Permission-based Android Malware Detection using Machine Learning

Saeed Seraj

A thesis submitted in partial fulfilment of the requirement of the University of Brighton for the degree of Doctor of Philosophy

November 2023

Abstract

Mobile devices, particularly Android-based devices, have become essential to our daily lives. However, this trend has also increased the number and sophistication of mobile malware, which can compromise user privacy, steal sensitive information, and cause other malicious activities. In this thesis, the focus is on detecting different types of Android malware using machine learning techniques. To achieve this goal, first, specialized datasets based on application permissions that are tailored to each type of malware was developed. Then optimized neural network architectures to detect each malware type was proposed. Specifically, this research focused on detecting malicious Antimalware and VPNs, Android Trojans, mobile Botnets, and malicious Adwares, some of the most prevalent and dangerous types of mobile malware. My approach has several advantages over traditional mobile security solutions. First, it provides a more fine-grained view of the behaviour of an application, enabling the detection of malicious apps that may appear benign based on their code alone. Second, it is scalable, enabling automated detection and classification of malware in the face of the rapidly growing number of Android devices and applications. Third, it is adaptable to new and emerging threats, making it more resilient to novel attacks. My results showed that my models achieved high accuracy rates in detecting these types of malware, outperforming existing methods. This work is the first to specifically target the detection of these types of Android malware based on permissions. This research's findings have important implications for the field of mobile security, as they provide a new way to defend against malware threats that are becoming increasingly sophisticated and prevalent. The developed models can be integrated into existing security solutions to provide more robust protection for users' devices and personal information. Overall, this thesis presents a novel approach to detecting targeted Android threats using machine learning techniques. By leveraging application permissions, specialised datasets and models for identifying various types of Android malware were developed. The results demonstrate that the developed models achieve high accuracy rates in detecting these types of malware and are effective at detecting targeted threats that may be missed by traditional signature-based approaches.

Acknowledgements

Dear Dr Nikolaos Polatidis and Dr Michalis Pavlidis,

I am writing to express my heartfelt gratitude for all the support and guidance you provided me during my journey to complete my doctoral dissertation. First and foremost, I thank God Almighty for granting me the strength and ability to complete this work.

You have been a constant source of inspiration and motivation throughout my time at the University of Brighton. Your unwavering commitment to my success, both professionally and personally, has been instrumental in helping me reach this milestone. I am deeply grateful for your endless patience, encouragement, and the valuable insights you provided, which have shaped my thinking and enriched my research.

I could not have asked for better mentors and advisors, and I consider myself fortunate to have had the opportunity to work under your guidance. Your dedication and passion for your work have inspired me to strive for excellence, and your unwavering support has helped me overcome any obstacles I encountered.

Lastly, I would like to express my appreciation to my family and friends for their support and encouragement throughout my academic journey. Without their love and encouragement, I would not have been able to achieve this milestone.

Once again, please accept my heartfelt thanks for your invaluable support and guidance. I am deeply grateful for everything you have done for me, and I look forward to staying in touch.

Sincerely,

Saeed Seraj

Declaration

I certify that this work has not been accepted in substance for any degree and is not concurrently being submitted for any degree other than that of Doctor of Philosophy (Ph.D.) being studied at the University of Brighton. I also declare that this work is the result of my own investigations except where otherwise identified by references and that I have not plagiarised the work of others.

Supervisors

Dr. Nikolaos Polatidis

Dr. Michalis Pavlidis

Contents

List of Publications

- ## Journal Papers

1. Seraj, S., Pavlidis, M., Trovati, M., & Polatidis, N. (2023). MadDroid: malicious adware detection in Android using deep learning. Journal of Cyber Security Technology, 1-28.

2. Seraj, S., Khodambashi, S., Pavlidis, M. and Polatidis, N., 2023. MVDroid: An Android malicious VPN detector using neural networks. *Neural Computing and Applications*, pp.1-11.

3. Seraj, S., Khodambashi, S., Pavlidis, M. and Polatidis, N., 2022. HamDroid: permission-based harmful android anti-malware detection using neural networks. *Neural Computing and Applications*, *34*(18), pp.15165-15174.

- ## Conference Papers

1. Seraj, S., Pimenidis, E., Pavlidis, M., Kapetanakis, S., Trovati, M. and Polatidis, N., 2023, June. BotDroid: Permission-based Android Botnet Detection Using Neural Networks. *Proceedings of the 24th International Conference on Engineering Applications of Neural Networks (EANN 2023) Leon, Spain 2023.*

2. Seraj, S., Pavlidis, M. and Polatidis, N., 2022, June. TrojanDroid: Android Malware Detection for Trojan Discovery Using Convolutional Neural Networks. *In Engineering Applications of Neural Networks: 23rd International Conference, EAAAI/EANN 2022, Chersonissos, Crete, Greece, June 17–20, 2022, Proceedings* (pp. 203-212). Cham: Springer International Publishing.

List of Figures

List of Tables

1. Introduction

In the past few years, smartphones have evolved from simple mobile phones into sophisticated computers. They are much more portable and consume less energy in comparison to personal computers. This fact extends their usage in business and home related activities such as surfing the Internet, Emails, SMS and MMS messages, online transactions and Internet banking, etc. All of these features make the smartphone a useful tool in our daily lives, but at the same time they render it more vulnerable to attacks by malicious applications (Wei *et al.,* 2012). Given that most users store sensitive information on their mobile phones, such as phone numbers, SMS messages, emails, pictures and videos, smart phones are a very appealing target for attackers and malware developers.

With almost 2.6 million applications on Android Google Play alone, Android has emerged as the leading mobile platform (AppBrain, 2023). Mobile application markets, such as Android Google Play, have fundamentally changed how consumers receive software, with daily updates and the inclusion of numerous applications. The rapid growth of the applications market, combined with the pervasive nature of applications provided on such platforms, has resulted in an increase in the sophistication and number of security threats targeting mobile platforms. According to recent studies, mobile markets have vulnerable or malicious applications, putting millions of devices at risk. For many years, malware has posed a threat to computer systems. It was only a matter of time before malware developers created malware for the Android platform, given the invention of Android systems and their significant market share. The market share of Android has increased significantly in recent years, attracting a slew of malware attacks that vary in complexity and scope (Demontis *et al.,* 2019).

The growing sophistication of mobile malware poses a severe threat to Android users worldwide. Existing signature-based security solutions are becoming less effective at detecting these advanced threats. Therefore, innovative techniques are needed to identify malicious Android apps with high accuracy. The intended beneficiaries of this technique are app store moderators, security companies performing vetting, and analysts evaluating new apps. By automating parts of the malware screening process, this technique can enhance efficiency and malware detection rates compared to purely manual review. Custom datasets improve detection over generic malware datasets by focusing on high-risk categories.

Since its initial release in 2008, the Android platform has seen tremendous growth, gaining a sizable market share over the years. The platform's popularity and widespread use are associated with increased interest from malware developers with a variety of malicious goals. With continued improvement and enhancement of security features, multiple aspects and vulnerabilities of the platform have been exploited. Various frameworks with varying capabilities and limitations have been developed over time. According to recent statistics, the popularity of the Android platform has resulted in a global market share of more than 86%. According to estimates, there are more than 1.2 billion monthly Android users. Android is the market leader, with 71.8% market share versus 27.6% for Apple iOS (Statista, 2023). Google's policies have enabled the Android platform to experience this level of growth and acceptance. Because of its open policy, millions of applications are now available on the platform, with a high level of tolerance for verification and release. Many factors are to blame for the rise of Android malware, some of which are not technical in nature, such as the lack of a regulatory body in open markets and the lack of consequences for those who provide applications with malicious potential or vulnerabilities. Because mobile applications are becoming more prevalent and complex, this scenario is likely to worsen.

Some of Google's policies are also dangerous to users' security. Due to open policies, third parties can operate an unofficial application store from which users can download applications without verifying security and authenticity. Because digital certificate usage on the Android platform is not strictly regulated, some application developers cannot be traced back to their original developers using digital signatures (Zhou & Jiang, 2012). Because digital certificates are not used, malware developers can easily release cracked versions of legitimate applications as well as Trojan horses disguised as legitimate applications. As a result, the Android ecosystem has become one of the most popular platforms for malware developers with a variety of malicious intentions. Despite the obvious malware threat, Google policymakers maintain that the company's open policy has done better than harm, benefiting millions of developers and security architects seeking to protect the platform. In the first half of 2019, approximately 1.9 million instances of Android malware were discovered, implying that an infected application was published every eight seconds. With more utilities being developed for the Android platform, it is almost certain that it will remain a target for malware (Gdata, 2019).

Attackers are primarily interested in gaining access to private information for individuals and entities, which is then used to orchestrate identity theft and hacking incidents. Private information from users' private profiles, such as online wallet access details, call logs, and contact information, can be obtained by attackers and used for malicious purposes. Because of the negative consequences of Android malware and the millions of potential victims, malware detection has been an ongoing security issue that has piqued the interest of a wide range of stakeholders. A variety of detection and security measures have been proposed and developed, with varying degrees of success and dependability. A common detection technique, for example, is the signature-based warning mechanism, which compares individual applications to known malware signatures (Burguera *et al.,* 2011). However, the method is limited in detecting mobile malware, which is constantly emerging, because a database may not contain their signature. This traditional technique has necessitated the development of more efficient malware detection methods such as static, dynamic, hybrid, and machine learning. Even with robust detection techniques, malware designers find ways to avoid detection, further complicating the process (Atkinson & Cavallaro, 2017).

Security can be thought of as a moving target. A significant portion of society has recognised the tedious form of computer protection as a nearly unavoidable effect in modern times. However, when compared to the world of personal computers, mobile represents a developing field. Mobile devices are now an important part of people's daily lives because they allow them to access a variety of ubiquitous services. The availability of such mobile and universal services has grown significantly as a result of various types of connectivity provided by mobile devices, such as Wi-Fi, Bluetooth, General Packet Radio Service, and Global System for Mobile Communications. Android also includes fully developed features for making use of cloud computing resources (la Polla *et al.,* 2013). Android security has emerged as an exciting research topic. Such research endeavours have examined Android security threats from various perspectives. They are dispersed across various research communities, resulting in a body of literature that spans multiple publication venues and domains. A substantial portion of the reviewed literature is published in the domains of software security and engineering. However, research on Android security runs concurrently with research on programming languages, mobile computing, and HCI, which considers topics such as the usability of security approaches.

Android is a Linux kernel-based operating system whose applications are written in Java and made available through built-in APIs. Its security framework includes app sandboxing, app signing, cryptographic APIs, and secure inter-process communication via the intents and permission model (Seo *et al.*, 2014). The permission model is a key security mechanism for preventing the unauthorised use of critical hardware and software resources (A. Felt *et al.*, 2011; Sarma *et al.*, 2012). To protect both the system and its users, Android requires apps to request permission before accessing certain system data and features. If the permissions are required to access the sensitive areas, the system grants the permission automatically, or it may ask the user to approve the request. Its effectiveness, however, is dependent on the user's response and other built-in features, most notably the intent. The intent is a messaging object that is used to ask another app component to perform a task. It allows components of the same or different applications to communicate with one another.

Because Android is the market leader, it is the primary target of Smartphone malware attacks (Do *et al.*, 2015). Android-based mobile devices have been constantly targeted due to their growing popularity and ease of developing, improving, re-packaging, and publishing apps (Wu *et al.*, 2013; Krutz *et al.*, 2015; Sufatrio *et al.*, 2015). Malware targeting the Android platform has skyrocketed in the last two years (Avdiienko *et al.*, 2015). The provision of installing third-party applications, as well as the increasing number of seemingly benign apps with malicious activities, aggravates the situation. The Android security framework has not been shown to be effective in preventing malware proliferation (Maiorca *et al.*, 2015). End-point security measures such as Antivirus software are unable to eliminate malware threats (Vidas & Christin, 2013; Maggi *et al.*, 2013; Penning *et al.*, 2014). This is due to the fact that the majority of solutions are signature-based and require regular updates to protect against an increasing number of malware variants, as well as a lack of obfuscation resilience (Feizollah *et al.*, 2015; Sheen *et al.*, 2015; Maier *et al.*, 2014; Faruki, Bharmal, *et al.*, 2015). To overcome the challenges of limited mobile device resources, outdated AV signatures, and malware code obfuscation techniques, innovative and resource-rich detection solutions are required.

The key innovation is the creation of custom datasets for each malware type extracted through manual static analysis of Android application package (APK) files. The datasets contain both benign apps and known malicious apps. Training machine learning models on these specialised datasets can enhance the detection of that particular malware type. These datasets are different from existing generic malware datasets because they focus on a single specific type of Android malware, allowing the machine learning models to specialise in identifying patterns unique to that malware type. The static analysis focuses on the app file itself, rather than dynamic runtime behaviour. This means any apps that become infected after installation would not be detected. The scope of this technique covers determining if a given Android app exhibits malicious behaviour or not before it is published. It does not consider collaborative attacks from multiple apps working together. The static analysis focuses on the app file itself, rather than dynamic runtime behaviour. This means any apps that become infected after installation would not be detected. This research aims to develop an efficient Android malware detection approach that can help app store moderators, security companies, and analysts quickly identify malicious apps before they are published and downloaded by users.

1.1 Android Malware

Malware (short for "malicious software") is a type of software that is annoying or harmful and is designed to secretly access a device without the user's knowledge (Enck *et al.*, 2011). Android has become the most popular smartphone operating system, making it a growing target for cybercriminals (Suarez-Tangil, Tapiador, Peris-Lopez & Ribagorda, 2014). Hackers are exploiting operating system and application vulnerabilities to gain access to systems, steal user data, and profit (Benats *et al.*, 2011; Fedler *et al.*, 2013; Liu & Liu, 2014). Android malware is rapidly evolving. As Android malware has grown exponentially over the years, McAfee Security Company's database now contains more than 100 million samples (Sanz, Santos, Ugarte-Pedrero, *et al.*, 2013). Because of new stealth techniques and encapsulation methods used by malware, detecting new malware apps has become quite difficult. Existing Android antivirus solutions are less effective at detecting and combating advanced malware (Li *et al.*, 2015).

As our mobile phones become more integrated into our personal and professional lives, they are more frequently targeted by cyber criminals than ever before. Because these devices contain valuable private information as well as access to financial services such as internet banking or e-commerce purchases, ensuring adequate security on the mobile phone is critical for all users. Android applications (apps) can be obtained from the official app store (Google Play Store) or from unofficial third-party stores such as GetJar or Slide ME. The Google Play Store was designed to be a store that meets all of the app requirements of the average user while also providing adequate security for app downloaders. To that end, Android phones include a security feature that prevents app installs from third-party stores, which users can disable. However, it is not recommended that it be disabled for most users.

The reason for such precautions is the possibility that adversaries may have injected malicious code into seemingly harmless Android apps (malcode). To compute and assess the level of threat that users face when downloading apps from third-party stores, we must compare the percentage of malware infections from Google Play to malware infections from third-party app stores. According to a Cheetah Mobile analysis (CheetahMobile, 2014)malware from third-party markets accounts for 99.86% of all malware infections, while Google Play accounts for only 0.14%.

Malicious Android apps pose a serious threat to users' privacy and security. However, detecting malicious apps is challenging given the sheer volume of apps submitted to app stores every day. Manual analysis of each app is time-consuming and not scalable. This research aims to develop an efficient Android malware detection approach that can help app store moderators, security companies, and analysts quickly identify malicious apps before they are published and downloaded by users.

The proposed technique focuses on detecting malicious antimalware apps, malicious VPNs, Trojans, Botnets, and malicious Adwares. These types of malware are commonly used to steal user data or gain unauthorized access to devices. By tailoring the approach to focus on these high-risk categories, the detection accuracy can be improved compared to general malware detection.

The key innovation is the creation of custom datasets for each malware type extracted through static analysis of Android application package (APK) files. The datasets contain both benign apps and known malicious apps. Training machine learning models on these specialized datasets can enhance the detection of that particular malware type. The manually extracted features to build these custom datasets set this work apart from existing generic malware datasets.

The scope of this technique covers determining if a given Android app exhibits malicious behaviour or not before it is published. It does not consider collaborative attacks from multiple apps working together. The static analysis focuses on the app file itself, rather than dynamic runtime behaviour. This means any apps that become infected after installation would not be detected. However, by detecting high-risk apps before publication, the approach can prevent a significant amount of malware.

The intended beneficiaries are app store such as Google Play moderators, security companies performing vetting, and analysts evaluating new apps. By automating parts of the malware screening process, this technique can enhance efficiency and malware detection rates compared to purely manual review. The custom datasets improve detection over generic malware datasets by focusing on specific malware types posing a high risk.

1.2 Evolution of Malware

Since the development of the first mobile worm, ‘Cabir’ designed to infect Nokia 60 series phones, mobile malware has come a long way. The infection appears to be innocuous because the worm displays the word 'Caribe' on the screen. The infection would spread via Bluetooth to other nearby Bluetooth-enabled devices such as printers, mobile phones, and so on (Apvrille, 2014). At the time, Symbian was a popular operating system that would provide a large market for mobile malware developers. In addition, Symbian's market share fell significantly in 2005, which could be attributed to Cabir's spread in Symbian mobile phones

(Tam *et al.,* 2017). ‘Cabir’ was succeeded in 2005 by CommWarrior, which spread through MMS as well as Bluetooth. The virus was created for the Symbian 60 platform and infected over 100,000 mobile devices by sending over 450,000 MMS. This spread added monetary value to malware development because each MMS sent would incur a carrier charge. The financial incentive was further exploited by RedBrowser, a Trojan discovered in 2006. The Trojan was designed to take advantage of premium SMS service, as each SMS would typically cost the device's owner $5. RedBrowser was a watershed moment in the evolution of mobile malware because it was the first malware to contaminate mobile phones running different operating systems by exploiting a flaw in the widely supported Java 2 Micro Edition (J2ME) (Apvrille, 2014). The following two years, 2007 and 2008, were relatively quiet in terms of the evolution of new mobile malware threats, and the development of non-commercial malware nearly died out (Maslennikov *et al.,* 2010). ‘Cabir’ is the first mobile worm, is an example of non-commercial malware because it was not created for monetary gain, as discussed at the beginning of the chapter. However, there has been a significant increase in the use of premium rate services by mobile malware (Apvrille, 2014). As cybercriminals began targeting online gamers to obtain their passwords, in-game assets, and virtual characters, the primary goal of malware development became data theft (Maslennikov *et al.,* 2010).

The mobile Botnet malware Yxes was discovered in 2009 and became one of the first malware for Symbian OS 9 (Apvrille, 2012). Because it was the first malware to send an SMS and access the Internet, this discovery was another watershed moment in the evolution of mobile malware as well as technological innovation (Apvrille, 2014). Since 2009, mobile malware has grown exponentially as a result of technological advancements that provide new avenues for profit, an increase in black market accessibility for selling and buying stolen information, and malware developers collaborating on exchanging malware code and system vulnerabilities (Tam *et al.,* 2017). (Apvrille, 2014) refer to 2010 as an industrial age for mobile malware, describing it as a transition from individuals to organised cybercriminals as monetary incentives increased significantly. The popularity of Android increased in the first quarter of the same year. Its global market share increased to 88% from 1.6% in the first quarter of 2009. Figure 1 depicts this popularity trend (Statista, 2023).

