Available online www.jsaer.com

***Journal of Scientific and Engineering Research*, 2021, 8(11):150-160**

Research Article

ISSN: 2394-2630
CODEN(USA): JSERBR

Advanced Penetration Testing for Mobile Applications: Insights and Techniques for Android and iOS

Amit Gupta

San Jose, CA
Email id: gupta25@gmail.com

Abstract This research paper explores the methodologies, tools, and techniques used in penetration testing for mobile applications, specifically focusing on Android and iOS platforms. With the proliferation of mobile applications across various sectors, ensuring their security has become paramount to protect sensitive user data and maintain trust. Penetration testing is a critical process in identifying vulnerabilities, weaknesses, and potential entry points that could be exploited by malicious actors. By conducting thorough penetration tests, organizations can enhance the security posture of their mobile applications and mitigate potential risks. This paper provides an in-depth analysis of the various tools and techniques used in penetration testing for both Android and iOS applications, highlighting their strengths, limitations, and areas of applicability. Furthermore, it compares the effectiveness of these tools and techniques in addressing specific security challenges unique to each platform. This comparative study aims to offer valuable insights and practical guidance for security professionals and developers seeking to implement robust security measures in their mobile applications.

Keywords Mobile Application, Mobile Application Security, Penetration Testing, Pen testing, Android Application Pen testing, iOS application pen testing, Mobile Device Security, Mobile Penetration Testing, Mobile Application Vulnerabilities; Mobile Application Security Testing Framework

Introduction

The rapid growth of mobile applications has made them a prime target for cyber-attacks. As smartphones and tablets become ubiquitous, users increasingly rely on mobile applications for a wide range of services, from banking and shopping to communication and entertainment. This reliance on mobile apps has expanded the attack surface for malicious actors, who exploit vulnerabilities to gain unauthorized access to sensitive data, disrupt services, and cause financial and reputational damage. The increasing complexity and functionality of mobile applications have also introduced new security challenges that require robust and dynamic solutions.

Penetration testing (pentesting) is a proactive approach to identifying and addressing security vulnerabilities in mobile applications. By simulating real-world attack scenarios, pentesters can uncover weaknesses in an application's design, implementation, and deployment. This process involves a comprehensive assessment of an application's security posture, including its codebase, network communications, and backend services. Pentesting not only helps in identifying vulnerabilities but also provides actionable insights for remediation, thus strengthening the overall security framework of mobile applications.

Given the distinct architectures and security models of Android and iOS platforms, effective pentesting requires specialized tools and techniques tailored to each environment. Android, with its open-source nature, offers greater flexibility for analysis but also introduces a higher degree of fragmentation due to the wide variety of devices and OS versions. iOS, on the other hand, benefits from a more controlled ecosystem but poses challenges related to its closed-source nature and stringent security controls.

This paper aims to provide a comprehensive overview of pentesting methodologies, tools, and techniques for Android and iOS platforms, highlighting their strengths and weaknesses. It explores the different phases of pentesting, from planning and reconnaissance to exploitation and reporting, and examines how specific tools

and techniques can be effectively utilized in each phase. By comparing the tools and techniques available for both platforms, this paper seeks to offer valuable insights for security professionals and developers striving to enhance the security of their mobile applications. The goal is to equip them with the knowledge and resources needed to perform thorough and effective penetration tests, ultimately leading to more secure and resilient mobile applications.

Motivation

Penetration testing is essential for mobile applications due to several reasons:

[1]. Increasing Mobile Malware Attacks: With the rise in mobile malware, penetration testing helps in identifying and mitigating vulnerabilities before they can be exploited. According to Symantec's Internet Security Threat Report (2020), mobile malware detections increased by 54% in 2019.

[2]. Data Breaches and Privacy Concerns: Mobile applications often handle sensitive user data, making them prime targets for data breaches. A study by Ponemon Institute (2019) revealed that the average cost of a data breach in the United States was $8.19 million.

[3]. Compliance Requirements: Many industries have stringent regulatory requirements for data protection. Penetration testing helps organizations comply with standards such as GDPR, HIPAA, and PCI-DSS.

