustKit. (2023, Apr.) Trustkit. [Online]. Available: https://github.com/datatheorem/TrustKit
12. J. Randolph, M. J. H. Faruk, B. Saha, H. Shahriar, M. Valero, L. Zhao, and N. Sakib, "Blockchain-based medical image sharing and automated critical-results notification: A novel framework," in 2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC), 2022, pp. 1756–1761.