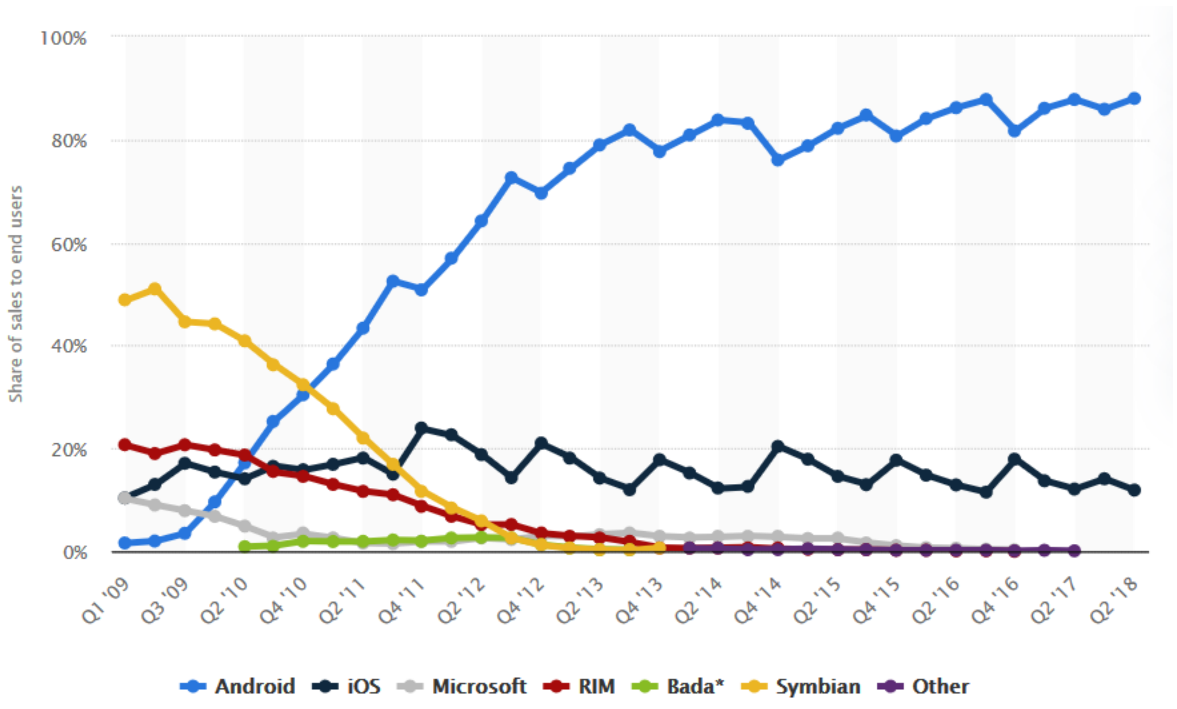


Figure 1. Global market share (Statista, 2023)

Because of the increased use of Android platforms, the platform has become vulnerable to new and sophisticated malware. The malware is designed to avoid detection while wreaking havoc on victims. Malware authors cause annual losses in the billions of dollars (Arshad *et al.*, 2016). As a result, as the number of Android system users grows, it becomes a profitable venture. (Amro, 2017) estimates that a new strain of malware is discovered every 10 seconds. This rate is not only alarming, but it also necessitates massive resources to identify and neutralise. In 2017, the percentage increase in evasive malware increased by more than 2000%, and this trend is expected to continue (Wei *et al.*, 2017). Massive attacks are being carried out, with thousands of users becoming victims and suffering losses. Malware authors have employed devious techniques to circumvent authentication requirements imposed as security measures. Because some malware attacks all authentication devices, authentication vulnerabilities are exposed (Azmoodeh *et al.*, 2018).

Alarms have been raised primarily as a result of an increase in ransomware cases in which malware authors hold Android systems hostage until they are paid. To hide their tracks, malware authors would primarily use untraceable cryptocurrencies (Azmoodeh *et al.*, 2018). Apart from the common malware target of locking device screens, ransomware has also targeted other malicious intentions such as data wiping, resetting security settings, GPS tracking, and personal information theft. Despite increased malware threats, antimalware companies and Google have been gradually investing resources in developing detection techniques. Critical databases have been developed over time and have played a significant role in improving Android security for millions of users; however, malware authors appear to be one step ahead all the time (Hicks & Dietrich, 2016).

1.3 Android Mobile Malware and Types

Mobile Malware is a malicious and unwanted piece of software that targets mobile phones and causes device damage as well as the loss or leakage of confidential data. The first mobile malware was discovered in 2004 and targeted the Symbian operating system. The first malware targeting Android was reported in 2010, and by 2011, Android had become the most popular OS for malware, with new malware families attacking every few weeks (Sufatrio *et al.*, 2015). Mobile malware targeting Android smartphones is widespread and rapidly expanding. This section provides a brief overview of the three most common types of malicious programmes that target mobile phones.

1.3.1 Trojan and Viruses

Trojans are pieces of malware that appear to be legitimate applications but contain harmful malicious code that, when executed, causes serious damage to the device (Enck, Gilbert, Chun, *et al.*, 2014). The most dangerous threat to Android devices is Trojanized apps, which can control the browser and steal account details, including bank login information. Trojans are viruses that can be installed in a variety of ways and cause damage ranging from simply annoying to highly destructive and irreversible. Mobile viruses can gain unauthorised access to sensitive files and memory by rooting the device.

1.3.2 Botnets

A bot is a type of malware that allows an attacker to take control of a mobile device that has been compromised. Bots are typically part of a network of infected mobile phones known as a "Botnet," which is made up of victim mobile phones that span the globe. They enable hackers to take control of a large number of mobile phones at once and turn them into "zombie" phones that work as part of a powerful "Botnet" to spread viruses, generate spam, and commit other types of online crime and fraud. Botnets infect devices by gaining access to their resources and data, allowing Botnet masters to control the device. They take advantage of system flaws and unpatched devices. They continue to spread by sending text messages or emails to the contacts of the infected device. Without the user's knowledge, hidden processes can secretly run executables or contact bot masters for new instructions. Future Botnets are expected to cause greater harm and to completely hijack and control infected devices.

1.3.3 Adware and Spyware

Spyware is malware that steals data from users and shares it with third parties for a variety of purposes, including future attacks. In some cases, these may be advertisers or marketing firms (Yang *et al.*, 2014), which is why spyware is sometimes referred to as "Adware". Adware refers to applications that use ad libraries. They collect user data in order to show relevant ads to users for marketing purposes. Ad libraries cause privacy leaks and can frustrate users by repeatedly displaying unwanted images or notifications on the screen (Suarez-Tangil, Tapiador, Peris-Lopez & Blasco, 2014). Spyware and Adware are typically installed without user consent by masquerading as legitimate apps or infecting legitimate apps with their payload.

1.4 Malware Propagation

Malware spreads via various sophisticated methods on mobile devices (Feng *et al.*, 2014). The following are some of the most common malware propagation methods:

- ***Infected websites:*** Cybercriminals create malicious websites that take advantage of system flaws to easily spread malware (Zhang *et al.*, 2013). When users access such websites from their mobile devices, they become infected.
- ***Third-party app markets:*** Third-party app stores have lax security controls over applications developed and uploaded by unknown parties (Rosen *et al.*, 2013). Malicious developers can upload Trojanized apps that users can download if the app has some appealing functionality. Third-party stores also distribute repackaged apps, which are popular apps that have been installed with malicious code, repackaged, and distributed.
- ***Spam emails and Botnets:*** Malware propagation via spam email is a simple and effective method. Attackers may send victims emails that appear to come from trusted sources such as the user's bank, Amazon, PayPal, or contacts. They may contain links to a malicious website, compelling them to change their password and then send the login information to a cybercriminal, or they may contain infected attachments that begin collecting data on their own the moment they are opened. Bots also spread malware by sending malicious links in text messages or e-mails to the contacts of infected users.

- ***Worms:*** Mobile worms are similar to viruses in that they can replicate and cause damage. Worms, unlike viruses, are standalone programmes that do not require an infected file or human intervention to spread. They spread to other devices via various exploits and system vulnerabilities.
- ***Onscreen Adware:*** Some appealing advertisements are displayed on the user's screen as a sidebar alongside a game or other app, and when the user clicks on them, he is directed to a malicious website.
- ***Dynamic payload:*** Hiding malicious code in the APK resources file and executing it with the Dex ClassLoader API after the main application has been installed.
- ***App updates:*** Malicious code is hidden in updates, and if the user installs them, the device will be infected.

1.5 Machine Learning and Data Mining

Machine learning is a subset of artificial intelligence in which an algorithm or method extracts patterns from data. The goal is to infer and generalise patterns from data automatically. Face recognition, handwriting digit recognition, spam filtering in email, and product recommendations from e-commerce sites are just a few examples of how machine learning can be used in our digital lives. To elaborate on one of the examples, the goal of handwriting digit recognition is to infer associations between drawn shapes and specific letters while accounting for variations of the same letter. Machine learning is a field that combines computer science, engineering, and statistics. Machine learning techniques can be useful in any field that requires data interpretation and action. Data mining is a field that is similar to machine learning in that we use many machine learning techniques. However, databases play a larger role in data mining. Data mining, also known as "knowledge discovery in databases," is an extension of exploratory data analysis with the same goal: to discover unknown and unexpected structures in data. The main difference is the size and dimensionality of the datasets involved. In general, data mining deals with much larger datasets for which highly interactive analysis is not possible (Wegman, 2003).

Consider the difference between learning from a fully labelled set of examples and learning from an unlabelled set of examples. When learning is performed from a fully labelled set of examples, as in this thesis, it is referred to as supervised learning. Unsupervised learning, on the other hand, is performed on a completely unlabelled set rather than a labelled set. The process of learning from labelled or unlabelled data sets is known as discovery or mining. Semi-supervised learning, which uses a partially set of examples for the learning activity, is a middle ground between supervised and unsupervised learning. In data mining applications, there are four distinct learning styles. The first is classification learning, which requires learning from a set of classified examples. The second is association learning, which seeks any association between features, the third is clustering, which seeks groups of examples that belong together, and the last is numeric prediction, which seeks to predict a numeric quantity rather than a discrete class (Witten *et al.,* 2005).

1.6 Data Mining for Information Security

Data mining is regarded as a promising solution to the ever-increasing issue of information security. Data mining-related solutions for information security are emerging as an alternative method of problem solving. Data mining or machine learning techniques such as classification, clustering, or association rule mining are used in a variety of information security applications. Induction algorithms are used in data exploration solutions to discover hidden patterns and build predictive models. Such techniques and algorithms have proven to be effective in addressing the majority of information security challenges. Attack pattern generalisation and discovery present a great opportunity for the data mining and information security communities to prevent and mitigate risks to information security. Classification, association rules, and clustering mechanisms can be implemented in the data before and after an information security compromise that maps the attack patterns of each individual attack. To deal with the most recent threats and risks, such as Distributed Denial of Service (DDoS) attacks, host-based intrusions, access control violations, and malicious code detection, powerful software solutions that incorporate the aforementioned techniques can be implemented.

Among the data mining use cases applied to information security issues are (Bhatnagar & Sharma, 2012).

• Detection of various anomalies and malware in the system by categorising benign and anomalous activities and categorising incoming data accordingly.

• Extraction of various security requirements, use of fuzzing techniques to identify vulnerabilities, definition and discovery of audit trails, and establishment of security policies.

• Detection of various cybercrimes, such as credit card fraud, money laundering fraud, and other financial crimes, as well as classification of criminals based on behaviour.

1.7 Data Mining for Cyber Security

Intrusion detection and malware detection are two areas of data mining for cybersecurity that have received a lot of attention. Despite the fact that both areas are relatively new in comparison to many classical theoretical computer science topics, there has been active research in these areas for 17 years, since the first research paper on data mining for malware detection was published in 2001 (Schultz *et al.,* 2001) .In the Joint Publication 1-02, Department of Defense (DoD) Dictionary of Military and Associated Terms, cyberspace is defined as "a global domain within the information environment consisting of the interdependent network of information technology infrastructures, including the Internet, telecommunication networks, computer systems, and embedded processors and controllers" (Department of Defense, 2015).

Sensors installed in cyber systems such as firewalls, intrusion detection systems (IDS), and Antivirus collect massive amounts of data. This data, whether network traffic or log data, is ripe for data mining to uncover valuable patterns and relationships for use in security research. Data mining may provide us with previously unimaginable capabilities. In addition to all of the tactical operations required to defend a cyber system, it has become critical to continuously sift through vast amounts of sensor data that could be made more efficient with advances in data mining techniques to accurately map the attack surface, collect and integrate data, extract knowledge, and produce useful visualisations (Blowers *et al.,* 2014). Strategic coordination of all data sources is becoming a critical component of effective cyber defence. This collection of data from various sources has the potential to become what we call big data. Dealing with large and rapidly growing data sources required us to develop new techniques, models, and a new type of computing infrastructure to process, analyse, and store data. Having a computing infrastructure that can ingest, validate, and analyse large volumes (size and/or rate) of data is one of many considerations when dealing with massive amounts of data. Another difficulty is evaluating mixed data (structured and unstructured) from various sources. It is frequently difficult to deal with unpredictable content that lacks obvious schema or structure, and it is frequently difficult to enable real-time or near-real-time collection, analysis, and results (Villars *et al.,* 2011).

1.8 Machine Learning and Data Mining for Malware Detection

The security industry is locked inside the endless loop of generating a specific signature to one kind of malware only for this specific kind of malware to be later modified to evade the present detection mechanisms (David & Netanyahu, 2015). This never-ending loop forces the security industry to attempt to defend against all attack vectors across the entire cyberattack spectrum, while attackers constantly improve their tools and attack methods. Instead of applying a single unique signature to each malware sample, data mining and machine learning propose a fundamentally different approach to detecting malware. Once malcode injection occurs, the traffic or app is no longer the same; there are clear indications of that mal code injection somewhere in the traffic or source code. Such changes that occur as a result of malcode injection provide us with the ability to learn about such changes in order to measure the degree of maliciousness of an executable in order to determine whether the file in question is benign or malicious. Data mining is a type of prediction in which we look for meaningful patterns in data and make classifications, form clusters, or predict numerical values based on these patterns. When there are no malicious code injections within the network traffic or source code of an app, it is in a clean state.

1.9 Defending Against Advanced Threats

Every day, cyber adversaries become more sophisticated and target organisations, corporations, and governments. I am dealing with advanced threats, which go beyond the attack sophistication threshold that we are accustomed to, and they include advanced malware and targeted attacks. Such adversaries' primary goal is to conduct industrial espionage, disrupt business and financial operations, and/or sabotage critical infrastructure. Many organisations today lack the workforce necessary to combat such threats. Traditional approaches to security, which use a rule, pattern, signature, or algorithm-based approach to detect malware or cyberattacks, are no longer effective against advanced threats. Traditional approaches to security have a major flaw in that they require constant updates and an influx of rules, signatures, or patterns to identify and mitigate each malware or threat.

Software and hardware solutions with data analytics at their core are quickly becoming the foundation of cyber and information security protection. Machine learning advancements are promising approaches to dealing with ever-changing and evolving advanced threats. Many machine learning techniques, algorithms, and tools are being used by security experts and researchers to combat some of the most advanced threats we face today, with many machine learning techniques, algorithms, and tools finding widespread applications and implementations in dealing with a wide range of security issues. When used correctly, machine learning can help us identify previously unknown vulnerabilities in software or hardware, detect complex cyber-attacks and malware, and mitigate insider threats through the detection of anomalous user behaviour.

1.9.1 Attacker Model and Malware Types Considered

This research aims to detect the following types of advanced malware threats targeting the Android platform:

- Malicious antimalware apps that pretend to provide security services but actually steal personal data or gain unauthorized access. These demonstrate sophisticated techniques like evading detection through obfuscation.
- Malicious VPN apps that claim to provide secure connections but monitor network traffic and steal sensitive information. These exhibit stealth behaviours by hiding their true intent.
- Android Trojans that masquerade as legitimate apps and are delivered through social engineering. They employ deception and transmit private data from devices.
- Botnets that take control of devices remotely and use them for malicious purposes like DDoS attacks. They utilize lateral movement and remote coordination.
- Malicious Adware that spies on user activity and exfiltrates data. These continuously extract information without the user's knowledge.

By focusing detection capabilities on these specific advanced threats, the approach aims to identify malicious behaviours such as deception, stealth, data exfiltration, and unauthorized access attempts. The datasets and machine learning models are tailored to these Android malware types which evade traditional signature-based defences.

1.10 Research Questions

1) What are the most effective feature extraction techniques and experimental methodologies for detecting and analysing Android malware? (Chapter 3 will address this question.)

2) How can Android permissions be leveraged to enhance the detection and prevention of malware, specifically focusing on antimalware, VPN, Trojans, Botnets, and malicious Adware? (Chapter 3 will address this question.)

3) What are the key performance metrics (e.g., detection accuracy, false positive rate) of existing antimalware solutions compared to proposed approaches, specifically for Trojan, Botnet, and malicious Adware detection? (Chapters 4, 5, 6, 7, and 8 will address this question.)

4) Does the proposed approach, specifically targeting antimalware, VPN, Trojans, Botnets, and malicious Adware, outperform existing comparative methods in terms of detection accuracy and efficiency? (Chapters 4, 5, 6, 7, and 8 will address this question.)

1.11 Aims

This research aims to gather information on existing claimed antimalware, VPN, Trojan, Botnet, and malicious Adware and examine their security issues. For this purpose, I collected Android antimalware, VPN, and APK file infected with Trojan, Botnet, and malicious Adware from Google Play and a few other websites and then classified them as benign or malware applications using “www.virustotal.com” which incorporates more than 70 reputed antimalware detection engines. To analyse the existing data including Android executable files, reverse engineering will be performed on these files by uploading each APK file in VirusTotal scanner and extract the required features. Then, the required features are extracted and converted to the desired format suitable for analysis. After that, I developed novel datasets with specialised datatypes which includes: Android antimalwares, VPNs and infected APKs with Trojans, Botnets, and malicious Adwares and indicated their required permissions in addition to their security issues as recognised by “www.virustotal.com”. Therefore, I proposed novel classifiers using deep learning algorithms such as multilayer perceptron neural networks (MLP) and convolutional neural networks (CNN) to detect and classify APK files as normal applications or any kind of malware. Also, my classifiers were tested and evaluated by evaluation metrics (e.g., Accuracy, Precision, Recall, F-1 Measure and AUC). The purpose is to analyse different batches of data and identify different subsets of malware in the Android operating system, which can help prevent intrusions into Android phones and prevent the hacking of the operating system and also classify detected malwares.

The overall aims of this research are:

- To develop an effective approach for detecting malicious Android apps, specifically malicious antimalware, malicious VPNs, Trojans, Botnets and malicious Adwares.
- To create custom datasets for training machine learning models tailored to these malware types.
- To evaluate the performance of different machine learning techniques on the custom datasets.
- To demonstrate that tailored detection outperforms generic malware detection.

1.12 Objectives

A lot of harmful exploiters have been engaging in profitable and abusive operations by getting information from mobile phones in a variety of ways, such as infecting antimalware and VPNs or utilising Trojans, Botnets, and Malicious Adwares to conduct their malicious activities on Android platforms. Therefore, this research:

1) Collect samples of the specific Android malware types to include in custom datasets.
2) Perform static analysis of Android APK files to extract features for the datasets.
3) Develop optimised machine learning models tailored to each malware dataset.
4) Evaluate detection performance using metrics like accuracy, precision and recall.
5) Compare performance to generic malware detection and other machine learning methods.
6) Analyse results to determine most effective techniques for each malware type.

This research is unique in focusing specifically on the detection of malicious antimalware apps and malicious VPNs for Android. Related works have focused on detecting generic Android malware, but do not specifically target fake security apps or VPNs. This is the first work to create custom datasets and machine learning models tailored to identifying these specific threats. Furthermore, these datasets are the first aiming at this specific area of android malware detection which means there are no similar works and datasets. Moreover, I proposed and tuned classifiers according to the characteristics of the novel datasets in order to achieve maximum efficiency of malware detection and classification and therefore results.

1.13 Contribution to Knowledge

The thesis makes the following contributions:

- A review of current approaches for analysis of Android malware, Android malware analysis, and custom-built malware detection technologies.

- A comparison was conducted among several well-known machine learning classifiers to understand which classifier performs best to detect malware.
- Developed five manually made novel datasets (antimalware, VPN, Trojan, Botnet, and Malicious Adware) focused on single type of Android malware.
- Developed optimised neural network algorithms in order to achieve optimum results in terms of well-known evaluation metrics and detection rate practically.

1.14 The Organisation of the Thesis

The rest of the thesis is organised as follows:

Chapter 1: Introduction

This chapter serves as the thesis's introduction. It gives a high-level overview of the research topic and its significance. The chapter begins by discussing Android malware and its prevalence in the digital landscape. It discusses malware evolution and focuses specifically on Android mobile malware and its various types, such as Trojans, viruses, Botnets, Adware, and Spyware.