[4]. Protecting Brand Reputation: Security incidents can severely damage a company's reputation. Ensuring robust security through penetration testing helps maintain customer trust and brand integrity.

[5]. Evolution of Mobile Application Security Threats: The threat landscape is continuously evolving, with new vulnerabilities and attack vectors emerging regularly. Penetration testing ensures that applications are resilient against the latest threats.

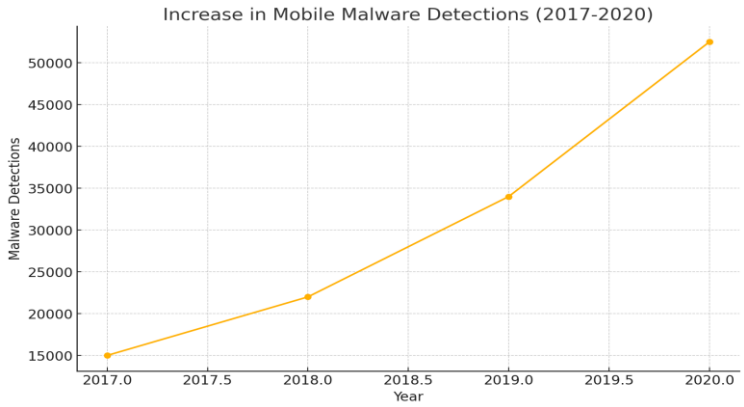

Figure 1: Increase in Mobile Malware Detections (2017-2020)

According to Symantec, mobile malware detections have been increasing consistently, as shown in Figure 1. The graph indicates a sharp rise in the number of detected malware, highlighting the growing threat to mobile applications.

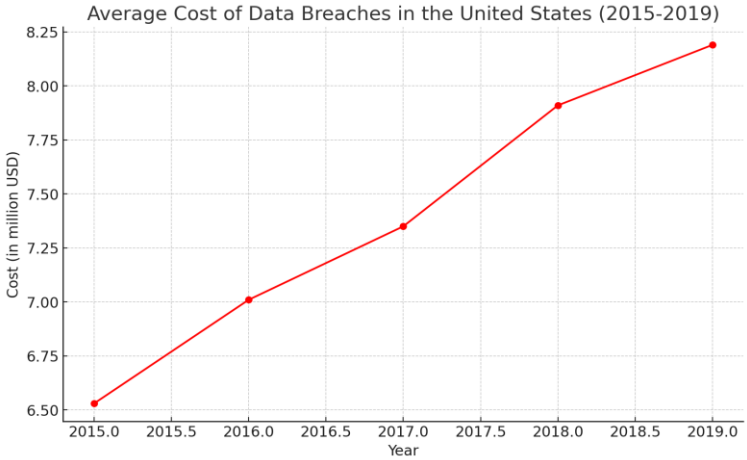

Figure 2: Average Cost of Data Breaches in the United States (2015-2019)

Figure 2 illustrates the average cost of data breaches in the United States over the past five years. The Ponemon Institute's study shows a steady increase in the cost, underlining the financial impact of security incidents on organizations.

Literature Review

The literature on mobile application security emphasizes the growing importance of robust security measures as mobile applications become integral to daily life. The proliferation of smartphones and the increasing reliance on mobile applications for various services—such as banking, communication, and e-commerce—have made them prime targets for cyber-attacks. Comprehensive security assessments are essential to protect sensitive user data and maintain the integrity of these applications.

According to a study by Arxan Technologies (2018), a staggering 92% of the top 100 paid Android apps and 87% of the top 100 paid iOS apps were found to have been hacked. This alarming statistic underscores the widespread vulnerabilities present in even the most popular and seemingly secure applications. The report highlights common security issues such as weak encryption, improper data storage, and inadequate code obfuscation. These vulnerabilities can lead to severe consequences, including unauthorized access to user data, financial loss, and damage to the application's reputation.

The Open Web Application Security Project (OWASP) has published extensive guidelines on mobile security testing, emphasizing the need for continuous security assessment throughout the development lifecycle. The OWASP Mobile Security Testing Guide (MSTG) provides a comprehensive framework for assessing mobile application security, covering aspects such as static analysis, dynamic analysis, and reverse engineering. It also includes a checklist of common vulnerabilities and best practices for mitigating them. This guide serves as a valuable resource for developers and security professionals seeking to improve the security posture of their mobile applications.

The National Institute of Standards and Technology (NIST) released guidelines on mobile application security in 2019, which stress the importance of integrating security into the software development lifecycle (SDLC). NIST's guidelines advocate for a proactive approach to mobile security, starting from the initial design phase through to deployment and maintenance. Key recommendations include conducting regular security assessments, implementing strong authentication and encryption mechanisms, and ensuring secure data storage practices. These guidelines aim to provide a standardized approach to mobile security, helping organizations to safeguard their applications against evolving threats.

Research by Symantec (2020) indicates that mobile malware attacks are on the rise, with a significant increase in sophisticated attacks targeting mobile platforms. The Symantec Internet Security Threat Report reveals that mobile malware detections increased by 54% in 2019, highlighting the growing threat landscape. These attacks often exploit vulnerabilities in mobile applications to gain access to sensitive data, install malicious software, or hijack devices for nefarious purposes. The report emphasizes the need for advanced pentesting techniques and tools to identify and mitigate these threats effectively. Techniques such as static and dynamic analysis, network traffic inspection, and code review are crucial for uncovering hidden vulnerabilities and ensuring robust security measures.

The literature and various breach News consistently highlights the critical importance of robust security measures for mobile applications. Studies and guidelines from Arxan Technologies, OWASP, NIST, and Symantec collectively underscore the need for continuous security assessment and advanced pentesting techniques. These resources provide valuable insights and practical guidance for developers and security professionals, enabling them to enhance the security of their mobile applications and protect against the ever-evolving threat landscape.

Penetration Testing Methodologies

Penetration testing methodologies generally follow a structured approach comprising several phases:

[1]. Planning and Reconnaissance: This phase involves understanding the application's architecture, identifying potential entry points, and gathering information about the application.
[2]. Scanning: Tools are used to analyze the application for vulnerabilities, such as insecure data storage, poor authentication mechanisms, and improper session management.
[3]. Exploitation: The identified vulnerabilities are exploited to determine the extent of potential damage.
[4]. Reporting: Findings are documented, and recommendations for remediation are provided.

Tools and Techniques for Android Penetration Testing

Android applications, being open-source, offer a wide array of tools for pentesting:

A. Static Analysis Tools:

[1]. MobSF (Mobile Security Framework): A comprehensive framework for static and dynamic analysis of Android applications.

[2]. AndroBugs: A tool that detects security vulnerabilities in Android applications' code.

B. Dynamic Analysis Tools:

[1]. Drozer: A comprehensive security testing framework for Android.

[2]. Frida: A dynamic instrumentation toolkit that allows testers to inject scripts into running processes.

C. Network Analysis Tools:

[1]. Burp Suite: A popular tool for analyzing network traffic between the mobile application and backend servers.

[2]. Wireshark: A network protocol analyzer used to capture and inspect the data traveling over a network.

D. Reverse Engineering Tools:

[1]. APKTool: A tool for decompiling and recompiling Android application packages (APKs).

[2]. JD-GUI: A Java decompiler that helps in understanding the application's code structure.

E. Step-by-Step Guide for Android Penetration Testing Using Tools and Techniques

[1]. Planning and Reconnaissance

A. Understanding the Application

[1]. Identify the purpose and functionality of the application.

[2]. Determine the target audience and gather information about the application's architecture.

B. Gathering Information

[1]. Collect information about the app's backend servers, APIs, and third-party services.

[2]. Use tools like Google Dorking to find sensitive information related to the application.

[2]. Static Analysis

A. Setting Up the Environment

[1]. Install Android Studio or a similar IDE to inspect the application code.

[2]. Download the APK file of the application from Google Play Store or an APK repository.