Chapter 2: Literature Review

This chapter provides a thorough examination of the pertinent literature in the field of Android malware detection. It discusses the Android operating system's history, architecture, components, and security model. The chapter also goes over various methods and techniques for detecting Android malware, such as static and dynamic analysis approaches. Furthermore, it examines related datasets and identifies the shortcomings of current methods, leading to the development of a novel solution.

Chapter 3: Research Methodology

The research methodology used in the thesis is described in this chapter. It describes the machine learning and deep learning algorithms used in the study. The chapter also introduces and describes VirusTotal, a platform for collaborative IT security, and its role in the research. It also gives an overview of the methodology used to create a novel dataset, such as dataset creation, pre-processing, and classification and detection techniques. The research evaluation metrics are also discussed.

Chapter 4: Android Malicious Antimalware Detection

This chapter focuses on detecting malicious antimalware applications on Android. It starts with an introduction and background information on the subject. The chapter then presents the experimental evaluation that was performed for the detection of these applications, as well as the evaluation metrics that were used. The chapter concludes with the results of the experiments, comparisons with other classifiers, and general conclusions.

Chapter 5: Android Malicious VPN Detection

Chapter 5 delves into the detection of malicious VPN apps for Android. It serves as an introduction and background on the subject. The chapter describes the experimental evaluation used to detect these applications, as well as the evaluation metrics used. The experimental results, as well as comparisons with other classifiers and related works, are discussed. The overall conclusions are presented at the end of the chapter.

Chapter 6: Android Trojan Malware Detection

The detection of Android Trojan malware is the focus of this chapter. It provides background information and introduces the topic. The chapter describes the experimental evaluation for detecting these types of malware, as well as the evaluation metrics used. The results of the experiments are presented and compared to other classifiers and related works. The main findings and conclusions are presented at the end of the chapter.

Chapter 7: Android Botnet Malware Detection

The detection of Android Botnet malware is covered in this chapter. It begins with an introduction and background information on the subject. The chapter describes the experimental evaluation performed for Botnet malware detection, as well as the evaluation metrics used. The experimental results are discussed, along with comparisons to other classifiers and related works. The chapter concludes with the research's overall conclusions.

Chapter 8: Android Malicious Adware Detection

This chapter focuses on detecting malicious Adware on Android. It serves as an introduction and background on the subject. The chapter describes the experimental evaluation for detecting Adware, as well as the evaluation metrics used. The experimental results, as well as comparisons with other classifiers and related works, are discussed. The main findings and conclusions are presented at the end of the chapter.

Chapter 9: Discussion

The research results and findings are discussed in depth in this chapter. The research's implications, limitations, and future directions are discussed. The chapter provides a thorough examination and interpretation of the research findings.

Chapter 10: Conclusion and Future Work

The final chapter summarises the thesis's main contributions and findings. It restates the objectives of the study and addresses the research questions. The chapter concludes with suggestions for future work and areas for further investigation.

2. Literature Review

2.1 Background

Android is a Linux-based open-source operating system. Android is made up of six major components: the Linux kernel, the Hardware Abstraction Layer (HAL), Native C/C++ libraries, the Android runtime, the Application framework, and the System applications. For drivers and core services such as security and memory management, Android relies on the Linux kernel, whereas HAL provides standard interfaces that expose device hardware capabilities to the higher-level Java Application Programming Interface (API) framework. The Android runtime, on the other hand, includes a set of core libraries that implement the majority of the Java programming language. The application framework includes APIs for controlling how the application looks and behaves. Finally, system applications include a set of essential applications such as Contacts, phones, calendars, and a browser. A typical Android application is divided into three sections: layouts, activities, and extra resources. Layouts define how the application should look and are typically implemented in XML, whereas activities define what the application should do and are typically implemented in Java. Extra resources, such as images, sound files, and data, are required.

- The Java source code of Android applications is compiled into.class files using the Java compiler, then converted into Android format (DEX) files, which are combined with the other application files in a package known as the Android application package (APK). Android then executes the compiled code using the Dalvik Virtual Machine (DVM) or the Android Runtime (ART). Figure 2 depicts the progress of the Android application execution. Every Android application includes a file called AndroidManifest.xml that contains application-specific information such as:
- Permissions requested to access specific components, features, or other applications (for example, *ACCESS_WIFI_STATE, ANSWE_PHONE_CALLS, BATTERY_STATS, DELETE_CACHE_FILES, and CAMERA).* These permissions have seven levels of protection: normal, dangerous, signature, signature-OrSystem, deprecated, not for use by third-party applications, and others.
- The application makes use of features (single hardware or software). For example, if the application wishes to use the device's camera and Bluetooth, the android.hardware.Bluetooth and android.hardware.camera elements will be declared in the AndroidManifest.XML file.

- Broadcast receivers monitor for system or application events, allowing the application to respond to these registered events. For example, if the application wants to display a message if the battery is low, it can use a broadcast receiver for Intent.*The_ACTION _BATTERY_LOW* event should be declared in the AndroidManifest.XML file. (Android Developers, 2020)

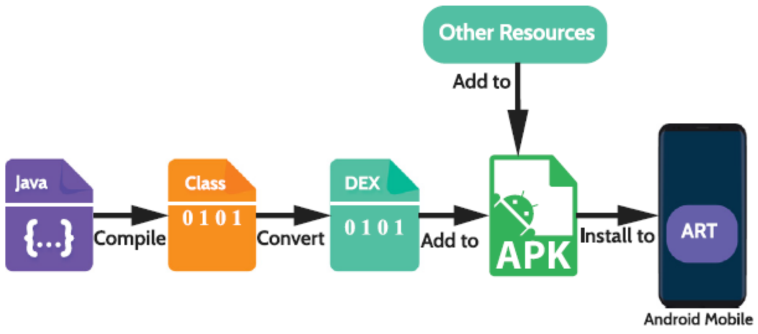


Figure 2. Android application execution progress

APK files are simply zip files that contain compiled source code in the form of DEX files, XML files, and additional resources. APK file disassembly into a near-original form of the source code for Android applications necessitates the use of specialised tools. Table 1 lists four of these specialised tools as well as their disadvantages.

Tool Name	Advantages	Disadvantages
Android Asset Packaging Tool (aapt)	Viewing AndroidManifest.xml content by using the command line.	Specific for AndroidMan-ifest.xml file only.
Smali/Baksmali Tool	Assembling and disassembling for DEX files format using a useful Graphical User Interface (GUI).	It needs multiple steps to be-called from command line.
APK Tool	– Decoding resources to nearly original form (as smali files A language that provides readable and understandable format for the DEX files. – Rebuild resources after making some modifications. – Enabling the calling from the command line with parameters.	Without GUI
Dex2jar with JD-GUI	– Dex2jar to convert the DEX files into a compressed jar file containing the .class files. – JD-GUI decompiles the jar file content to the original source. – Effective GUI	– It cannot be called from the command line and from a cus-tomized tool. – Enabling viewing only fea-ture.

Table 1. List of tools to decompile APK file

2.2 Android Operating System (OS)

This section provides a brief overview of the Android operating system and its security features. This provides background on the Android platform which is the focus of this research. Understanding the architecture and security features of Android systems is crucial for developing effective malware detection techniques. Google's Android is an open-source mobile operating system that it developed and maintains. Android is built on a Linux kernel, and its source code is available under an Apache licence. Google bought Android Inc., the company that created the Android operating system, in August 2005 as a strategic move to enter the mobile market (Robinson & Weir, 2015). Google launched Android in 2007 (Wikipedia, 2023a), paving the way for HTC to launch the first commercially produced Android device, the 'HTC Dream,' in September 2008 (Wikipedia, 2023b). With over two billion activated Android-powered devices and over two billion monthly active Android users, Android has since become a ubiquitous operating system (Lee, 2017).

2.3 Android Platform Components

The Android operating system is made up of several components, which I will go over in detail in the following subsections. The components discussed, like permissions and application packaging, directly relate to the static analysis techniques used in this work for extracting features from Android apps

2.3.1 Android Architecture

Android is an open source, Linux-based software stack created for a wide array of devices and form factors. Android is a platform designed for mobile devices; its architecture is divided into six layers, as shown in Figure 3 In the following subsections, I will examine all six layers briefly.

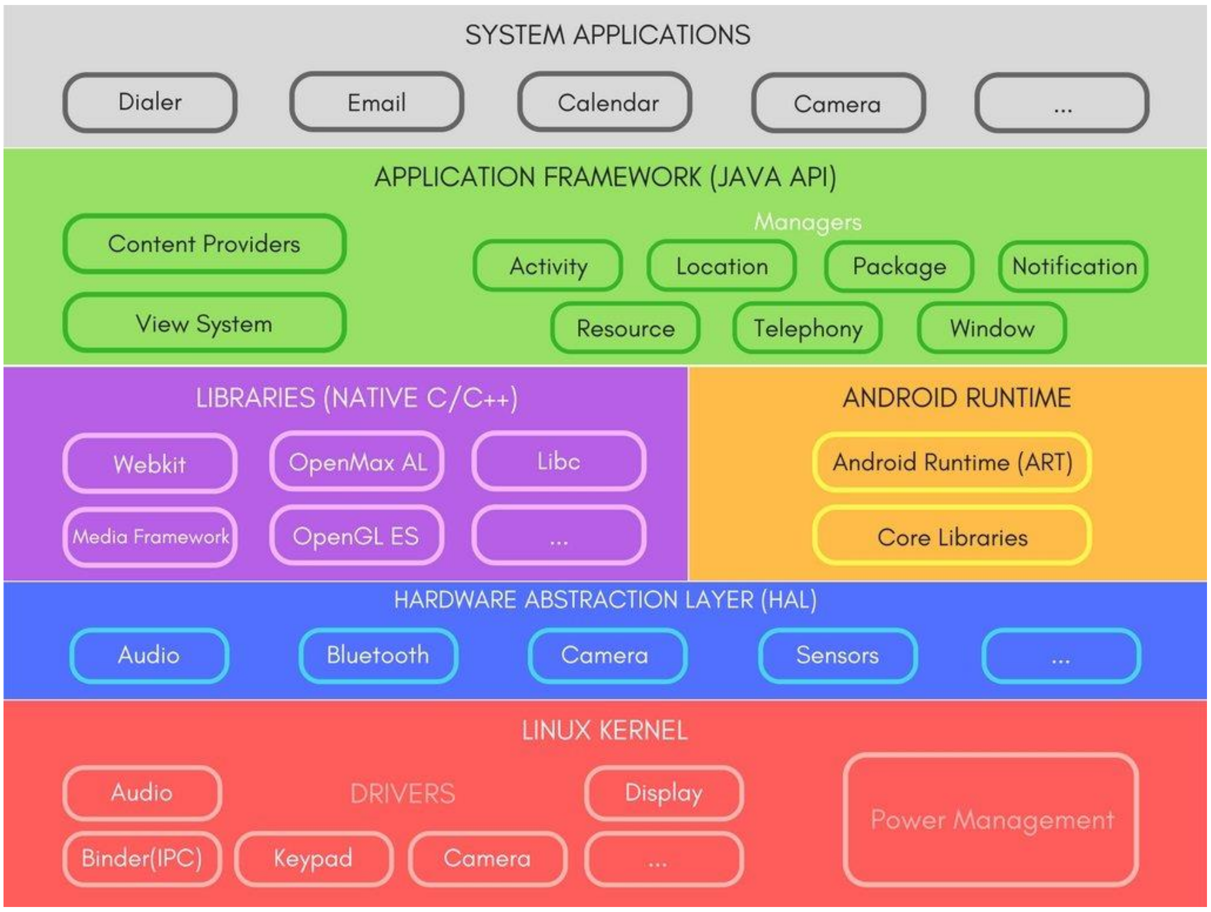


Figure 3. Android software stack

2.3.2 Android Applications

System applications, developed by the Android team, and all third-party applications are installed in the topmost layer of the Android software stack. Android includes a set of core applications that allow users to perform basic tasks such as SMS messaging, calendars, contacts, making or answering phone calls, and more. If a user prefers to use a different application for a specific purpose rather than the default system one, system applications have no precedence over third-party applications. Android allows you to use individual applications to perform a basic function provided by a system application, for example, any third-party application can become the default application for sending and receiving SMS messages.

2.3.3 Java API Framework

The Java programming language is primarily used to create Android applications using a variety of application programming interfaces (API) provided by the Android software development kit (SDK). This layer is made up of modular and reusable core components and services that are used to build Android apps. These reusable components include the View System, which is responsible for building the application's user interface using UI components such as list boxes, grids, and buttons, the Resource Manager, which is responsible for providing access to static non-code assets such as graphics, localised date-time, or string variables, the Notification Manager, which is responsible for allowing applications to present alerts to users, the Activity Manager, which is responsible for controlling the life cycle of applications and providing navigation, and the Content Provider.

Android applications are created by utilising some basic tools that manage the device's primary functions, such as call reception, text messaging, and battery usage monitoring, among other things. The following are some of the key components of the Application framework:

- ***Activity manager:*** The Activity Manager monitors all active applications on the device and disables background processes when memory is low. It also identifies applications that go more than five seconds without responding to an input event.
- ***Content providers:*** Content Providers are in charge of data sharing between applications (Enck *et al.,* 2011). Photos and contact lists, for example, can be accessed by multiple applications and are thus stored in the content provider.
- ***Telephony manager:*** Telephony manager manages phone calls and provides access to parameters such as set's (IMEI).
- ***Location manager:*** The location Manager is in charge of providing location services that are used by various applications to determine geographical location through embedded GPS or cell tower communication.
- ***Resource manager:*** The Resource Manager manages the resources that are used by various applications.

2.3.4 Native C/C++ Libraries

Many core Android features, such as ART and HAL, are supported by the native C/C++ layer. These fundamental features are written in native code and require native C/C++ libraries to function properly. This access is provided by the Android platform via Java framework APIs. For example, the Java OpenGL API can be used to access OpenGL ES in order to add drawing and graphics manipulation support to a programme. Third-party app developers who need C/C++ code can use the Android Native Development Kit (NDK) to access native libraries.

2.3.5 Android Runtime

The managed runtime environment used by some Android system services and third-party applications is known as Android Runtime (ART). The Dalvik virtual machine (Dalvik VM) was the runtime environment for Android prior to Android version 5.0, API level 21. Because each application runs in its own process, which in turn has its own instance of the Android Runtime, applications running on Android version 5.0, API level 21 or later use ART for faster startup and ongoing execution. At installation time, ART pre-compiles the bytecode into native code using a method known as Ahead of time (AOT). This eliminates the need to execute applications in interpreted code, resulting in faster execution. ART also has a better garbage collector (GC) and better debugging support.

2.3.6 Hardware Abstraction Layer

The hardware abstraction layer (HAL) allows application developers to access the hardware capabilities of the device. Applications can make use of the Java API framework to access multiple library modules that implement an interface for various hardware components exposed by HAL. This layer also allows developers to create their own drivers.

2.3.7 Linux Kernel

The Linux Kernel is the foundation of the Android platform architecture and is the lowest level layer in the Android software stack. Memory management, multi-threading, network stack, process isolation, storage management, security management, and other low-level operating system tasks are handled by the Kernel.

2.4 Dalvik Virtual Machine

The virtual machine is integral to how Android apps are executed, which influences the malware detection approach. A runtime is a set of software instructions that are executed when a programme is launched. These instructions are in charge of translating the application code into machine code that the device can execute. To run the Android Package (APK) files that comprise an Android application, Android uses a virtual machine as its runtime environment. Since Android's inception in 2007, Dalvik has served as the default virtual machine for running applications on top of device hardware. To interpret bytecode, the Dalvik runtime employs the Just-in-Time (JIT) compilation method, which was first introduced in Android 2.2 Froyo. JIT means that applications are partially compiled and built, which means that each time an application is launched, it must first be compiled. The downside of this approach, which was introduced as an improvement over the previous conventional interpreter approach that compiled and ran code line by line, is the massive overhead when launching applications. The following are two significant advantages of this approach: For starters, because the code is isolated from the core, any intentional or unintentional security threat is contained within the virtual machine and does not affect the primary OS. Second, the code can be compiled on another platform and then run on the mobile platform via the virtual machine.

2.5 Components of an Android Application

These components like Activities and Intents are leveraged by malware and also provide signals for detecting malicious apps based on capability. Components are the fundamental building blocks of an Android application. Each Android application is made up of four standard components that manage various aspects of the application's functionality. The following are the four types of application components:

2.5.1 Activities

Activities offer a single screen as well as an interface. Each application may contain activities that perform various tasks, such as reading e-mail in an e-mail application or displaying available routes in a navigation application. Each activity works independently to create a unified user experience for a specific app. A different app can initiate activities belonging to another app (as long as permission is granted); for example, a camera app may initiate an e-mail activity in the e-mail app.

2.5.2 Services

Background functionality is provided by services for long-term operations that do not require an interface. Services are similar to activities; however, the only major difference is that no interface is required for each activity. A service could be music playing in the background while the user is using another app or downloading data over the network without interfering with user interaction through the use of an activity. Services can be launched by other app components such as an activity or a broadcast receiver.

2.5.3 Content Providers

Content providers manage shared app data. The data may be stored in the file system, a SQLite database, or any other persistent storage location accessible to the app. Other apps can query or even modify the data as long as the content provider allows it. The Android operating system includes a content provider that manages the user's contacts information, and an app with the necessary permissions can send queries to the content provider to read and write information about a specific contact.

2.5.4 Broadcast Receivers

A broadcast receiver listens for specific system-wide broadcast announcements in order to determine whether or not they are the intended recipient. While many broadcasts are initiated by the system, such as a low battery, apps can also initiate broadcasts, for example, to notify other apps that data has been downloaded to the device and is ready for use. Broadcast receivers typically serve as a gateway to other components and lack a user interface. They are tasked with launching a background service to carry out a task in response to a specific event. Non-ordered and ordered broadcasts are the two types of broadcasts. Non-ordered broadcasts are sent to all interested receivers at the same time; ordered broadcasts, on the other hand, are sent to the receiver with the highest priority first, before being forwarded to the receiver with the second highest priority. The battery-low announcement is an example of a nonordered broadcast, whereas an incoming SMS text message announcement is an example of an ordered broadcast. When a receiver receives an ordered broadcast, he or she may choose to abort it so that it is not forwarded to other receivers. This enables vendors to create alternatives to the official Android apps, such as a text message manager that can disable the official Android messaging app by using a higher priority receiver and aborting the broadcast after handling the incoming message.

2.5.5 Intents

The message routing system based on Uniform Resource Indicators is used by Android to establish communication among application components (URIs). In order to activate components such as activities, services, and broadcast receivers, asynchronous messages known as "intents" are sent. Intents are nearly equivalent to parameters passed to API calls, with the following key differences in how API calls and intents invoke components:

- API calls are synchronous while intent-based invocations are asynchronous.
- API calls are compile-time bindings while intent-based calls are run-time bindings.

In order to listen for intent, intent filters that specify the types of intent that an activity, service, or broadcast receiver can respond to must be implemented. An intent filter declares a component's capabilities. It defines the tasks that an activity or service can perform and the types of broadcasts that a receiver can handle. It allows Intents of the declared type to be received by the corresponding component. In most cases, intent filters are defined in the AndroidManifest.xml file. The category, action, and data filters of an Intent Filter define it. It may also include additional metadata. When an intent is broadcast and received by the appropriate listener, the Android platform invokes the intent filter to complete the task. This means that both components are unaware of each other's existence and can still collaborate to provide the desired outcome for the end user. When an intent is broadcast and received by the appropriate listener, the Android platform invokes the intent filter to complete the task. This means that both components are unaware of each other's existence and can still collaborate to provide the desired outcome for the end user. Some intents may need to be sent with specific permissions, whereas system intents can be sent by processes with the system's UID. The latter, regardless of permissions, cannot be sent by an application and can only be sent by system processes.