B. Decompiling the APK

[1]. Use APKTool to decompile the APK and convert it to a readable format:

```
bash
apktool d <app_name>.apk
```

C. Analyzing the Decompiled Code

[1]. Use JD-GUI to inspect the decompiled Java code:

[2]. Look for hard coded secrets, API keys, and sensitive data in the source code and xml files.

```
bash
java -jar jd-gui.jar
```

D. Using MobSF for Static Analysis

```
bash
git clone https://github.com/MobSF/Mobile-Security-Framework-MobSF.git
cd Mobile-Security-Framework-MobSF
./setup.sh
```

[1]. Download and install MobSF:

[2]. Run MobSF and upload the APK file for analysis.

[3]. Review the MobSF report for vulnerabilities such as insecure data storage, improper permissions, and weak cryptographic algorithms.

[3]. Dynamic Analysis

A. Setting Up an Emulator or Physical Device

[1]. Use an Android emulator provided by Android Studio or a physical Android device.

[2]. Ensure the device is rooted to enable full dynamic analysis capabilities.

B. Using Drozer for Dynamic Analysis

[1]. Install Drozer on the testing machine and the target device.

[2]. Start the Drozer server and connect the device:

[3]. Use Drozer to enumerate and exploit vulnerabilities in the application's components:

```
bash
adb forward tcp:31415 tcp:31415
drozer console connect
```

```
bash
run app.package.list
run app.package.info -a <package_name>
run app.activity.info -a <package_name>
```

C. Using Frida for Instrumentation

[1]. - Install Frida on the testing machine and the target device:

[2]. Use Frida scripts to dynamically analyze and manipulate the application's behavior.

```
bash
pip install frida
frida -U -l <script.js> -f <package_name>
```

[4]. Network Analysis

A. Setting Up a Proxy Tool

[1]. Configure Burp Suite or Charles Proxy to intercept and analyze network traffic between the application and its backend servers.

B. Capturing Network Traffic

[1]. Set up the Android device or emulator to use the proxy tool.

[2]. Start the application and perform various actions to capture network traffic.

[3]. Analyze the captured traffic for sensitive information, unencrypted data, and potential vulnerabilities.

C. Using Wireshark for Deep Packet Inspection

[1]. Capture and inspect network packets using Wireshark:

[2]. Look for insecure communication channels, sensitive data leakage, and potential attack vectors.

```
bash
wireshark
```

[5]. Reverse Engineering

A. Using APKTool for Reverse Engineering

[1]. Decompile the APK file using APKTool:

```
bash
apktool d <app_name>.apk
```

B. Analyzing Decompiled Resources

[1]. Inspect the decompiled resources such as manifest files, layouts, and XML configurations for security misconfigurations.

C. Using JD-GUI for Code Analysis

[1]. Use JD-GUI to view and analyze the decompiled Java code:

[2]. Look for logic flaws, hardcoded credentials, and other security issues in the code.

```
bash
java -jar jd-gui.jar
```

Tools and Techniques for iOS Penetration Testing

iOS applications require different tools due to the closed nature of the platform:

A. Static Analysis Tools:

[1]. MobSF (Mobile Security Framework): Also supports iOS for static and dynamic analysis.

[2]. iRET (iOS Reverse Engineering Toolkit): A toolkit for reverse engineering iOS applications.

B. Dynamic Analysis Tools:

[1]. Frida: Works on iOS as well, allowing script injection into running processes.

[2]. Cycript: A JavaScript interpreter that enables dynamic analysis of iOS applications.

C. Network Analysis Tools:

[1]. Burp Suite: Similarly used for iOS to intercept and analyze network traffic.

[2]. Charles Proxy: A web debugging proxy application for monitoring network traffic.

D. Reverse Engineering Tools:

[1]. Class-dump: A tool to examine the Objective-C runtime of iOS applications.

[2]. Hopper Disassembler: A reverse engineering tool that allows the inspection of executable files.

E. Step-by-Step Guide for iOS Penetration Testing Using Tools and Techniques

[1]. Planning and Reconnaissance

A. Understanding the Application

[1]. Identify the purpose and functionality of the application.

[2]. Determine the target audience and gather information about the application's architecture.

B. Gathering Information

[1]. Collect information about the app's backend servers, APIs, and third-party services.

[2]. Use tools like Google Dorking to find sensitive information related to the application.

[2]. Static Analysis

A. Setting Up the Environment

[1]. Install Xcode for iOS development and analysis.

[2]. Obtain the IPA file of the application from the App Store or from a device.

B. Decrypting the IPA (if necessary)

[1]. If the IPA is encrypted, use tools like `Clutch` or `Frida-ios-dump` to decrypt it:

```
bash
git clone https://github.com/KJCracks/Clutch.git
cd Clutch
make
./Clutch -i <bundle_id>
./Clutch -d <bundle_id>
```

C. Decompiling the IPA

[1]. Use tools like `class-dump` to inspect the Objective-C headers:

```
bash
class-dump -H <decrypted_app>.app -o <output_directory>
```

D. Using MobSF for Static Analysis

[1]. Download and install MobSF:

[2]. Run MobSF and upload the IPA file for analysis.

[3]. Review the MobSF report for vulnerabilities such as insecure data storage, improper permissions, and weak cryptographic algorithms.

```
bash
```

```
git clone https://github.com/MobSF/Mobile-Security-Framework-MobSF.git
cd Mobile-Security-Framework-MobSF
./setup.sh
```

E. Using iRET for Additional Static Analysis

[1]. Download and set up iRET on a jailbroken device:

[2]. Use iRET to analyze the application's files and identify potential vulnerabilities.

```
bash
git clone https://github.com/S3Jensen/iOSRETToolKit.git
cd iOSRETToolKit
```

[3]. Dynamic Analysis

A. Setting Up a Jailbroken Device

[1]. Use a jailbroken iOS device to enable full dynamic analysis capabilities.

[2]. Install `Cydia` and necessary tools such as `OpenSSH` and `Filza`.

B. Using Frida for Instrumentation

[1]. Install Frida on the testing machine and the target device:

[2]. Use Frida scripts to dynamically analyze and manipulate the application's behavior.

```
bash
pip install frida
frida -U -l <script.js> -f <bundle_id>
```

C. Using Cycript for Dynamic Analysis

[1]. Install Cycript on the jailbroken device:

[2]. Attach Cycript to the running application and analyze its runtime behavior:

```
bash
wget http://apt.saurik.com/cycript/cycript_0.9.594_iphoneos-arm.deb
dpkg -i cycript_0.9.594_iphoneos-arm.deb
```

```
cycript -p <process_name>
```

[4]. Network Analysis

A. Setting Up a Proxy Tool

[1]. Configure Burp Suite or Charles Proxy to intercept and analyze network traffic between the application and its backend servers.

B. Capturing Network Traffic

[1]. Set up the iOS device to use the proxy tool.

[2]. Start the application and perform various actions to capture network traffic.

[3]. Analyze the captured traffic for sensitive information, unencrypted data, and potential vulnerabilities.

C. Using Wireshark for Deep Packet Inspection

[1]. Capture and inspect network packets using Wireshark:

[2]. Look for insecure communication channels, sensitive data leakage, and potential attack vectors.

```
bash
wireshark
```

[5]. Reverse Engineering

A. Using class-dump for Reverse Engineering

Extract the headers using class-dump:

```
bash
class-dump -H <decrypted_app>.app -o <output_directory>
```

B. Analyzing Decompiled Headers

Inspect the extracted headers for sensitive information and potential vulnerabilities.

C. Using Hopper Disassembler for Code Analysis

[1]. Download and install Hopper Disassembler.

[2]. Load the application's binary into Hopper for advanced code analysis.

[3]. Analyze the application's assembly code and identify potential vulnerabilities.