2.5.6 The Manifest File

An "AndroidManifest.xml" file is required for every Android application. It is some kind of configuration file that contains references to the implemented components. This file describes each component of the application and how they interact with one another. All application components must be declared in this file, which is located at the root of the app project directory. Activities and services that are not declared in the manifest are not permitted to be carried out. Broadcast receivers, on the other hand, can be declared in the manifest or dynamically registered using the registerReceiver() method. Additionally, the manifest specifies application requirements such as special hardware requirements (e.g., camera, temperature sensor) or the minimum API version required to run the app. An application must be granted the appropriate permission to access the protected components (e.g., external storage, accessing the contact list). The app's permissions must be defined in the app's AndroidManifest.xml file. This way, the Android OS can prompt the user during runtime to grant the specific required permission(s) to enable the app to access these components via specific APIs. The protected components have a unique Linux group ID within the OS, and granting the corresponding permission makes the app's VM a member of the corresponding unique group, allowing access to the restricted components.

2.6 Permissions

The permission model is a key part of Android security and serves as the primary feature used in this research's machine learning techniques. For a long time, Android relied on presenting users with a list of all the permissions requested by the app for acceptance prior to app installation. This method of asking a user to accept a list of all the permissions the app requires was revamped with Android's version 6.0 (API level 23) update, into asking for permission acceptance when the app required it during runtime. The AndroidManifest.xml file declares all of the permissions that an application requires. Each Android application contains several strings for permission usage, such as "android.permission.CAMERA" or "android.permission.READ_LOGS". As the examples show, a permission name typically includes the package name as a prefix. The Manifest file's default format is binary XML, which means it must be parsed programmatically to extract the permission strings. Several factors make permission names suitable for use in machine learning for malware detection. First, an app must be granted specific permissions in order to use specific phone features via API calls. For example, an app that does not have CAMERA permission will be unable to use the phone's camera to take photos or record videos. Second, certain permission combinations may be associated with malicious behaviour. For example, malware that uses SMS to spread to the list of contacts in your phone's address book while also carrying out cyber espionage operations must have a specific combination of permissions including READ_CONTACTS, SEND_SMS, INTERNET, CAMERA, and RECORD_AUDIO.

2.6.1 Application Permission Structure

Permissions are enforced in the Android OS through permission validation mechanisms that must be invoked by some key components. The system process is specifically tasked with implementing the permission validation mechanism via several invocations scattered throughout the API. The key components that comprise the Android permission enforcement model will be discussed further below. Third-party applications on the Android platform have access to phone hardware, settings, and user data via an extensive API. The permissions security feature governs access to security or privacy-sensitive parts of the API. Before installing each application, the user is presented with a list of permissions that must be granted in order for the application to function properly. Each application developer must determine the required list of permissions ahead of time, and if a user notices anything suspicious, they can cancel the installation entirely. This allows a user to assess an application as potentially dangerous or benign on some level. The fact that most people are negligent about reading such information, on the other hand, is the Android OS's weakest point. There are 134 officially defined application permissions in total (Felt *et al.,* 2012b), divided into four protection levels, with each level enforcing a different security policy. There are three levels of risk, ranging from low to high:

2.6.1.1 Normal Permissions

Included are permissions that pose the least risk to the user due to the use of API calls that cannot be used to harm the user. They allow only isolated application-level features to be accessed, posing little risk to other applications, the system, or the user. These types of permissions are granted automatically without the user's explicit consent.

2.6.1.2 Dangerous Permissions

Permissions in this category allow access to API calls that could be malicious and allow access to private user data. They provide a broader range of access to device resources and give requesting applications control over the device, which can have a negative impact on the user. Applications that require these types of permissions will require the user's explicit approval before being installed.

2.6.1.3 Signature Permissions

A permission granted by the system only if the requesting application is signed with the same certificate as the application that declared the permission. If the certificates match, the system grants the permission without requiring the user's explicit approval.

2.6.1.4 SignatureOrSystem Permissions

This permission type is only granted to applications that are either part of the system image or are signed with the same certificate as the application that declared the permission. This permission type should be avoided in most cases because the signature permission should provide adequate protection regardless of where an application is stored. This permission is reserved for special cases, such as when multiple vendors want to embed applications inside the system image and need to explicitly share specific application features. This Android security model is static because an application only needs to obtain a permission once, and the list of permissions the application has cannot be changed during the application's lifetime on the device.

2.7 Android Application Programming Interface (API)

The API calls apps make can indicate malicious behaviour and are commonly used features in Android malware detection. The Android public API consists of 8,648 distinct methods, some of which are protected by permissions (Chin *et al.,* 2011). However, there are no centralised policies in place to perform permission checks when an API is called. The Android API framework is divided into two parts: a library that lives in each application's virtual machine and an API implementation that runs as a system process. The library includes the tools needed to interact with the API implementation. The API implementation in the system process is not constrained by the restrictions imposed by permissions systems, whereas the API library is constrained by the set of permissions accepted during application installation. In the Android operating system, API calls are handled in three steps: First, the application invokes the library's public API, then the library invokes a private interface (an RPC stub), and finally, the RPC stub initiates an RPC request, instructing the system process to instruct a system service to perform the desired operation. Each application's permission checks are stored in the API implementation in the system process. To determine whether the invoking application has the necessary permissions, the permission validation mechanism is invoked.

2.8 Application Configuration

The manifest file structure provides key insights into requested permissions and app capabilities. The manifest refers to a required configuration file (AndroidManifest.xml) that is included with all Android applications. It specifies many things, including the main components of the application, including their abilities and types, as well as enforced and required permissions. The values of the manifest file are attached to the Android application during compilation, and they cannot be changed during run-time. Aside from sandboxing, permission enforcement is another mechanism provided by the Android framework for application protection. Indeed, permissions are a strength of the Android security model. The application manifest permissions define secure access to sensitive resources and cross-app interactions. When users install applications, the Android system requests permissions from the user before installation. If the user refuses to provide the application with the necessary consent, the application's installation is cancelled. Furthermore, any Android application may describe its permissions for self-protection using the built-in permissions that the Android system provides to protect various system resources.

Because Android's access control model is at the application level, there is no mechanism for determining the security state of seemingly benign applications colluding with one another. This type of access control model introduces numerous security challenges, such as application collisions (Bugiel *et al.*, 2012) and re-delegation attacks (A. P. Felt, Finifter, *et al.*, 2011), which have been discovered to exist in market applications (A. P. Felt, Wang, *et al.*, 2011; Davi *et al.*, 2011). Application collusion attacks occur when malware developers distribute malicious code across two or more applications and launch an attack when a user installs all of the malicious code-containing applications. The Android intra-application communication feature is used to communicate among the affected applications. A re-delegation attack is a type of application collusion in which a lower-privilege application collaborates with a higher-privilege application to perform unauthorised operations (A. P. Felt, Finifter, *et al.*, 2011).

2.9 Android Security Model

Understanding the Android security mechanisms provides context for where malware can bypass or exploit weaknesses. The Android security model is intended to address the platform's unique security requirements. The Android security model is divided into five domains, each of which defines different security rules for the platform. The first domain necessitates multiparty agreement. The Android platform can be divided into three categories in this regard: user, developer, and platform. Users have control over the data in shared storage, whereas developers have control over the information in application folders. The platform manages data in locations that are only accessible to the operating system. Each party must agree to provide the data whenever it is requested by a specific programme and has the right to revoke the privilege at any time. The rule ensures that all stakeholders are aware of the use of their data elsewhere. The Android platform, which is an open ecosystem platform, is the second domain. It gives developers control over the data they are willing to share with other programmes, thereby increasing security. Furthermore, mobile devices must be security compatible. To be compatible with the Android operating system, mobile phones must pass Google's Compatibility Test Suite. Manufacturers must follow a number of recommendations to ensure the security of devices running the Android operating system (Mayrhofer *et al.*, 2021).

The third domain consists of application programmes that act as security controllers. Running a programme with administrative user privileges on the traditional desktop ensures that it has complete access to the system's resources. The same is not true for the Android platform, and applications are not considered capable of fully authorising user actions. Such a design creates a sandpit environment in which applications can run without affecting other applications or system settings. A multiparty consent is used by an application that requires data from other areas by requiring permission from the data owner, such as the platform, application programme, or the user. The fourth domain is the recommended platform for downloading applications, Google Play. Google's platform is a service that ensures the company has control over the programmes installed on user devices. The control's primary goal is to ensure user data security by ensuring that mobile applications meet various security-related requirements. Furthermore, because Google Play is installed on Android users' devices, it acts as an Antivirus by scanning applications for malicious code during downloads just before they are installed on the device. Furthermore, Google Play scans an Android phone on a regular basis to ensure that application updates are not malware. When malware is detected during the download of an application, the user is usually warned about the problem. The application will also remain available on the Google Play store until it is removed following an extensive review of the issues detected by Play Protect (Hutchinson *et al.,* 2019). The final dimension is the fail-safe, which returns a device to its factory settings, which are typically safe. When a mobile phone is infected with persistent malware, the user can restore it to a safe state by formatting the writable parts and returning it to a state that uses only verified system code. As a result, the Android security model has several distinct dimensions that contribute to the overall security of a device running the operating system (Mayrhofer *et al.,* 2021). Following subparagraphs discuss some of the key security features of the Android security framework.

2.9.1 System and Kernel Level Security

Android provides traditional Linux kernel security guarantees with the addition of secure IPC for application isolation.

2.9.2 The Application Sandbox

Android employs Linux user-based protection for application identification and isolation. Android creates a kernel-level sandpit for each application, which has its own user id and runs in its own process. Permissions govern how applications interact with system resources and other applications. As it is located at the kernel level, application sandpit is equally effective in exercising the same controls on system applications and native code. The application sandpit contains the operating system libraries, application framework, runtime, and applications (Wu *et al.*, 2013; Vidas & Christin, 2014).

2.9.3 File System Permissions

These permissions protect the privacy of the user's information (files). In the case of Android, unless the developer specifies otherwise, files from one application are not shared with other applications (Zhang *et al.*, 2013).

2.9.4 Security-Enhanced Linux

For access control, Android employs Security-Enhanced Linux (SELinux). SELinux is a Linux kernel security module that supports access control security policies such as Mandatory Access Control (MAC) systems. It provides a mechanism for enforcing information separation based on confidentiality and integrity requirements, allowing threats of tampering and bypassing application security mechanisms to be addressed and limiting the damage that malicious or flawed applications can cause. It includes a collection of sample security policy configuration files designed to meet common, all-purpose security objectives.

2.9.5 Android Permission Model

Android apps have restricted access to system resources. The permission model manages and restricts access to system resources by linking access to permissions (Felt *et al.*, 2012a). Permissions are requested to access the resources as a whole during the application installation phase, thus linking the application installation with the grant of permissions (A. Felt *et al.*, 2011). As a result, denial is out of the question for the intended user. Until recently, permissions were granted for the duration of the installed application. However, in the most recent versions of Android, the user can scroll through the permissions and select/de-select them. Some app features will not function in such cases due to a lack of required permissions. Application permissions can also be set for other applications (Sarma *et al.*, 2012; Wijesekera *et al.*, 2015). Permissions (how and who) are defined in a protection level attribute, which communicates with the system for this purpose (Benton *et al.*, 2013; Armando *et al.*, 2015).

2.9.6 Inter-Component Communication

Android uses a sandboxing mechanism to protect applications from one another and system resources from applications as one of its security mechanisms. Interactions between applications, on which Android relies for application protection, must take place via a message-passing tool known as inter-component communication (ICC). In Android, intent filters are used to represent the types of requests to which a specific component may respond. An intent message is an event that should be available for actions to be taken, as well as the data that supports those actions. Component invocations come in several flavours, including inter-app, intra-app, implicit, and explicit. Android's ICC allows for late runtime binding involving components from different or similar applications, where calls are not explicit within the code but are enabled via event messaging, which is a significant feature for event-oriented systems. The ICC interaction mechanism for Android has been found to introduce a number of security issues (Chin *et al.,* 2011). Because there is no standard authentication or encryption applied to intent-event messages that interact in components, they can be tampered with or intercepted (Davi *et al.,* 2011). Furthermore, there is no mechanism in place to prevent an ICC callee from misrepresenting its caller's intentions to third parties (DIetz *et al.,* 2011).

Communication of components an Android application is made up of components. Activities, services, broadcasts, and providers are the four types of components (Shekhar *et al.,* 2012). The Android platform offers a secure ICC, similar to IPC in a Unix system. The binder mechanism, which is located in Android's middleware layer, provides ICC. The binder is a custom Linux driver's remote procedure call (Android Developers). Intention leads to ICC. The intent is a message that optionally displays the target along with some data (Gibler *et al.,* 2012). It can be used in explicit communication to identify the receiver's name, or it can be used in implicit communication to determine whether the receiver can access this intent or not. Inter-process communication occurs using a traditional UNIX-style mechanism that is restricted by Linux permissions. The following are the components of Android IPC:

- ***Binder:*** It is a mechanism for handling in-process and cross-process calls via Remote Procedure Call (RPC).
- ***Services:*** Services can provide interfaces directly accessible using binder.

- ***Intents:*** Intent is a communication mechanism that informs the system of the intention to perform a specific action (Yang *et al.*, 2013; Feizollah *et al.*, 2017). For example, if a website is to be opened, the 'intent' to open the corresponding URL is sent to the system. The system would pass the intent to the browser, who would then perform the action specified by the intent.
- ***Content providers:*** A Content Provider makes it easier to use the device's data, such as the contact list or music preferences (Ye *et al.*, 2013). Through Content Provider, an application can access the data provided by other applications, and it can define its Content Providers to share its data as well (Sasnauskas & Regehr, 2014; Yang *et al.*, 2013).

2.9.7 Protected APIs

Protected APIs are resources that are only accessible by the operating system (Peiravian & Zhu, 2013). Cameras, GPS, telephony, Bluetooth, SMS/MMS, and network/data are some examples. The application must define these resources in its manifest in order to use them.

2.9.8 Cost Sensitive APIs

APIs that may incur a fee for use are classified as cost-sensitive APIs, which include telephony, SMS/MMS, data/network, and NFC (Wu *et al.*, 2012). These APIs are on the OS-controlled list of protected APIs that require exclusive approval from the device's user (Sarma *et al.*, 2012).

2.9.9 Application Signing

Before installing apps from the app store, Android requires developers to sign them with a digital certificate. If the app is not digitally signed, the Google Play store and installer package prevent it from being installed. Application signing is used to identify the app's developer and to update the app without requiring complicated procedures or additional permissions (Rastogi *et al.*, 2013). It also enables inter-app communication via well-defined IPC (Vidas *et al.*, 2011). The Package Manager verifies the developer's signature in APK files (Loorak *et al.*, 2014). CA verification of application certificates is not performed by Android. The app signing key generates a digital certificate that includes the public key of a public/private key pair as well as some additional metadata identifying the key's owner. The corresponding private key is held by the certificate's owner. When a developer signs an APK, the signing tool associates the APK with the developer and its corresponding private key by attaching the public-key certificate. This helps Android ensure that future apps are legitimate and come from the app's creator. For users to be able to install new versions as app updates, every app must use the same certificate throughout its lifespan. Applications that are signed with the same certificate can share user IDs (Zonouz *et al.,* 2013).

2.9.10 Sensitive User Data

Some APIs on Android may provide access to user data of protected APIs. Personal information, sensitive input devices, and device metadata are the three types of sensitive user data.

- ***Personal information:*** Users can get an idea of the type of data that can be accessed by the application by controlling the content providers that contain personal information such as contacts and calendars (Do *et al.,* 2015). Any application can access these resources if the user grants the requesting app-controlled permissions. Any application that collects personal information will, by default, restrict the data to that application; however, it can share the data with other applications via IPC and permissions mechanisms (Sato *et al.,* 2013).

- ***Sensitive Data Input Devices:*** Android devices have sensitive data input sensors that enable many applications to interact with external media such as GPS, microphones, and cameras. If a third-party application requires access to these resources, it must request permission from the user (Rastogi *et al.*, 2013; Sarma *et al.*, 2012).
- ***Device Metadata:*** Android limits access to sensitive data, but it may share important information such as user preferences or how a user uses his device. Only applications with appropriate permissions can access the key resources. If permission is not granted, the installation will be halted (Sarma *et al.*, 2012; Do *et al.*, 2015).

2.9.11 Publishing and Distribution of Apps

The Android apps are prepared for distribution to users after publishing. Two key tasks are involved in publishing:

- ***Preparation of the application for release:*** On Android devices, a release version of the app is available for download and installation.
- ***Release of application to users:*** Application release refers to the promotion, marketing, and distribution of the application's released version to users (Seo *et al.,* 2014). App stores like Google Play are used to release new apps. Apps can, however,

also be downloaded from specific websites or via email. By configuring its options, uploading the necessary assets, and then publishing the application, an Android application is made available on Google Play (Sarma *et al.,* 2012).

2.10 Related Work

In recent years, a large volume of research work related to automatic Android malware analysis is proposed by introducing data mining and machine learning approaches, achieving relevantly decent detection performance. These approaches utilise a series of machine learning algorithms to build a prediction model based on extracted features from the Android application package (APK). The features extracted from the description or other information of an application are viewed as metadata-based features. The information is usually shown in app markets directly, like category, description, permission, rating, developer information, etc. On the other hand, the features extracted by dynamic analysis are likewise diverse. When running Android samples, used API calls, using permissions, system calls, or dynamic behaviours like internet access and resource consumption can be utilised to analyse Android applications.

Various machine learning based Android malware analysis techniques which have been proposed in the literature and are described in this section. Particularly, five types of approaches have been proposed to detect Android antimalware, VPN, Trojan, Botnet and Malicious Adware: static, dynamic and hybrid. In the Android platform, there are several works available in the literature due to its popularity and numerous malwares, that exist. I have identified recent relevant works and discussed them here but the related works that are about Trojan detection and Malicious Adware detection are non-existent in the literature. To the best of my knowledge, there is only one related work about Trojan detection that is based on dynamic analysis and not on permissions. In addition, there is no related work on detecting Malicious Adware based on permissions which means I am the only researcher that used permissions to identify Malicious Adware on Android platform.

2.10.1 Static Methods

2.10.1.1 Methods Based on Code Analysis

The first category of work focuses on the analysis of an application's code, whether at the bytecode level or at the source level. In the following, the most representative works by this approach will be discussed. TinyDroid (Chen *et al.,* 2018) is a static Android malware detection mechanism that relies on a two-step process which are first abstracting machine instructions, followed by a machine learning phase. First, the APK file of each app is decompiled into Smali code using a system called Apktool.3 Smali can be seen as a higher-level description of Dalvik bytecode, which will be further abstracted to symbolic instructions by TinyDroid. Then, this malware detection system computes the n-grams of abstract instructions occurring in the code and applies that information as the basis for its classification. Thus, a set of n-grams is computed for each application compared to the set of n-grams extracted from applications that are known to be either malicious or benign. If an application is stated as malicious, the set of n-grams that characterizes its behaviour will be summoned to TinyDroid's malicious applications n-grams database. The security tool is not publicly available. Moreover, the benign applications have been collected from the Google Play Store randomly, however, the real contents of the sample are not disclosed. (Chen *et al.,* 2015) studied the use of a code clone detector, designed for malicious Android software identification. They used a static analysis approach to examine and test the source code of the applications. First, the authors applied dex2jar to convert the Dalvik virtual machine bytecode to JVM bytecode. Then, the Java bytecode was decompiled using the Java decompiler JD-CORE subsequently which is performed on higher-level code. The authors successfully trained NiCad to perform malware detection effectively using a training set consisting of known malicious and benign applications. This method allows malicious applications belonging to certain malware families to be located efficiently and reliably. 'NiCad' is an open-source program that detects similar segments of codes. The described approach was tested using a dataset that contained only malware. (Potharaju *et al.,* 2012) intends to detect repackaged applications which they refer to as 'plagiarized applications' under different levels of obfuscation, containing malware. The favourable precision and recall of such methods are highly linked to the precise value given to these parameters. (W. Zhou *et al.,* 2012) sought to analyse and detect repackaged applications automatically. They implemented an application similarity measurement framework called DroidMOSS that uses a fuzzy hash system to detect and also locate changes in an application's behaviour effectively. Unlike several of the approaches seen in this section, it operates on the Dalvik bytecode directly without needing access to the source code. DroidMOSS relies on the existence of the corresponding original applications in the dataset and may miss some repackaged applications.