Table 1: Android and iOS Tools for Pen testing

Category	Android Tools and Techniques	iOS Tools and Techniques
Static Analysis	MobSF, AndroBugs	MobSF, iRET
Dynamic Analysis	Drozer, Frida	Frida, Cycript
Network Analysis	Burp Suite, Wireshark	Burp Suite, Charles Proxy
Reverse Engineering	APKTool, JD-GUI	Class-dump, Hopper Disassembler

Comparative Analysis

A. Static Analysis

[1]. Android: Tools like MobSF and AndroBugs provide comprehensive static analysis capabilities. Android's open-source nature allows deeper insights into application internals.

[2]. iOS: Static analysis on iOS is limited due to the platform's closed nature. Tools like MobSF and iRET offer some functionality, but with less depth compared to Android.

B. Dynamic Analysis

[1]. Android: Tools such as Drozer and Frida provide robust dynamic analysis capabilities, allowing testers to simulate attacks in real-time.

[2]. iOS: Dynamic analysis on iOS is more challenging but tools like Frida and Cycript offer substantial capabilities, although they require a jailbroken device for full functionality.

C. Network Analysis

Both platforms benefit equally from tools like Burp Suite and Wireshark, which are essential for intercepting and analyzing network traffic. Charles Proxy is also highly effective for iOS.

D. Reverse Engineering

[1]. Android: The availability of tools like APKTool and JD-GUI makes reverse engineering more straightforward on Android.

[2]. iOS: Tools such as Class-dump and Hopper Disassembler are effective but often require a jailbroken device to access the full range of features.

Challenges in Mobile Application Penetration Testing

A. Platform Differences:

The inherent differences between Android and iOS in terms of architecture and security models pose unique challenges for testers. Android's open-source nature allows deeper access and modification, whereas iOS's closed ecosystem restricts access, making comprehensive testing more difficult.

B. Encryption and Obfuscation:

Mobile applications often use encryption and obfuscation techniques to protect sensitive data, making it difficult to analyze and exploit vulnerabilities. This is especially challenging on iOS, where developers tend to use Apple's encryption and security frameworks extensively.

C. Environment Constraints:

Setting up a controlled testing environment can be challenging, particularly for iOS:

[1]. Jailbreaking Difficulties: Jailbreaking an iOS device is often necessary for in-depth penetration testing, but it is becoming increasingly difficult with each new iOS release. Apple continuously patches vulnerabilities used for jailbreaking, and newer devices come with enhanced security mechanisms.

[2]. Legal and Ethical Issues: Jailbreaking an iOS device can void warranties and may be considered a violation of Apple's terms of service. It also raises ethical questions and potential legal issues if done without proper authorization.

D. Challenges Specific to iOS:

[1]. Limited Tool Availability: There are fewer tools available for iOS penetration testing compared to Android. Many tools require a jailbroken device to function fully, limiting their effectiveness on non-jailbroken devices.

[2]. App Store Restrictions: Apple's App Store policies restrict the distribution and use of certain testing tools. Testers may need to use enterprise certificates to install testing applications, which adds complexity and potential security risks.

[3]. Code Signing and Execution: iOS requires all applications to be signed with a valid Apple Developer certificate. This adds an extra layer of complexity when modifying and testing applications, as any changes to the code will require re-signing and proper configuration.

[4]. Dynamic Analysis Challenges: Dynamic analysis on iOS is more challenging due to the platform's security mechanisms, such as sandboxing, System Integrity Protection (SIP), and Secure Enclave. These mechanisms make it difficult to execute arbitrary code and access certain parts of the system.

[5]. Firmware and Security Updates: Frequent firmware and security updates from Apple can disrupt the testing environment by patching vulnerabilities used for testing or breaking compatibility with testing tools. Keeping up with these updates is a constant challenge for penetration testers.

E. Challenges Specific to Android:

[1]. Device and OS Fragmentation: Android's open-source nature leads to a wide variety of devices and OS versions. This fragmentation makes it challenging to ensure comprehensive coverage during testing, as different devices and versions may behave differently and have unique vulnerabilities.

[2]. Custom ROMs and Manufacturer Modifications: Many Android devices run custom ROMs or have manufacturer-specific modifications. These variations can introduce additional vulnerabilities or alter the behavior of the operating system, complicating the testing process.