2.10.1.2 Methods Based on API Calls and Permissions

The second category of static approaches is concerned with the analysis of the permissions requested by the application, and the various API calls that occur in its source code. The DroidSieve (Suarez-Tangil *et al.*, 2017) examines several syntactical characteristics of applications to classify and detect Android malware. These features are purely static and include the list of API calls occurring in the code, the permissions it requests, and the set of all application components. DroidSieve may not be robust against attacks, application cloning, or Adware since it performs malware detection by looking for patterns in the application's code. (Qiao *et al.*, 2016) introduced a malware detection approach based on automated learning of the permissions and API function calls present in Android Applications. The permissions of the API used in the code, are organized in feature vectors and the classification proceeds by applying three different machine learning (ML) algorithms as Support Vector Machines (SVMs), (Tong & Chang, 2001), Random Forest, and Artificial Neural Networks (ANN). (Wu *et al.*, 2012) proposed DroidMat, a system that draws upon multiple elements of static information, including permissions, intents which are messaging objects that contain information about other components, and API calls to characterize the behaviour of Android applications. With regards to API calls, their model includes the API calls as well as the type of component in terms of service and activity in which the API is called. Also, DroidMat uses a combination of K-Means (Wagstaff *et al.*, 2001) and k-nearest neighbours (KNN) (Altman, 1992). Authors declare that DroidMat is not able to detect a specific type of malware. Malware that extracts the malicious payload from external sources at runtime, instead of preserving it in the application's code itself, this process is called dynamic loading. (Sarma *et al.*, 2012), developed and introduced an alarming type of system that considers both the permissions requested by the application. The category and subcategory of the application as well as the permissions requested by other applications belonging to the same category.

(Peng *et al.*, 2012) use a probabilistic model to assign a risk score to Android applications based on the requested permissions during installation and their category. Each user can then make an informed decision about the risk-return of installing the application. The risk score is calculated in the way that the more permissions an application requests, the higher its score will be. The scheme will encourage developers to reduce the number of permissions their applications request, thus reducing the attack surface of the end user's device. (Enck *et al.*, 2009) proposed a platform called Kirin, which examines the permissions requested by an application to determine if it meets a higher-level security policy. First, this platform extracts the permissions from the manifest file and then compares these permissions to nine rules. The rules are defined by the authors that conservatively overestimate templates of undesirable security properties required by various types of common malware. (Aafer *et al.*, 2013) introduced an approach called DroidAPIMiner to extract Android malware features at the API level concentrating on critical API calls. This approach extracts from the application under consideration the API calls and their package-level information, as well as the requested permissions of the applications.

2.10.1.3 Other Static Methods

Finally, I list in a third category static methods that do not fall into API or source code analysis. Static methods are fast and secure with low resource consumption. Nonetheless, they are not able to analyse encrypted and obfuscated malware. Most of the static methods are usually incapable of dealing with unknown malware and result in false positives. Hence, static analysis may need to be coupled with other security models for efficient malware detection. Static analysis may be based on signature (Sihag et al., 2021), permission (Mathur et al., 2021) or Dalvik bytecode (Sihag *et al.*, 2021b). For better accuracy, a combination of these methods is possible (Arshad *et al.*, 2016). A static approach examines malicious behaviour in an application without executing it. The application is disassembled to its source code and analysed by reverse engineering tools such as Apktool, dex2jar, dexdump, baksmali, dedexer. Specific patterns or signatures are generated through feature extraction of the source code and get compared with the signatures which previously recognized as harmful. The signature-based analysis exploits either cryptographic or similarity measurement algorithms. However, cryptographic hashes are exact and suffer from code obfuscation. Fuzzy hashing (Kornblum, 2006) and SDHash (Roussev, 2010) attempt to measure the similarity level between two files. SDHash is mostly used for image and video files, while fuzzy hashing performs well for text files (Faruki, Laxmi, *et al.*, 2015). Unlike permission-based approaches, a Dalvik bytecode-based analysis consumes power and storage space and fails on native code execution. DroidMOSS (Zhou & Jiang, 2012) implements fuzzy hashing to defeat application repackaging changes. (Faruki, Laxmi, *et al.*, 2015) used fuzzy hash to create variable-length signatures, comparing them with malware signatures in the database. However, this method is unreliable to detect zero-day malware especially when the database is limited. To overcome this problem, the YARA project provides a public repository fed with malware signatures by a community of people. Further, YARA rules help researchers to classify an application. (Wang *et al.*, 2014) extracted high-risk permissions from the manifest file to identify malware. (Li *et al.*, 2018) introduced the permissions related to both malicious and benign characters. (Talha *et al.*, 2015) and (Sanz, Santos, Laorden, *et al.*, 2013) classified the apps as malware or benign based on the permissions. (Verma & Muttoo, 2016) presented a malware detection framework based on machine learning which used Android permissions of an application. (Milosevic *et al.*, 2017) proposed a machine-learning Android malware detector based on permissions and source code analysis. (Kang *et al.*, 2016) introduced a machine learning antimalware based on n-gram opcode feature extraction. (Kim *et al.*, 2012) and (Rastogi *et al.*, 2015) analysed the Dalvik bytecode and monitored privacy leakage to any remote server. It is sometimes hard to find the source of leakage and its target, plus the manipulation of sensitive information by malware is also possible. MLDroid (Mahindru & Sangal, 2021) is a recent framework for Android malware detection using machine learning techniques, API calls and app ratings. A work that presents a web-based framework that helped to detect malware from Android devices. The proposed framework detects Android malware applications by performing its dynamic analysis measures can be found in MLDroid. A machine learning-based malware detection platform is proposed to distinguish Android malware from benign applications. It is aimed to remove unnecessary features by using a linear regression-based feature selection approach at the feature selection stage of the proposed malware detection framework It is a very efficient way to detect malware of unknown family types with very accurate results. The algorithm relatively accurately detects malware in general but considers only precision and recall results (Şahin *et al.*, 2021). GDroid (Gao *et al.*, 2021) is an Android malware detection methodology which is the first one that is based on graph neural networks and is able to detect malware in general with high accuracy. The study introduces a new scheme for Android malware detection and familial classification based on the Graph Convolutional Network (GCN). The general idea is to map Android applications and APIs into a large heterogeneous graph and convert the original problems into a node classification task. Unfortunately, most static methods are unable to analyse encrypted, obfuscated, or unknown malware which results in false positives. However, they are fast and secure with low resource consumption. It is a good idea to use static analysis along with other security models for better efficiency.

The study in KronoDroid (Guerra-Manzanares *et al.*, 2021) was a novel hybrid-featured Android dataset that provides timestamps for each data sample which covers all years of Android history from the years 2008 to 2020 and considers the distinct dynamic data sources.

Researchers presented a new malware detection framework for Android applications that are evolutionary 'HAWK'(Hei *et al.,* 2021). Their model can pinpoint rapidly the proximity between a new application and existing applications and assemble their numerical embeddings under different semantics as described. MAPAS (Kim *et al.,* 2022) is a malware detection platform that achieved high accuracy and adaptable usage of computing resources. Moreover, MAPAS analysed malicious app behaviours based on API call graphs of them by using convolutional neural networks (CNN). NSDroid (Liu *et al.,* 2021) is 'a time-efficient malware multi-classification approach based on neighbourhood signatures in local function call graphs (FCGs). This method uses a scheme based on neighbourhood signature to calculate the similarity of the different applications which is significantly faster than traditional approaches according to subgraph isomorphism. In their work 'NATICUSdroid' (Mathur *et al.,* 2021) a new Android malware detection system that investigates and classifies benign and malware using statistically selected native and custom Android application permissions as features for various machine learning classifiers. One more work is an innovative Android malware detection framework that uses a deep convolutional neural network (CNN). In this system, Malware classification is performed based on static analysis of the raw opcode sequence from a disassembled program (McLaughlin *et al.,* 2017). Another research proposes a novel approach based on behaviour for Android malware classification. In ProDroid (Sasidharan & Thomas, 2021) proposed method, the Android malware dataset is decompiled to identify the suspicious API classes and generate an encoded list. In addition, this framework classifies unknown applications as benign or malicious applications based on the log-likelihood score generated. The DroidRanger is a tool (Y. Zhou *et al.,* 2012) which detects behaviours characteristics in malware from several malicious families. It also relies on a crawler to collect Android apps from existing Android markets and then stores them in a local repository. And then, extracts the fundamental properties associated with each application and organizes them into a central database. Finally, it will verify to check if they display malicious behaviour at runtime. (Arp *et al.,* 2014) created DREBIN, a tool that performs malware detection on the results of a static analysis of the applications. they create 8 feature sets for each app, using data from the Android manifest file. Detection is then performed using SVMs. including permissions, components and requested hardware, API calls and network addresses.

2.10.2 Dynamic Methods

2.10.2.1 System Call Monitoring

The first category of work is related to the observation of system calls. Three main lines of work in this category were identified. (Xiao *et al.*, 2019) proposed a malware detection method on Android applications based on processing system calls. Drawing upon the Long Short-Term Memory model (LSTM) (Hochreiter & Schmidhuber, 1997), a type of neural network model is applied in the processing of natural languages. As opposed to the commonly used n-grams that only consider subsequences of length n, the LSTM model allows the classifiers to draw upon the complete history of the sequence up to a given call. Sanya et al. introduced an approach to detect Android malicious behaviour at runtime using Naive Bayes, the Random Forest, and the stochastic descent gradient algorithms. Therefore, malware could potentially evade this detection scheme if the malicious behaviour does not occur during the training period. (Canfora *et al.*, 2015) also relied upon system calls to perform malware detection. The static methods of (Potharaju et al., 2012) who searched for similar code segments in various applications were based on a similar concept.

2.10.2.2 Monitoring of System-level Behaviour

The second category of dynamic methods concentrates on system-level information other than system calls to detect malicious applications. Some of these approaches also include system calls in their analysis. (Feng *et al.*, 2018) proposed a malware detection system called EnDroid which is based on several types of dynamic behaviour at the system level. EnDroid proceeds in two phases: the learning phase and the detection phase. The system then takes as input the feature vectors generated by malicious and benign applications and trains many basic classifiers. Then, it forms a final classification model by adopting a meta-classifier based on the forecast probabilities of these basic classifiers for each application. Andromaly (Shabtai *et al.*, 2012), is an application that consistently monitors various system measures to detect suspicious activity by applying supervised anomaly detection techniques. Other factors include battery life, CPU usage, the number of packets sent over the network, and the number of active processes. However, as a result, the suggested method would only be useful for identifying persistent, protracted attacks, like DDoS attacks, and less useful for identifying sudden, instantaneous attacks. A fact that could be used for malware detection is that various categories of applications exhibit different runtime behaviours. For example, it would be simple to mandate those application developers include a label identifying the goal and general operation of an application in the AndroidManifest.xml file. The label would then specify a more limited set of acceptable app behaviour the detection mechanisms would refer to this label when performing malware detection. For instance, a clustering algorithm will compare an application labelled as a game with other benign game applications to decide if the former behaves in an abnormal manner. Indeed, (Sarma *et al.*, 2012) relied upon the application's category in the application store to create risk signals. The one-out-of-k access policy (Edjlali *et al.*, 1998), an early access control policy for Java, was also based on a similar principle.

2.10.2.3 Monitoring of User-Space Level Behaviour

A third category of works uses information gathered at the user-space level to detect malicious applications. This typically includes call information at the API (rather than system) level. Semantic and syntactic analysis are combined by the tool RepassDroid (Xie *et al.*, 2018) to automatically identify malicious Android apps. combining the syntactic and semantic functions of the application's API with the semantic function of the critical permissions. Then, it uses learning. generates a call graph of each application. The features of the application (APIs and permissions) are then extracted from this graph to create feature vectors. (Wen & Yu, 2017) introduced an Android malware detection scheme based on the support vector machine (SVM) automatic learning classifier. Their scheme operates on the smartphone of the user and is optimized for this purpose directly. MD5 signature hash, the application will be submitted to the server for further processing, and then features are extracted in the feature extraction module using a combination of static and dynamic analysis. (Bugiel *et al.*, 2011) proposed a security tool called XManDroid (eXtended Monitoring on Android), which dynamically analyses application permission usage to detect and prevent privilege escalation attacks at runtime. This kind of attack happens when a program indirectly calls the code of another program, abusing the privileges of that program. This mechanism specifically targets malware that engages in privilege escalation attacks.

2.10.2.4 Observation of an App's Behaviour Using Other Measurements

In this last category, I classify approaches that perform a dynamic observation of an application's behaviour using other measurements than system calls or user-level information. Unlike static analysis, it is late in that it only detects a violation right at the moment when it is about to occur. It also suffers from coverage limitations, since it only considers a single execution, rather than all possible program executions. And also, unlike static methods which analyse the apps before installation on Android platforms, dynamic approaches demand monitoring of all system calls and resources during the execution of the apps in a test platform i.e., CPU usage, network traffic, battery usage, number of active processes etc. Dynamic methods can deal with obfuscated and encrypted malware due to their runtime behaviour and interactions with the system. Although dynamic methods usually detect both known and unknown malware more accurately, they are slow, resource consuming and vulnerable due to the limitation of code reachability. Hence, they may be unsafe sometimes and require certain expertise and a considerable amount of time yet suffer from runtime detection methods (Vidas & Christin, 2014).

'MCDM' (Mohamad Arif, Ab Razak, Tuan Mat, *et al.*, 2021) which is 'a multi-criteria decision-making based' mobile malware detection system that uses a risk-based fuzzy analytical hierarchy process (AHP) approach to evaluate the Android mobile applications. This research concentrated by using permission-based features to assess the Android mobile malware detection system approach. In another research dynamic analysis was used to detect their features. Therefore, a parameter such as a system call was investigated in this study. The purpose of this research is to detect Android malware based on dynamic analysis [2]. Another research paper proposes a novel detection technique called PermPair (Arora *et al.*, 2020) that builds and compares the graphs for malware and normal samples by extracting the permission pairs from the manifest file inside the application. Yet another research presents a platform named DroidCat (Cai *et al.*, 2019) which is a novel dynamic application classification model to complement those methods that are existing. DroidCat uses various sets of dynamic features based on method calls and inter-component communication (ICC) Intents without involving any permission, application resources, or system calls. One other study proposes an innovative Android malware detection framework based on feature weighting with the joint optimization of weight-mapping and the parameters of the classifier named JOWMDroid (Cai *et al.*, 2021).

Programs are executed in a sandbox environment and their malicious behaviours are observed. Dynamic analysis can be based on Andromaly (Shabtai *et al.*, 2012), TaintDroid (Enck et al., 2014), emulation (Zhang *et al.*, 2015) or memory (Sylve *et al.*, 2012). Yet, dynamic approaches suffer from runtime detection methods (Vidas & Christin, 2014) while they require certain expertise and a considerable amount of time. (Shabtai *et al.*, 2012) used machine learning methods to recognize malware by monitoring system resources including CPU usage, network traffic, battery usage, number of active processes etc. Crowdroid (Burguera *et al.*, 2011) fed learning algorithms with system calls data collected by people in the cloud. Monitoring system calls require a lot of resources and may lead to false alarms particularly when a legitimate application invokes too many system calls. Taintdroid (Enck, Gilbert, Han, *et al.*, 2014) tracked data leakage by monitoring real-time data accesses and labelling sensitive data, although it is hard to determine a malicious outgoing flow. Particularly, when a legitimate application invokes too many system calls, monitoring the system calls may lead to a false alarm while requiring a lot of resources. DroidScope (Yan & Yin, 2012) is a virtual machine introspection framework which detected attacks by monitoring OS and Dalvik semantics. (Portokalidis *et al.*, 2010) developed a tool called 'Paranoid Android' that performs multiple attack detection strategies on remote servers hosting an exact replica of the user's device at the same time. On the user's device, a tracer records all necessary information to produce again its execution accurately. This info includes user input as well as events originating in the Kernel, such as system calls. Several of examined mechanisms are similarly targeted to a specific class of malware, which allows such mechanisms to achieve higher detection rates. More research is needed to comprehend how multiple targeted mechanisms can be combined to achieve complete protection against all types of malware. For this reason, it is important that the benchmarks used in testing security tools include a variety of malware (Zhou & Jiang, 2012).

2.10.3 Hybrid Method

Hybrid techniques benefit from both static and dynamic approaches together for better accuracy. They first analyse an application by a static method. In general, hybrid methods obtain the best results most of the time. Nevertheless, they are very resource and time-consuming due to their complexity. In a hybrid approach, first, the application is analysed through a static method followed by dynamic analysis to enhance the accuracy and overcome both of static and dynamic limitations (Arshad *et al.*, 2016). In general, the highest accuracy is usually obtained by hybrid methods. However, they are very time and resource-consuming due to their complexity. Andrubis (Lindorfer *et al.*, 2016) was a hybrid method that used both Dalvik and system monitoring to detect malware. EspyDroid (Gajrani *et al.*, 2020) was a hybrid method which precisely reflected the analysis of Android apps and prevailed drawbacks of static approaches as well as runtime-dependent parameters.

In their article, researchers introduced a novel TAN (Tree Augmented naive Bayes)-based—hybrid Android malware detection mechanism that involves the conditional dependencies which are required for the functionality of an application among relevant static and dynamic features (Surendran *et al.*, 2020). The next work is a survey aimed to provide an overview of the way machine learning (ML) has been employed in the context of malware analyses. They also conducted survey papers based on their objectives, what kind of information about malware they used specifically, and what type of machine learning techniques they employed (Ucci *et al.*, 2019). DAE is a hybrid model based on a deep autoencoder and a CNN. This mechanism is proposed to improve Android malware detection accuracy. To achieve this, they reconstructed the high-dimensional features of Android applications and employed multiple CNN to detect Android malware (Wang *et al.*, 2019). In the next research article, a new detection approach is introduced based on deep learning techniques to detect Android malware from trusted applications. To achieve that, they treat one system called sequence as a sentence in the language and build a classifier according to the Long Short-Term Memory (LSTM) language model (Xiao *et al.*, 2019).

An EfficientNet-B4 CNN-based model is presented for Android malware detection by employing image-based malware representations of the Android DEX file. This model extracts relevant features from the Android malware images (Yadav *et al.*, 2022). In the following paper, a new classifier fusion scheme based on a multilevel architecture is introduced that enables an effective combination of machine learning algorithms for improved accuracy which is called DroidFusion. The induced multilevel model can be utilised as an improved accuracy predictor for Android malware detection (Yerima & Sezer, 2019). A Machine Learning-based method that utilizes more than 200 features extracted from both static analysis and dynamic analysis of Android applications for malware detection was proposed (Yuan *et al.*, 2015). A platform that is capable to detect Android malware applications is introduced to support the organized Android Market. The proposed framework intended to develop a machine learning-based malware detection framework on Android to detect malware applications and to increase the security and privacy of smartphone users (Aung & Zaw, 2013). CoDroid (Zhang *et al.*, 2021) is a hybrid Android malware detection approach based on the sequence which utilizes the sequences of static opcode and dynamic system call.

Finally, researchers have combined the high accuracy of the traditional graph-based method with the high scalability of the social network analysis-based approach for Android malware detection (Zou *et al.*, 2021).

2.11 Related Datasets

This section explains the need for a comprehensive and reliable dataset to test and validate the malware detection system on Android devices by evaluating several publicly available Android malware datasets spanning from 2012 to 2020.

One of the first attempts in 2012 that is publicly accessible is the genome project (Zhou & Jiang, 2012). The authors of this project gathered 1260 malware samples from Android malware vendors between 2010 and 2011. By analysing the installation, activation, and payload of samples, they use static analysis techniques to define malware behaviour. These techniques only concentrate on a static test and statically scan malicious source code fragments, track API calls, and analyse permission lists. They also tested the effectiveness of the current Antivirus software using their proposed dataset on actual smartphones.

After that, the Drebin dataset (2014) (Arp *et al.*, 2014), provided 123,453 benign samples from 2010 to 2012 and 5560 malware samples from 20 families. They used static features such as network addresses (extracted from disassembled code) and hardware components (extracted from manifest files and suspicious/restricted API calls and used permissions) to train their classification system and assess their dataset.

They gathered a sizable collection of Android Botnet samples representing 14 different Botnet families in order to provide an in-depth analysis of Android Botnets. Their collective dataset includes some Botnet samples from the Android Genome Malware project, malware security blogs, VirusTotal, and samples provided by well-known antimalware vendors. Overall, their dataset consists of 1,929 samples that span the years 2010 (the year the first Android Botnet appeared) through 2014 (Abdul Kadir *et al.*, 2015).

The following four Android malware datasets were created by (HCRL, 2018), AndroTracker, SAPIMMDS, Andro-Dumpsys, and Andro-Profiler. Each malware family in the AndroTracker dataset (Kang *et al.*, 2015) was written by the same creator, who used his or her certifi-

cation (certificate serial number) to create the malware apps. To categorise malware samples, they used similarity scoring connected to creator data and other static features like intent, crucial permissions, and suspicious API calls.
Thirteen malware families are represented by 906 samples each in the dataset SAPIMMDS (Jang & Kim, 2016) from the Korea Internet Security Agency (KISA), along with 1776 benign samples. From March to December 2014, this dataset was generated. They used memory dump techniques to analyse suspicious API call patterns from bytecode to extract call patterns for specific malicious functionality from their dataset.

To detect and categorise malware, the Andro-Dumpsys dataset (J. W. Jang *et al.*, 2016) combines malware-centric attributes with intent-based features. There are 1776 benign samples and 906 malware samples in it. The authors used system commands to execute forged files as well as the serial numbers of certificates, suspicious API call patterns, permission distributions, and intents as feature vectors. Based on these feature vectors, they carried out profiling patterns for each malware family based on the connections between opcode and bytecode (captured by memory dumping techniques). Each APK request from the client side and their profiling patterns were used to calculate a similarity score in the detection and classification engines on the server side.

The concept of behaviour profiling was used by the authors of the Andro-Profiler dataset (J. wook Jang *et al.*, 2016) to extract system calls and system logs from malicious application executions on an emulator. They created a client- and server-based hybrid antimalware system.

The Kharon dataset (Kiss *et al.*, 2016), which used the AndroBlare tool to track information flow between system objects like files, processes, or sockets at the system level and provide a human-readable directed graph for each malware sample, was created by running 7 malware samples on a real mobile device. The analysis behaviour of each graph was thoroughly described by the authors.

(Lashkari, Akadir, *et al.*, 2018) contributed 1500 benign samples and 400 malware samples in 12 families of three categories to the AAGM dataset (benign, general and Adware). On real smartphones that had these samples installed, user interaction scenarios were run to record network traffic. Based on algorithms for machine learning, they conducted their analysis.

Additionally, they made use of the Droidkin project (Gonzalez *et al.,* 2015) for finding relationships between various apps and clustering them based on the source code component's similarity. They made their validation dataset public, which included information about app relationships.

Since 2017, another publicly accessible dataset containing Android malware profile information is the AMD dataset (Wei *et al.,* 2017). They collected 405 Android malware samples from 71 families and 4 categories. Their methodical approach entails analysing malicious components and evaluating malware behaviour according to their priorities.

According to Malton ('Xue *et al.,* 2017), we can get around many anti-emulator strategies, by running samples on real devices. The majority of malware applications require user-interaction scenarios to be tricked, such as BOOT-COMPLETED, as they are sensitive to a trigger point of maliciously intended activation. Additionally, an inclusive dataset may collect as many features as it can, including both static and dynamic features, for more desirable use.

This study (Keyes *et al.,* 2021; Rahali *et al.,* 2020) proposes a new large and comprehensive Android malware dataset called CCCS-CIC-AndMal-2020. The dataset contains 200K benign and 200K malware samples, for a total of 400K Android apps, with 14 prominent malware categories and 191 prominent malware families.

2.11.1 Drawbacks

The aforementioned datasets improved researchers' methodologies in the field of Android malware detection, but they are still plagued by significant weaknesses in their contents. The stated datasets lacked the variety of categories and families as well as the vast amount of malware samples that make up a complete dataset of Android malware. Malware generally hides its true malicious behaviour until it is installed on actual devices. Even though some datasets recorded dynamic features of their samples, they avoided running the apps on actual smartphones and instead only ran the malware on emulators and Android Virtual Devices (AVD).

However, none of the mentioned datasets has completely captured the range of malware behaviour, including discrete features such as memory dumps, permissions, memory usage, battery usage and network usage as well as continuous features like API calls, system calls, logs, and network traffic. The dataset that was used for the research's evaluation had to strike a balance between the number of malware samples and the number of benign samples for the results to be reliable. If not, achieving a high accuracy might not be valid proof of their work. According to Symantec's Internet Security Threat Report (SISTR), (Symantec, 2017) the normal distribution of benign and malicious applications in the real world is 80% to 20% as mentioned before in (Lashkari, Akadir, *et al.*, 2018).

In addition to the reasons mentioned earlier, there are some specific factors that influenced my decision not to use the Android Botnet dataset (Abdul Kadir et al., 2015) and the CCCS-CIC-AndMal2020 dataset (Keyes et al., 2021; Rahali et al., 2020) for my research.

One of the main concerns with these datasets is the lack of comprehensive explanations for the features used. It is crucial to have a clear understanding of the features in a dataset, as they play a vital role in shaping the outcomes and conclusions of any research study. Without proper documentation and explanations, it becomes challenging to interpret the results accurately and draw meaningful insights.

Recognising the importance of transparency and clarity in feature selection, I decided to create my own datasets. By doing so, I could ensure that every feature included in the dataset was well-documented, thoroughly explained, and understood. This approach not only increases the reliability and validity of my research but also facilitates better comprehension and reproducibility by other researchers in the field.

By carefully developing datasets with transparent feature descriptions, I aim to contribute to the scientific community and foster a more robust and rigorous research environment.

Year of Publication	**Name of Dataset**	**Type of Data Capturing**	**Number of Benign Apps**	**Number of Malware**
2012	Genome (Zhou & Jiang, 2012)	Static	-	1260
2014	Drebin (Arp *et al.*, 2014)	Static	123453	5560
2015	AndroTracker (Kang *et al.*, 2015)	Static	51179	4554
2015	Android Botnet dataset (Abdul Kadir *et al.*, 2015)	Hybrid	-	1929
2016	SAPIMMDS (Jang & Kim, 2016)	Hybrid	1776	906

2016	Andro-Dumpsys (J. W. Jang *et al.*, 2016)	Hybrid	1776	906
2016	Andro-Profiler (J. wook Jang *et al.*, 2016)	Hybrid	8840	643
2016	Kharon (Kiss *et al.*, 2016)	Hybrid	-	7
2017	AAGM (Lashkari, Akadir, *et al.*, 2018)	Dynamic	1500	400
2017	AMD (Wei *et al.*, 2017)	Static	-	405
2018	CICAndMal2017 (Lashkari, Kadir, *et al.*, 2018)	Hybrid	1700	426
2019	InvesCICAndMal2019 (Taheri *et al.*, 2019)	Hybrid	5065	426
2020	CCCS-CIC-AndMal2020 (Keyes *et al.*, 2021; Rahali *et al.*, 2020)	Hybrid	200K	200K

Table 2. Currently available Android malware datasets specification

Dataset	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**Installed On**
Genome (Zhou & Jiang, 2012)	-	-	Y	-	-	Y	Y	-	Y	Y	N	Y	Y	N	-
Drebin (Arp *et al.*, 2014)	-	-	Y	-	-	Y	Y	N	Y	Y	Y	Y	N	N	-
AndroTracker (Kang *et al.*, 2015)	-	-	Y	-	-	N	Y	N	Y	Y	Y	Y	N	N	-
SAPIMMDS (Jang & Kim, 2016)	N	Y	Y	N	N	Y	Y	Y	Y	N	N	Y	N	N	Emulator
Andro-Dumpsys (J. W. Jang *et al.*, 2016)	N	Y	Y	N	N	Y	Y	Y	Y	N	Y	Y	N	N	Emulator
Andro-Profiler (J. wook Jang *et al.*, 2016)	N	Y	Y	N	N	Y	N	Y	Y	Y	N	Y	N	N	Emulator
Android Botnet dataset (Abdul Kadir *et al.*, 2015)	Y	N	Y	-	N	Y	Y	N	-	-	N	-	Y	N	RealPhone
Kharon (Kiss *et al.*, 2016)	Y	N	Y	Y	N	Y	N	-	Y	N	N	Y	N	Y	RealPhone
AAGM (Lashkari, Akadir, *et al.*, 2018)	Y	Y	Y	Y	N	Y	N	Y	Y	Y	N	Y	N	Y	RealPhone
AMD (Wei *et al.*, 2017)	-	-	Y	-	-	Y	Y	-	Y	Y	Y	Y	Y	Y	-
CICAndMal2017 (Lashkari, Kadir, *et al.*, 2018)	Y	Y	Y	Y	Y	Y	Y	N	Y	Y	Y	Y	Y	N	RealPhone
InvesCICAndMal2019 (Taheri *et al.*, 2019)	Y	Y	Y	Y	Y	Y	Y	N	Y	Y	Y	Y	Y	N	RealPhone
CCCS-CIC-AndMal2020 (Keyes *et al.*, 2021; Rahali *et al.*, 2020)	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	RealPhone

Table 3. Comparison of publicly available Android malware datasets

A: Utilizing Real-Phone devices instead of emulators.
B: Having network architecture for the experiment set up.
C: Examining real-world malware samples.
D: Having malware activation scenario.
E: Defining multiple states of data capturing.
F: Having trustable fully labelled malware samples.
G: Including diverse malware categories and families.
H: Keeping a balance between malicious and benign samples.
I: Avoiding anonymity and preserving all captured data.

Y: Yes
N: No
(-): None

J: Containing a heterogeneous set of resources.
K: Providing a variety of feature sets for other researchers.
L: For meta-data, includes proper documentation.
M: Including malware taxonomy.
N: Being up to date.

The proposed datasets unlike the above-mentioned datasets are not composed of various categories or families of malware. The proposed datasets are focused which means datasets consist of only one type of software (antimalware and VPN datasets) or Malware (Trojan, Botnet and Malicious Adware). According to my tests and results, those datasets containing one type of app regardless of whether malicious or benign can provide better results than those with entries from various categories.

2.12 Drawbacks of the Current Methods and Proposing Solution

Various methods have been proposed for Android malware detection so far. However, they deal with all types of applications the same way, including Android antimalware and VPN. As far as I know, no efficient methodology has been proposed yet, particularly for identifying Android antimalwares and VPNs with malware presence that may threaten Android users by gaining their trust. This research aims to present quick and robust techniques to detect malicious Android antimalware, and VPNs containing malware such as Trojan, Botnet and Malicious Adware before installation on users' devices. With regards to the high risk that users may face after installation, those platforms utilise machine learning techniques as well as an updating dataset to combat malware. Those platforms with a permission-based analysis can provide a robust, secure, and reliable method able to identify malicious antimalware and VPNs, Trojans, Botnets and Malicious Adwares in Android. The datasets contain 1200 antimalwares, 1300 VPNs and 2593(1058 Trojans and 1535 Benign), 2713(1229 Botnets and 1483 Benign) and 2000 (500 Malicious Adwares and 1500 Benign) records including permissions and the risk of being infected by malware for each record. The risk of each entry in the dataset has been evaluated by VirusTotal scan of that record. I applied an optimised MLP neural network for antimalware and Botnet datasets and used a tune CNN for VPN, Trojan and Malicious Adware datasets for better classification results.

2.13 Difference Between This Research and Other Malware Detection Works

The difference between my research and other Android malware detection works that detect different types of malware can be attributed to several factors such as the machine learning algorithms used, the datasets used for training, the feature extraction techniques employed, and the evaluation metrics used.

My research focuses on developing machine learning-based techniques for detecting specific types of Android malware, such as harmful apps, malicious VPNs, Trojans, Botnets, and Malicious Adwares. Each of my chapters proposes a different method for detecting a specific type of Android malware, using different neural network architectures and training datasets.

On the other hand, other Android malware detection works may focus on detecting different types of malware, such as ransomware, spyware, or phishing attacks. These works may use different machine learning algorithms, such as decision trees or support vector machines, and may employ different feature extraction and selection techniques. Additionally, other works may use different evaluation metrics to measure the performance of their detection systems.

The effectiveness of any Android malware detection system will depend on the specific techniques used, the quality of the data used for training and testing, and the ability of the system to accurately detect malware while minimizing false positives.

The other works detect Android malware with a high detection rate while I am detecting a single type of malware with a high detection rate using my created dataset based on permissions asked by users as well as using a trained and optimised neural network.

- Permission-based analysis: my approach focuses on analysing the permissions requested by an app to detect potentially malicious behaviour. This approach is based on the assumption that apps that request many sensitive permissions are more likely to be malicious.
- Use of neural networks: my approach uses a deep neural network to learn the patterns of malicious behaviour from a large dataset of known malware samples. The neural network is trained to classify apps as either malicious or benign based on their permission requests.
- Efficient feature selection: my approach uses a feature selection method that reduces the number of features (i.e., permissions) needed to accurately classify an app as

malicious or benign. This helps to reduce the computational overhead of the malware detection process.

To the best of my knowledge, I am the first researcher creating such large datasets based on permissions to detect a specific type of Android malware while other Android malware detection works either using other available datasets or creating datasets that contain all types of malware. There are many different approaches to Android malware detection, and the specific differences between my research and other Android malware detection works will depend on the specific works being compared. However, in general, the differences between my research and other Android malware detection approaches may include:

- The specific machine learning techniques used: My approach make use of neural networks and deep learning techniques for Android malware detection, while other works may rely on different machine learning techniques.
- The specific features used: Different Android malware detection works may use different sets of features to train their models. For example, some works may rely on static analysis features such as API calls or Intents, while others may use dynamic analysis features such as system call traces or network traffic.
- The specific types of malware targeted: My work focuses on a specific type of Android malware, while other works may target a broader range of malware types.
- The specific datasets used: The datasets used to train and evaluate Android malware detection models can vary widely in terms of size, composition, and quality. My research uses its own specific datasets, while other works may use different datasets or variations of existing datasets.
- The evaluation metrics used: Different Android malware detection works may use different evaluation metrics to measure the performance of their models, while I used well-known evaluation metrics for each chapter such as accuracy, precision, recall, or F1 score.

In summary, the specific differences between my approach and other Android malware detection works will depend on the specific works being compared but may include differences in the machine learning techniques used, the features and datasets used, the types of malware targeted, and the evaluation metrics used.

3. Research Methodology

Data mining is the process of analysing data in order to discover hidden patterns and unexpected relationships (Sadiq *et al.,* 2018). In general, the dataset that will be used for data mining contains examples (referred to as instances) and several attributes (called features). Data mining employs supervised machine-learning techniques to predict an attribute value that has not yet been observed (referred to as a class label) by establishing the relationship and correlation between features in training instances and their labels, and then applying the train models to testing data (referred to as a classification task).

Machine learning is an artificial intelligence (AI) application that allows systems to learn and improve their experience without being explicitly programmed. The study of machines focuses on software that can access and use data for its own purposes. The process begins with observations or information such as examples, direct experience, or training to find patterns in data and make future decisions based on the examples I provide (Cai *et al.,* 2019). The main goal is to enable computers to learn and change their behaviour automatically, without the need for human intervention or assistance. Controlled learning machines can use labelled examples to predict future events and apply previous knowledge to new data. Based on an analysis of a specified training dataset, the learning algorithm generates an inferred function to predict the output values. After adequate training, the programme can provide objectives for any new input. The study algorithm can also compare its output to the desired, correct output and detect errors to adjust the model accordingly. Massive data analysis is possible with machine learning. This typically yields faster and more accurate results in identifying cost-effective opportunities or risky threats, but it may also necessitate additional time and resources to properly train. Machine learning, in conjunction with AI and cognitive technology, can improve the efficiency with which large amounts of information are processed (Arul & Punidha, 2021).

3.1 Applied Deep Learning Algorithms

- ***Convolutional Neural network (CNN):*** A Convolutional Neural Network (CNN) is a type of feedforward neural network that allows information to flow only forward from input nodes, through hidden nodes, and to output nodes, with no loops or cycles. CNNs are primarily used for pattern recognition tasks. They are effective at detecting simple patterns in data and using those patterns to create more complex ones in deeper layers. Typically, CNNs are composed of convolutional and pooling layers. The convolutional layer detects local features from the previous layer, while the pooling layer combines similar features into a single one. CNNs use shared weights, local receptive fields, and spatial subsampling to solve high-dimensional non-convex problems with parallel and cascaded convolutional filters. These filters allow CNNs to be used for tasks such as regression, image classification, semantic segmentation, and object detection. Compared to traditional neural networks, CNNs require fewer parameters and are easier to train due to weight sharing and processing limited dimensions.

A one-dimensional convolutional neural network (1D CNN) is useful for datasets with a one-dimensional structure, where shorter segments of the feature set can be analysed and the feature's location in the segment is irrelevant. It is particularly useful when vectorized data is used to represent the properties of the items whose state or category is being predicted, such as Android applications. 1D CNN can be used to extract more meaningful feature representations that describe patterns or relationships within vector segments. These features are then processed by a classifier, eliminating the need for separate feature ranking and selection outside of the deep learning model. In summary, 1D CNNs can be used as feature extraction layers for a given classifier, providing a more efficient and integrated deep-learning model.

(Yerima *et al.,* 2021).

- ***Multilayer Perceptron*:** A multilayer Perceptron (MLP) is another name for a Multilayer Neural Network. It is made up of three layers: an input layer, an output layer, and a hidden layer. It has several output units. The hidden layer's units are used as input for the next layer (Joo *et al.,* 2014).

3.2 Standard Machine Learning Algorithms Used to Compare

- ***Decision Tree:*** The structure of DT is similar to that of a tree, with the topmost node representing the root node, the terminal or leaf node holding a class label, and the tree branch displaying the test result of the tree displaying the results of the test. (Ross Quinlan, 1993).

- ***Random Forest:*** RF is an ensemble learning technique that employs a large number of individual decision trees that work together as an ensemble. Every decision tree generates a classification for the input data, which is then collected and illustrated by RF using majority voting (Liaw & Wiener, 2018).

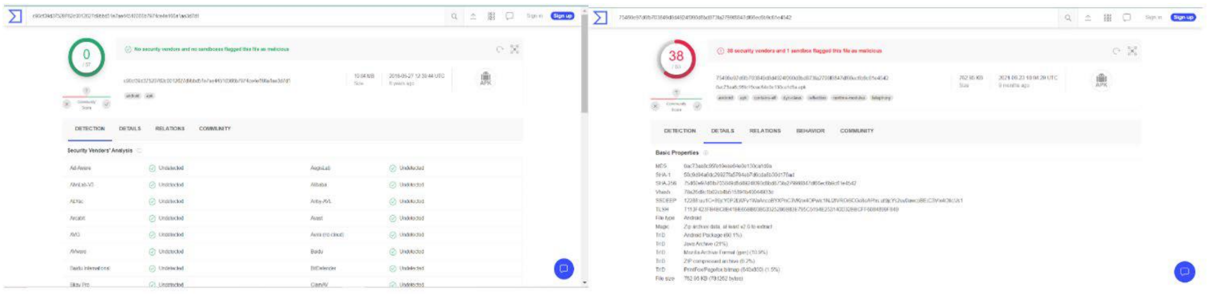
- ***K-Nearest Neighbour:*** K-NN is one of the most straightforward supervised learning methods. A lazy learner is another term for it. This method does not rely on the data structure; whenever a new instance appears, it finds the training samples that are closest to the new instance using distance measures such as Euclidean distance and Manhattan distance. Finally, it determines the class of the new instance using majority voting concepts (Shakhnarovich *et al.*, 2006).
- ***Support Vector Machine:*** SVM is a method for dividing data that uses a hyperplane. It functions as a decision point. It draws the hyperplane at random and then computes the distance between the hyperplane and the nearest data points (also called a support vector). It tries to find the best hyperplane to maximise the margin (Keerthi & Gilbert, 2002).
- ***Naïve Bayes:*** The Bayes theorem underpins the NB concept. It forecasts class membership probabilities or the likelihood that a given tuple belongs to a specific class. It applies to both binary and multiclass classification problems (Domingos & Pazzani, 1997).