[3]. Permission Management: Android's permission model has evolved over time, leading to inconsistencies in how permissions are requested and managed across different versions. Testers need to account for these differences to effectively assess permission-related vulnerabilities.

[4]. Security Patch Levels: Not all Android devices receive regular security updates, leading to a wide range of patch levels in the wild. This variability can make it difficult to assess the risk associated with certain vulnerabilities, as some devices may be more susceptible than others.

[5]. Malware and Adware Proliferation: The relatively open nature of the Android ecosystem makes it easier for malicious apps to proliferate. Penetration testers must be vigilant in identifying and analyzing potentially malicious apps that could compromise the security of the target application.

[6]. App Store and Third-Party Distribution: Android apps can be distributed through various channels, including the Google Play Store, third-party app stores, and direct downloads. This diversity increases the risk of encountering malicious or compromised apps, which testers must consider when assessing the security of an application.

F. Cross-Platform Challenges:

[1]. User Data Privacy: Ensuring the privacy and security of user data is a critical concern on both platforms. Mobile applications often handle sensitive information, and any vulnerability that compromises user data can have severe consequences.

[2]. Third-Party Libraries and SDKs: Mobile applications frequently use third-party libraries and SDKs, which can introduce vulnerabilities. Testers must thoroughly assess these components to ensure they do not compromise the application's security.

[3]. API Security: Both Android and iOS applications often rely on backend APIs for functionality. Securing these APIs and ensuring they do not introduce vulnerabilities is a critical aspect of mobile application security.

These challenges highlight the complexities and constraints specific to both Android and iOS penetration testing, underscoring the need for specialized knowledge and tools to effectively assess the security of mobile applications.

Future Directions

[1]. **AI and Machine Learning Integration:** Future pentesting tools could leverage AI and machine learning to automate the detection of vulnerabilities and predict potential attack vectors.

[2]. **Enhanced Dynamic Analysis:** Developing more sophisticated dynamic analysis tools that do not require jailbreaking for iOS could significantly enhance pentesting capabilities.

[3]. **Cross-Platform Testing Frameworks:** Creating unified testing frameworks that can seamlessly work across both Android and iOS platforms would streamline the testing process.

[4]. **Blockchain for Security:** Utilizing blockchain technology to secure mobile applications could be a promising area for future research, potentially preventing unauthorized access and ensuring data integrity.

[5]. **IoT and Wearable Device Security:** As mobile applications increasingly integrate with IoT and wearable devices, future research should focus on securing these interconnected ecosystems.

Conclusion

Penetration testing is crucial for ensuring the security of mobile applications on both Android and iOS platforms. While Android offers more flexibility and a wider range of tools due to its open-source nature, iOS requires specialized tools and often a jailbroken device for effective testing. Despite the challenges, a combination of static, dynamic, network analysis, and reverse engineering techniques can provide a comprehensive security assessment. Continuous advancements in penetration testing tools and methodologies are essential to keep pace with the evolving threat landscape.

References

[1]. Arxan Technologies. (2018). State of Mobile App Security. https://www.arxan.com/resources/state-of-mobile-app-security-2018

[2]. OWASP. Mobile Security Testing Guide. https://owasp.org/www-project-mobile-security-testing-guide/

[3]. National Institute of Standards and Technology (NIST). (2019). Mobile Application Security. Retrieved from https://csrc.nist.gov/publications/detail/sp/800-163/rev-1/final

[4]. Symantec. (2020). Internet Security Threat Report. https://symantec-enterprise-blogs.security.com/blogs/threat-intelligence/istr-25

[5]. S. Raj and N. K. Walia, "A Study on Metasploit Framework: A Pen-Testing Tool," 2020 International Conference on Computational Performance Evaluation (ComPE), Shillong, India, 2020, pp. 296-302, doi: 10.1109/ComPE49325.2020.9200028.