3.3 VirusTotal

VirusTotal examines items using over 70 antivirus scanners, URL/domain block listing services, and various tools to extract the signals from the studied content. Any user can use their browser to select a file from their computer and send it to VirusTotal. The primary public web interface, desktop uploaders, browser extensions, and a programmatic API are all available file submission methods from VirusTotal. Among the publicly available submission methods, the web interface has the highest scanning priority. The HTTP-based public API allows submissions to be scripted in any programming language.

3.3.1 How VirusTotal Works

URLs, like files, can be submitted through a variety of channels, including the VirusTotal website, browser extensions, and the API. When a file or URL is submitted, the basic results are shared with the submitter as well as the examining partners, who use the results to improve their systems. As a result, by submitting files, URLs, domains, and so on to VirusTotal, you are helping to improve global IT security. This core analysis also serves as the foundation for several other features, such as the VirusTotal Community, which allows users to comment on files and URLs and share notes with one another. VirusTotal can help detect malicious content as well as identify false positives, which are normal and harmless items that have been flagged as malicious by one or more scanners. The aggregated data on VirusTotal is the result of numerous antivirus engines, website scanners, file and URL analysis tools, and user contributions. The file and URL characterization tools they aggregate serve a variety of purposes, including heuristic engines, known-bad signatures, metadata extraction, malicious signal detection, and so on. Figure 4 declares the files of the benign applications to be virus-free and identifies the malicious applications by the number of detection of antimalware companies in VirusTotal scanner.



0 / 57
No security vendors and no sandboxes flagged this file as malicious
Sign in
Sign up
APK
Community Score
DETECTION
DETAILS
RELATIONS
COMMUNITY
Security Vendors' Analysis
38 / 63
38 security vendors and 1 sandbox flagged this file as malicious
BEHAVIOR
Basic Properties

Ad-Aware	Undetected	AegisLab	Undetected
AhnLab-V3	Undetected	Alibaba	Undetected
ALYac	Undetected	Antiy-AVL	Undetected
Arcabit	Undetected	Avast	Undetected
AVG	Undetected	Avira (no cloud)	Undetected
AVware	Undetected	Baidu	Undetected
Baidu International	Undetected	BitDefender	Undetected
Bkav Pro	Undetected	ClamAV	Undetected

MD5	[illegible]
SHA-1	[illegible]
SHA-256	[illegible]
Vhash	[illegible]
SSDEEP	[illegible]
TLSH	[illegible]
File type	Android
Magic	Zip archive data, at least v2.0 to extract
TrID	Android Package (60.1%)
TrID	Java Archive (21%)
TrID	Mozilla Archive Format (gen) (10.9%)
TrID	ZIP compressed archive (6.2%)
TrID	PrintFox/Pagefox bitmap (640x800) (1.5%)
File size	762.95 KB (781262 bytes)

Figure 4. VirusTotal indication of the benign applications and identification of the malware

3.3.2 Increasing Global IT Security Through Collaboration

VirusTotal scanning reports are made available to the public VirusTotal community. Users can leave comments and vote on whether certain content is harmful. Users can thus contribute to the community's collective understanding of potentially harmful content and identify false positives (i.e. harmless items detected as malicious by one or more scanners). The contents of submitted files or pages may be shared with VirusTotal's premium customers. The VirusTotal file corpus provides valuable insights into emerging cyber threats and malware behaviours to cybersecurity professionals and security product developers. VirusTotal's premium services commercial offering provides qualified customers and Antivirus partners with tools to perform complex criteria-based searches in order to identify and access harmful file samples for further investigation. This assists organisations in discovering and analysing new threats, as well as developing new mitigations and defences.

3.3.3 Real-time Updates

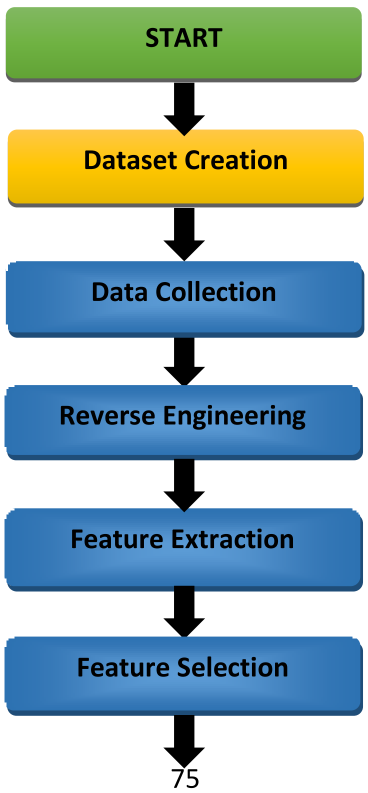
Malware signatures are frequently updated by VirusTotal as they are distributed by Antivirus companies, ensuring that our service uses the most up-to-date signature sets. In some cases, website scanning is performed by querying vendor databases shared with VirusTotal and stored on our premises, while in others, API queries to an antivirus company's solution are used. As a result, whenever a particular contributor blocklists a URL, it is immediately reflected in user-facing verdicts.

3.3.4 Results

VirusTotal not only tells you whether a particular Antivirus solution identified a submitted file as malicious, but it also displays the detection label for each engine (e.g., I Worm.Allaple.gen). The same is true for URL scanners, which can distinguish between malware sites, phishing sites, suspicious sites, and so on. Some engines will provide additional information, such as whether a given URL is associated with a specific Botnet, which brand is targeted by a given phishing site, and so on.

3.4 Overview of Methodology for Novel Dataset Development

This section illustrates the overview of methodology process stages from start to end. There are total of three main stages in the process which are Dataset creation, Pre-processing and Classification/Detection. Figure 5 demonstrates the overview of methodology Process analysis.



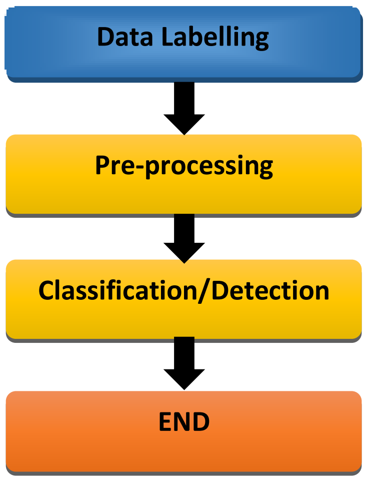


Figure 5. Overview of methodology process analysis

3.4.1 Dataset Creation

The Dataset creation stage consists of five phases: *Phase A: Data Collection, Phase B: Reverse-Engineering, Phase C: Feature Extraction, Phase D: Feature Selection, Phase E: Data Labelling*.

- In this section, I have added more specifics on the sources for benign apps (Google Play, different categories) and malware apps (multiple external sources).
- I have added a note explaining that the app collection process is manual and the challenges involved in identifying VPN apps specifically by examining the AndroidManifest files.
- For feature extraction (Phase C), I have clarified that this is an automated process using static analysis tools on the APK files and manifests.
- For feature selection (Phase D), I have explained that this is a manual process where I selected permissions based on their relevance in detecting malware, referring to existing literature for guidance.
- I have also added a new diagram (Figure 6) to illustrate the overall processes of extracting features from the APK files and then manually selecting the permissions.

Phase A (Data Collection): The initial phase of creating a dataset is data collection. Android apps are usually collected from multiple sources. These apps are stored in Android Application Packages (APK) file format. The benign apps from various categories are collected from Google Play and malicious apps are collected from multiple sources.

Phase B (Reverse-Engineering):

(VirusTotal, n.d.) was used to decompress my datasets and benign applications' APK files. By uploading the APK file to VirusTotal scanner, it decompiles the files to source code folders that provide detailed information about each dataset file, allowing the features to be extracted. Basic properties, permissions, activities, receivers, intent filters by action, intent filters by category, interesting strings, warnings, contents metadata, contained files by type, and contained files by extension are among the useful information.

Phase C (Feature Extraction): The static analysis consists of collecting features that do not require the execution of the code. The advantage of static collection is that it is generally more efficient than dynamic analysis. The traditional way to extract features is that the APK file is saved as a compressed zip file. After that, we must first unzip or unpack APK file in order to view its contents. Then, the APK file is made up of classes, a DEX file, an AndroidManifest.xml file, res, lib, and assets folders. Finally, using VirusTotal scanner, I extracted different types of static features. The "AndroidManifest.xml" file contains permissions information and other types of features. Specifically, the AndroidManifest.xml is parsed to extract requested permissions.

Phase D (Feature Selection): The selected features play a critical role in determining a machine learning model's accuracy. It is also referred to as attribute selection. It is used to reduce dimensionality, which aids in the selection of relevant features. Irrelevant and redundant features can degrade the classification model's quality and accuracy. Higher-dimensional datasets necessitated more storage space and computation time. Selecting relevant features will help to reduce space and time complexity while also increasing accuracy. The primary goal of permissions is to protect users' privacy. Apps must ask for permission to access user-sensitive data and system features. The system may grant permission itself at times or may prompt users to accept the request. Permissions are primarily declared in the "AndroidManifest.xml" file. Permissions play an important role in detecting malicious Android apps. I consider activities in addition to permissions because they are the starting point for user interaction. An app can have multiple activities, and an activity from one app can be used by another if permission is granted (Suresh *et al.*, 2019) (Dhalaria & Gandotra, 2021). Table 4 elaborates on the protection level of Android permissions, descriptions, and examples of the permissions (Mohamad Arif, Ab Razak, Awang, *et al.*, 2021).

While declared permissions in the app manifest provide a good baseline feature set, there are some limitations to only using the AndroidManifest.xml file. Malware authors could intentionally omit certain dangerous permissions to avoid suspicion, and then access protected data or resources directly via Java reflection or native code without checking for permissions. This unauthorized API usage would not be detected by just examining the manifest. Similarly, advanced static analysis tools that decompile and deeply analyse the full bytecode can reveal illegal usage of APIs even without the corresponding permission declared. So, a feature extraction approach that relies solely on manifest permissions could potentially miss some behaviours that indicate malicious intent, reducing detection accuracy. However, manifest permissions still provide a strong signal, as most benign apps properly declare the permissions they need. To augment the approach, deeper static analysis of the code and bytecode can be performed to catch improper API usage not backed by a permission. But manifest permissions themselves serve as excellent baseline features for malware detection, as they clearly indicate the intentions of the app developer.

Protection Level	Description	Permission Examples
Normal	Users and apps are not at risk. The permission was automatically granted, and the user did not revoke it.	ACCESS_LOCATION_EXTRA_COMMANDS, ACCESS_NETWORK_STATE, ACCESS_NOTIFICATION_POLICY, ACCESS_WIFI_STATE
Dangerous	The user is at high risk. Apps must prompt the user and wait for approval.	ACCESS_MEDIA_LOCATION, ACCESS_FINE_LOCATION, ACCESS_BACKGROUND_LOCATION, ACCEPT_HANDOVER
Signature	Apps signed with the same certificate are granted.	BIND_ACCESSIBILITY_SERVICE, BIND_AUTOFILL_SERVICE
Signature Or System	Apps in a dedicated folder and signed with the same certificate are granted.	BATTERY_STATS BIND_CALL_REDIRECTION_SERVICE

Table 4. Android permission protection level

Phase E (Data Labelling): The Android apps (APK files) obtained from the previous phase are scanned using the VirusTotal tool for labelling purpose. It means that once we upload the

APK file into the VirusTotal Scanner, the antimalware companies that incorporate VirusTotal need to flag the APK file as malware so that we ensure the APK file is malicious and then we can label it as ‘1’ which is malware. The benign apps are labelled as '0' and the malicious apps are labelled as '1' in the dataset.

While declared permissions in the app manifest provide a good baseline feature set, there are some limitations to only using the AndroidManifest.xml file. Malware authors could intentionally omit certain dangerous permissions to avoid suspicion, and then access protected data or resources directly via Java reflection or native code without checking for permissions. This unauthorized API usage would not be detected by just examining the manifest. Similarly, advanced static analysis tools that decompile and deeply analyse the full bytecode can reveal illegal usage of APIs even without the corresponding permission declared. So a feature extraction approach that relies solely on manifest permissions could potentially miss some behaviours that indicate malicious intent, reducing detection accuracy. However, manifest permissions still provide a strong signal, as most benign apps properly declare the permissions they need. To augment the approach, deeper static analysis of the code and bytecode can be performed to catch improper API usage not backed by a permission. But manifest permissions themselves serve as excellent baseline features for malware detection, as they clearly indicate the intentions of the app developer.

3.4.2 Proposed Datasets

I present novel datasets (antimalware, VPN, Trojan, Botnet, malicious Adware) that are created manually based on Android app permissions for the purpose of detecting malware in Android platform. In order to do this, I downloaded a large number of Android malware from various families as well as a large number of benign apps from different categories from Google Play and other resources. To examine all APK files and extract app permissions, I used VirusTotal scanner. I classified APK files using over 70 trusted antimalware detection engines. The process of extracting and selecting features is shown in Figure 6.

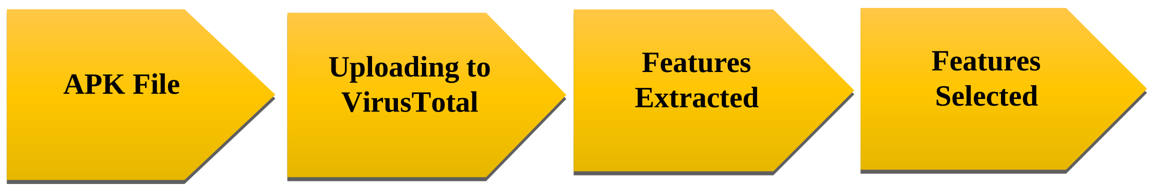


Figure 6. The process of extracting and selecting features

The list of Android malware types and the number of samples are listed in Table 5. I put all the information in a file to make the dataset usable CSV file format, which is simple to open and process. Each dataset contains hundreds of columns which are specific permissions and the label, which is the last column. The first row of datasets describes column titles, and the remaining rows contain features from Android malware and benign applications. All values are in binary format, which means they are either ‘0’ or ‘1’. When an app requires permission, the value in the corresponding dataset entry is ‘1’, and when an app does not require permission, the value is ‘0’. Based on the VirusTotal report, an Android app that is recognised as malware by most antimalware companies is considered risky, and the value in the label column is set to ‘1’, indicating a malware.

Dataset	Number of Samples		Number of Features
Antimalware(Seraj, 2021)	1200	Benign: 869	328
		Malicious: 331	
VPN(Seraj, 2022b)	1300	Benign: 1179	184
		Malicious: 121	
Trojan(Seraj, 2022a)	2593	Benign: 1535	449
		Malicious: 1058	
Botnet(Seraj, 2023a)	2713	Benign: 1483	453
		Malicious: 1229	
Malicious Adware(Seraj, 2023b)	2000	Benign: 1500	400
		Malicious: 500	

Table 5. List of Android malware datasets and the number of samples

3.4.3 Preprocessing

Preprocessing also known as ‘Data Cleaning’. The dataset containing permissions extracted from Android applications and expressed as static properties in comma-separated values (CSV) format is reserved for testing and training. Dataset pre-processing removes NAN and duplicate values. Family and category features were removed from the study that used binary classification. Malware samples are assigned a value of ‘1’, while benign samples are assigned a value of ‘0’. In this section, initially, I performed a ‘missing value check’. For this purpose, I performed a lost data operation and managed empty data. In most cases, the data is missing or contains null values known as NANs. There are numerous solutions to this problem: Cleaning the dataset with the NumPy library of the Python language to find all the NAN data and specifying that there are multiple NAN data in each column, and then deciding whether the data is to be deleted or replaced from the dataset. The average data column is replaced by NAN data. NAN values are also removed from rows with labelled columns.

The data should not contain any missing values. If it does, either the missing data should be removed, or some type of missing value imputation needs to be performed. Then I performed ‘Data Type Conversion’. For doing this, when using data, we must ensure that we use the correct data and ignore items that will result in errors and unrealistic results. I used the Pandas library for this, which returns the dataset's correct data types.

3.4.4 Classification and Detection

I used 5-fold cross-validation on the datasets to test the model's generalisability. The dataset is randomly divided into 5 subsets for 5-fold cross-validation. The data is then subjected to 5 cycles of training and testing. One of the 5 subsets is excluded from the training process and used for testing in each cycle. This is repeated 5 times until each of the 5 subsets has been tested once. Each cycle yields a classifier along with its performance metrics. If the variance between these metrics is high, the classifier is over-fitted and does not generalise the dataset properly. The mean values of the performance metrics are reliable if the variance is low. All experiments were performed on 64-bit Microsoft Windows 11 pro–operating system and using hardware with intel(R) Core (TM) i5-8365U @ 1.60GHz 1.90 GHz CPU, 16.00GB RAM, and an Intel UHD Graphics 620 GPU.

3.4.5 Evaluation Metrics

For the experimental analysis, I used the Python programming language and the Scikit Learn, Keras and TensorFlow libraries. As evaluation metrics, I used Accuracy, Precision, Recall, and F1; these metrics are described in equations 1, 2, 3, and 4, respectively. The abbreviations for true positive, true negative, false positive, and false negative are TP, TN, FP, and FN. Accuracy in Equation 1 demonstrates overall performance. Precision, which is calculated using equation 2 and describes the percentage of predicted malware, is another important metric. Equation 3 defines the Recall metric or the percentage of malware that is correctly classified. The F1-score is a ‘0’ to ‘1’ number that represents the harmonic mean of precision and recall as calculated by equation 4. The basic four performance measures of a binary ML-based classifier are:

- True Positives (TP) — The number of positive samples correctly classified as such. The number of test instances whose true and predicted values are ‘1’, is divided by the number of test in- stances whose true value is ‘1’.
- False Positives (FP)—The number of negative samples misclassified as positive. The number of test instances whose true value is ‘0’ and the predicted value is ‘1’, is divided by the number of test instances whose true value is ‘0’.
- True Negatives (TN)—The number of negative samples correctly classified as such. The number of test instances whose true and predicted values are ‘0’, is divided by the number of test in- stances whose true value is ‘0’.
- False Negatives (FN)—The number of positive samples misclassified as negative. The number of test instances whose true value is ‘1’ and the predicted value is ‘0’, is divided by the number of test instances whose true value is ‘1’.

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{1}$$

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

$$F1 = \frac{2\ x\ Precision\ x\ Recall}{Precision + Recall} \tag{4}$$

I consider receiver operating characteristic (ROC) analysis in addition to well-known evaluation metrics. A scatterplot yields a ROC curve by graphing the true positive rate (TPR) by equation 1 versus the false positive rate (FPR) which uses equation 2 as the threshold varies across the range of values.

$$TPR = \frac{TP}{TP + FN} \tag{1}$$

$$FPR = \frac{FP}{FP + TN} \tag{2}$$

The area under the ROC curve (AUC) ranges from ‘0’ to ‘1’, with ‘1’ indicating ideal separation, in which case no misclassifications occur. An AUC of ‘0.5’ indicates a binary classifier that is no better than flipping a coin, whereas an AUC of x ‘0.5’ can be converted to an AUC of 1 x > 0.5 by simply reversing the classifier's sense. Finally, the AUC can be interpreted as the probability that a randomly chosen positive instance will outperform a randomly chosen negative instance (Bradley, 1997).