[6]. S. Jadhav, T. Oh, Y. H. Kim and J. N. Kim, "Mobile device penetration testing framework and platform for the mobile device security course," 2015 17th International Conference on Advanced Communication Technology (ICACT), PyeongChang, Korea (South), 2015, pp. 675-680, doi: 10.1109/ICACT.2015.7224881.

[7]. Muccini, A. Di Francesco and P. Esposito, "Software testing of mobile applications: Challenges and future research directions," 2012 7th International Workshop on Automation of Software Test (AST), Zurich, Switzerland, 2012, pp. 29-35, doi: 10.1109/IWAST.2012.6228987.

[8]. Mainka, J. Somorovsky and J. Schwenk, "Penetration Testing Tool for Web Services Security," 2012 IEEE Eighth World Congress on Services, Honolulu, HI, USA, 2012, pp. 163-170, doi: 10.1109/SERVICES.2012.7.

[9]. M. Denis, C. Zena and T. Hayajneh, "Penetration testing: Concepts, attack methods, and defense strategies," 2016 IEEE Long Island Systems, Applications and Technology Conference (LISAT), Farmingdale, NY, USA, 2016, pp. 1-6, doi: 10.1109/LISAT.2016.7494156.

[10]. M. Z. A. Shebli and B. D. Beheshti, "A study on penetration testing process and tools," 2018 IEEE Long Island Systems, Applications and Technology Conference (LISAT), Farmingdale, NY, USA, 2018, pp. 1-7, doi: 10.1109/LISAT.2018.8378035.

[11]. Stahl, F., and J. Ströher. "OWASP and AppSec Security Testing Guidelines for Mobile Apps." Security_Testing_Guidelines_for_mobile_Apps_-_Florian_Stahl% 2BJohannes_Stroeher. pdf (2013).

[12]. He, S. Chan and M. Guizani, "Mobile application security: malware threats and defenses," in IEEE Wireless Communications, vol. 22, no. 1, pp. 138-144, February 2015, doi: 10.1109/MWC.2015.7054729.
[13]. M. Karami, M. Elsabagh, P. Najafiborazjani and A. Stavrou, "Behavioral Analysis of Android Applications Using Automated Instrumentation," 2013 IEEE Seventh International Conference on Software Security and Reliability Companion, Gaithersburg, MD, USA, 2013, pp. 182-187, doi: 10.1109/SERE-C.2013.35.
[14]. John Yeo, Using penetration testing to enhance your company's security, Computer Fraud & Security, Volume 2013, Issue 4, 2013, Pages 17-20, ISSN 1361-3723, https://doi.org/10.1016/S1361-3723(13)70039-3
[15]. M. S. Rahman, B. Kojusner, R. Kennedy, P. Pathak, L. Qi and B. Williams, "So {U} R CERER: Developer-Driven Security Testing Framework for Android Apps," 2021 36th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW), Melbourne, Australia, 2021, pp. 40-46, doi: 10.1109/ASEW52652.2021.00020.
[16]. Shahriar, Hossain, Md Arabin Talukder, and Md Saiful Islam. "An exploratory analysis of mobile security tools." (2019). doi: 10.1080/19393555.2020.1741743.
[17]. Shahriar, C. Zhang, M. A. Talukder, and S. Islam, "Mobile Application Security Using Static and Dynamic Analysis," Studies in Computational Intelligence. Springer International Publishing, pp. 443–459, Dec. 15, 2020. doi: 10.1007/978-3-030-57024-8_20.
[18]. M. Antonishyn, "Mobile applications vulnerabilities testing model," Collection "Information Technology and Security," vol. 8, no. 1. Kyiv Politechnic Institute, pp. 49–57, Jul. 09, 2020. doi: 10.20535/2411-1031.2020.8.1.218003.
[19]. Yankson, J. V. K, P. C. K. Hung, F. Iqbal and L. Ali, "Security Assessment for Zenbo Robot Using Drozer and mobSF Frameworks," 2021 11th IFIP International Conference on New Technologies, Mobility and Security (NTMS), Paris, France, 2021, pp. 1-7, doi: 10.1109/NTMS49979.2021.9432666.