3.5 Static Analysis

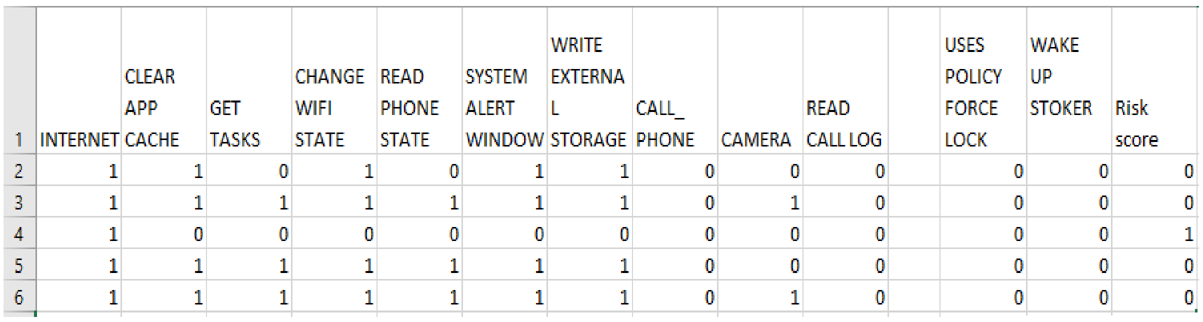
Static analysis is an approach that does not require an application to be run and is considered passive. As the detection is done before the execution of the application, there is no impact on the system from any malicious behaviour. The manifest file, which is a component of the APK file, provides information for static analysis, including the hardware properties, permissions, themes, and activity properties for the application. The tags in the manifest file are used to define the application's permissions, such as internet access, camera access, and file reading and writing (Bayazit *et al.,* 2022).

3.6 Android Malicious Antimalware Detection Methodology

3.6.1 Proposed Dataset

I started my research based on the hypothesis that fake Android antimalware are recognisable through the special privileges that they require to be installed on the device. Hence, I downloaded over 1200 APK files of Android antimalware mostly from Google Play and other websites such as “androidapksfree.com”. I analysed all APK files using VirusTotal scanner to extract all their features including internet access and other required app permissions. Moreover, I have used over 70 reputed antimalware detection engines to classify the APK files into two groups, i.e., the regular antimalware apps or the malicious ones pretending to be regular but harmful for the device. Assessments showed that there are 869 regular apps out of the downloaded antimalware while 331 of them are harmful. Besides, there are 328 specific permissions an antimalware may ask for during installation on a device.

To create the dataset, I have put all the information in a file in CSV format which can be opened by many software applications. There are 329 columns in the dataset including 328 specific permissions plus the risk score column determining the harmfulness of the entries. The first row is column titles, and the rest are 1200 APK features extracted from Android antimalware apps. All values are in binary format i.e., ‘0’ or ‘1’. When an app requires specific permission, the value in the related entry of the dataset is ‘1’, accordingly unnecessary permissions of an app are ‘0’. The last column i.e., risk score has also binary values. A downloaded app which is recognised as malware by most Antivirus companies based on VirusTotal.com report is risky and obtains ‘1’ as its risk score. However, another safe Android antimalware has a zero-risk score. The complete dataset is accessible on Kaggle (Seraj, 2021) for further research. Figure 7 shows a small part of the dataset as a sample. The dataset is reliable since the classifications have been made through VirusTotal scanner, a popular web-based antimalware scanning tool which relies on several Antivirus engines and website scanners to identify malicious patterns.



1	INTERNET	CLEAR APP CACHE	GET TASKS	CHANGE WIFI STATE	READ PHONE STATE	SYSTEM ALERT WINDOW	WRITE EXTERNA L STORAGE	CALL_ PHONE	CAMERA	READ CALL LOG	USES POLICY FORCE LOCK	WAKE UP STOKER	Risk score
2	1	1	0	1	0	1	1	0	0	0	0	0	0
3	1	1	1	1	1	1	1	0	1	0	0	0	0
4	1	0	0	0	0	0	0	0	0	0	0	0	1
5	1	1	1	1	1	1	1	0	0	0	0	0	0
6	1	1	1	1	1	1	1	0	1	0	0	0	0

Figure 7. An illustration of a small part of the proposed dataset

In the next section, I have proposed a multilayer perceptron neural network as a classifier for detecting fake Android antimalware. My proposed dataset has been applied for training and verification of the MLP neural network. Other classification algorithms can be used as well for distinguishing original antimalware, with the results presented in chapter 4.

3.6.2 Proposed Classifier

I have used an MLP neural network as the proposed classifier. An MLP includes an additional layer of nodes i.e., more than just the input and output layer. Regardless of the input dimensionality, it turns out that a single-layer perceptron can solve a problem only if the data are linearly separable. The math performed by a multilayer perceptron allows for immense flexibility in the overall function and making it a good estimator in my case. It is a purely mathematical system approximating complex input-output relationships gradually. Hence, large amounts of data help the network to continue refining its weights and thereby achieve greater overall efficacy.

According to my Android antimalware dataset, I have proposed an MLP neural network for malicious antimalware detection illustrated in Figure 8. The dimensionality of the permissions must match the dimensionality of the input layer. Each sample in my dataset includes 328 different privileges which decisions made based on them. Hence, there are 328 input nodes in the neural network structure. Furthermore, the classification here is a yes or no decision making. Accordingly, only one output node is needed even for so many input nodes and one hidden layer is enough for extremely powerful classification. The number of nodes within the hidden layer can be variable and I extensively search to find the optimal number through trial and error. According to my experimental results in section chapter 4, for 328 input nodes and an output node, the optimal number of hidden nodes was obtained 16.

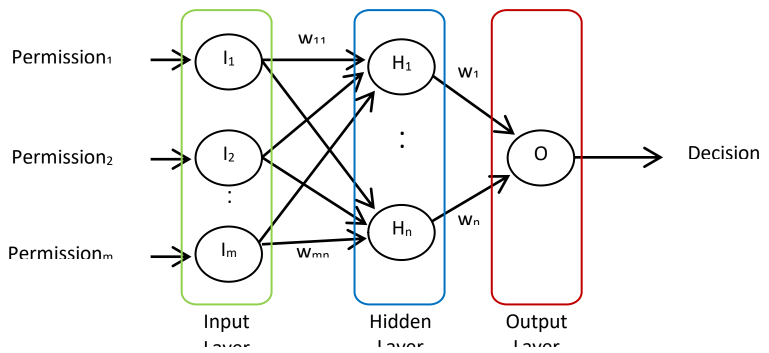


Figure 8. Proposed multilayer perceptron neural network

Data that move from one node to another are multiplied by weights. Numerical data are summed as they arrive at computational nodes, then they are subjected to an activation function. I intended to train the neural network using gradient descent and I needed a differentiable activation function. I applied the standard logistic sigmoid as the activation function for both hidden and output nodes in the MLP structure (K=1, L=1) as shown in equation 1. Figure 9 shows the output value of the activation function versus the input.

$$f(x) = \frac{L}{1+e^{-kx}} \xrightarrow{L=1, K=1} f(x) = \frac{1}{1+e^{-x}} \qquad (1)$$

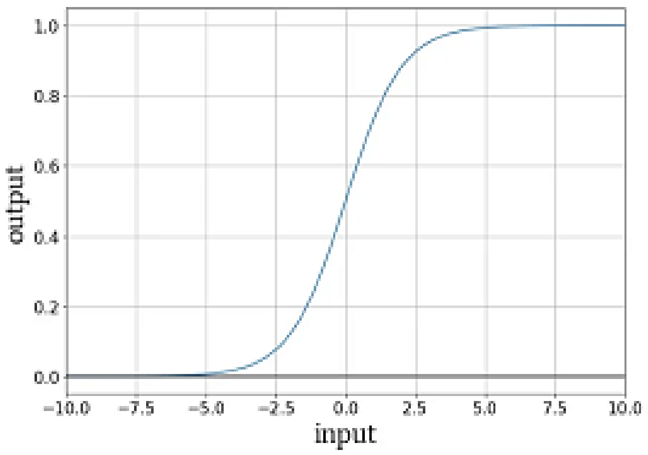


Figure 9. Standard logistic sigmoid function

The logistic activation function is an excellent improvement upon the unit-step function because the general behaviour is equivalent, but the smoothness in the transition region ensures that the function is continuous and therefore differentiable. The shape of the logistic curve as shown in Figure 9 with high derivative near the middle and low variations near the maximum and minimum, promotes successful training with contribution to the stability of the learning of the system. Equation 2 shows that the derivative of the logistic function is related to the original function. Hence, there is no need to use the derivative expression when I have already calculated the output of the logistic function for a given input value.

$$f'(x) = \frac{e^x}{(1+e^x)^2} = f(x)(1-f(x)) \qquad (2)$$

Two pre-node and post-node signals are assumed for each computational node. A pre-node signal is computed by performing a dot product i.e., the corresponding elements of two arrays are multiplied and then all the individual products are summed. The first array holds the post-node values of the preceding layer to the current layer, and the second array includes the weights. The pre-node signal calculation is performed according to equation 3 where N denotes the post-node array of the preceding layer, n is number of nodes in the preceding layer, w denotes the weight vector and $preN_i$ denotes the computed value of pre-node signal for node N_i.

$$preN_i = w.N = w_1N_1 + w_2N_2 + \ ... + \ w_nN_n \tag{3}$$

Since there is no direct path to the output node from input-to-hidden weights, the relationship between these weights and the network's output is very complex as shown in Figure 10. For the sake of simplicity, I have shown the pre-node signal by the word pre plus the node's name and the post-node signal by the node's name. The pre-node signal is input to activation function of the node and the result would be assigned to the post-node signal according to Equation 1.

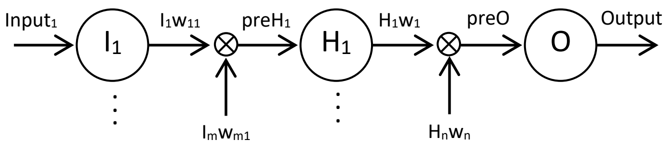


Figure 10. Data-path of the proposed neural network

A gradient gives us information about how to modify weights. I need partial derivatives to make each weight modification proportional to the slope of the error function with respect to the weight being modified as shown in equation 4 in which a is the learning rate, *target* is the expected output, *f'* is derivative of the logistic activation function, *input* and *output* are pre-node and post-node signals respectively.

$$weight_{new} = \ weight_{old} + a \ x \ (target - output) \ x \ f'(input) \tag{4}$$

I also need to update input-to-hidden weights based on the difference between the network's generated output and the target output values, but these weights influence the generated output indirectly. I send an error signal back toward the hidden layer and scale that error signal using both output weights of a hidden node as well as the derivative of that hidden node's activation function. This process is called backpropagation whereby weights are updated based on the weight's contribution to the output error. The steps are shown in equations 5, 6, 7 and 8.

$$FE = output - target \tag{5}$$

$$S_{error} = FE = f'(preOutput) \tag{6}$$

$$\delta_{HO} = S_{error}\ x\ H \tag{7}$$

$$weight_{HO} = weight_{HO} - \alpha\ x\ \delta_{HO} \tag{8}$$

FE is the final error value which is the difference between the post-node signal of the Output node and the correct output value, *f'* is the derivative of the activation function applied to the pre-node signal delivered to the Output node, error signal (S_{error}) is the final error propagated back toward the hidden layer through the activation function of the Output node, d_{HO} represents the contribution of a given weight (hidden layer to output) to the error signal which is finally subtracted from the current weight in order to calculate the new value of that weight, a is the learning rate for changing the step size. For the input-to-hidden weights, the error must be propagated back through an additional layer as shown in Equations 9, 10 and 11.

$$\delta_{IH} = FE\ x\ f'(preOutput)\ x\ weight_{HO}\ x\ f'(preH)\ x\ ipnut \tag{9}$$

$$\rightarrow \delta_{IH} = S_{error}\ x\ weight_{HO} x\ f'(preH)\ x\ input \tag{10}$$

$$weight_{IH} = weight_{IH} - \alpha\ x\ \delta_{IH} \tag{11}$$

According to equation 9, the error signal is multiplied by hidden-to-output weight connected to the hidden node of interest as well as the derivative of activation function on that hidden node's pre-node signal multiplied by the input value. The input value can be thought of as the post-node signal from the input node.

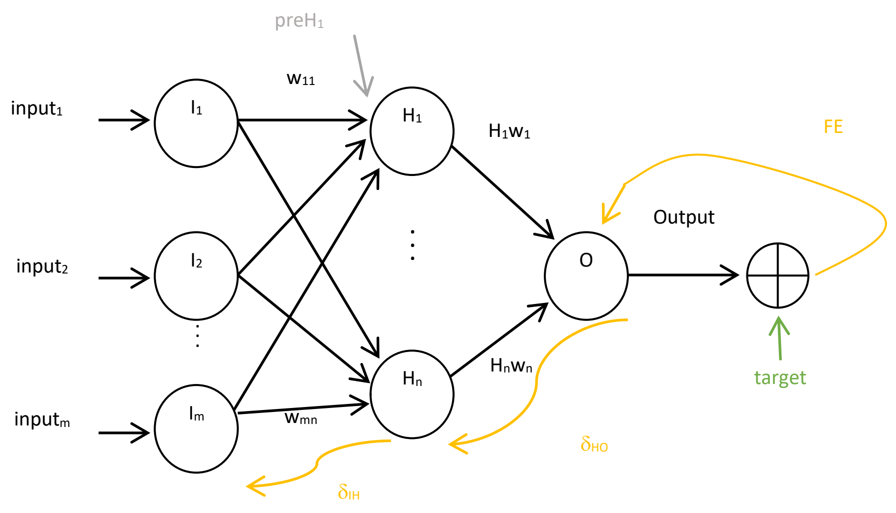


Figure 11. Propagating error back to correct weights

At the next step, training takes place which is a process that allows a neural network to create a mathematical pathway from input to output. A neural network can perform classification because it automatically finds and implements a mathematical relationship between input data and output values via the training. The goal of the training is to provide data that allow the neural network to converge upon a reliable mathematical relationship between input and output. In this chapter, I selected a portion of the dataset as training samples and gave the neural network input values and the corresponding output values. Training process applies a fixed mathematical procedure to gradually modify the network's weights such that the network will be able to calculate correct output values even with input data that it has never seen before. The MLP can approximate the true, generalized relationship between input and output only if I incorporate variety of antimalware into my training samples, unless a deficient and oversimplified relationship would be found by the network. To avoid the neural network of being negatively affected by the order of training samples, I shuffled them after each epoch.

3.7 Android Fake VPN Detection Methodology

3.7.1 Proposed Dataset

This section demonstrates my proposed method for identifying, detecting, and characterising Android VPN apps on Google Play and any other unofficial websites that distribute Android apps. In the literature, it has been shown that it is particularly important to be able to identify the family that malware belongs to. Recent research has shown that to be able to act against a malicious app it is necessary to identify it, and this happens with very high accuracy (D'Angelo *et al.*, 2022; Mahdavifar *et al.*, 2022). Considering that it is an active area of research but there are cases in that malware cannot be classified into a family correctly there is recent research that builds on top of that, and datasets have been developed for individual malware families. For example, there are recent works in the literature that have collected data and developed methodologies to detect individual types of malware such as antimalware, malware that pretends to remove malware but('Seraj *et al.*, 2022; Ullah *et al.*, 2022a; Moodi *et al.*, 2021; Yerima *et al.*, 2021) al., 2021; Yerima et al., 2021). Regular works can identify if an app is malicious or not with very high accuracy as well but are limited in allowing family detection and further research to be easily built on top of that (Wang *et al.*, 2022; Amer, 2021).

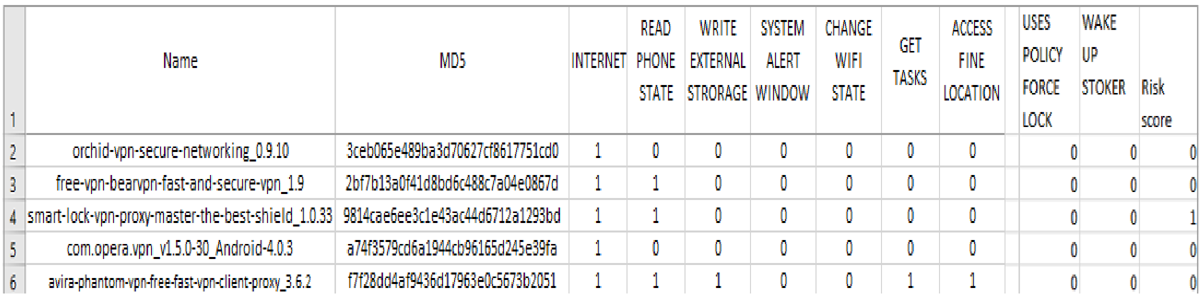
Furthermore, in the industry there has been recently active research on malicious VPNs by Kaspersky, NordVPN and researchers in the ESET research group ('Bahar, 2022; 'Glover, 2022; 'Bratisilva, 2022). This shows that there is an increasing interest in detecting such apps since these are popular among users who want to change their IP address and since there are many such apps available regular users are more susceptible in installing one. In view of the above facts, a novel dataset for malicious VPN detection will help the research community to identify the properties of such malicious apps and will allow further research to be developed, thus improving malware detection approaches.

Moreover, Searching and downloading VPN apps on Google Play and other websites to develop a dataset is not a trivial task. Considering that a given app's profile is available on Google Play and the list of permissions requested for the specific app are available, in the permission live is not necessarily included the use of BIND_VPN_SERVICE permission by the app. A given Android app contains an AndroidManifest file including all the permissions required by the app in relation to the app, service, or specific activity. A service performs long-running operations in the background, while activities are app elements that run in the foreground on a single screen and require user interaction. The BIND_VPN_SERVICE permission will not appear in the list of Android permissions available on Google Play when it is declared by an app developer within the "service" tag. Hence, we have crawled Google Play to download each APK file and then decompiled them to inspect their AndroidManifest files in detail. This way, we make sure that I correctly identify VPN apps at scale. I used Google Play's search feature with keywords like "VPN", "anonymity", "privacy", "censorship", "security" or "virtual private network" to find apps containing that keyword in their description. I applied a 'breadth-first-search' method for any other app considered by Google Play as "similar" as well as for other apps created and published by the same developer.

I collected a total of over 1300 APK files of VPN apps, mostly from Google Play and other websites during a one-month period in November 2021. I extracted all features from APK files, including internet access and other required app permissions, and analysed all the files using VirusTotal scanner which has received reports from more than 70 well-known antivirus detection engines. I classified the APK files into two categories; legitimate or regular VPN apps and malicious apps that pose as legitimate apps but are harmful to the target device. Assessments showed that 9.3% of collected VPNs had any type of malware, while the other portion was safe. Moreover, a VPN might ask for 184 specific permissions at most on an Android platform. To make the dataset accessible to a wide range of software applications, I put all the data in a file in the.csv format. There are 185 columns in the proposed dataset, and 184 of them are specific app permissions plus the risk flag indicating the presence of malware. The first row of the dataset is permission titles, and the rest of the entries are 1300 APK files, which are Android VPN apps. All columns have binary values, so when a VPN requests specific permission, the corresponding column of the dataset entry has a value of '1', and unneeded permissions requested by an app have a value of '0'. Typically, a user can assume that the risk flag of a VPN app, which has been recognised as malware by most of the antimalware corporations based on the VirusTotal report which in the dataset has a value of '1', meaning that it is malicious. Similarly, risk flags for other safe VPN apps are equal to '0'. However, in the literature, this is not the case since the work of (Li *et al.,* 2017) sets the minimum number as 1, (Arp *et al.,* 2014) as 2, (Pendlebury *et al.,* 2019) as 4 and (Salem *et al.,* 2021) as 10, thus since there is not a standardised approach to follow in my experiments, I decided to set the minimum threshold as '1' but all statistical information obtained from VirusTotal is included in the dataset for this value to be changed, The complete version of the dataset is available and publicly accessible on (Seraj, 2022) for other researchers. For the sake of clarity, Figure 12 indicates a small portion of the dataset. My proposed dataset is reliable since the classifications have been made through the VirusTotal mechanism, a popular web-based antimalware scanning tool which incorporates several reputed antivirus engines to identify malicious patterns.

The motivation behind permissions is that an app can request many permissions from a user before it is installed and the user in many cases will not be bothered by what each one is, thus making it easier to install a malicious app. By creating a dataset based on all permissions, I show that malicious VPNs can be detected with very high accuracy when there is ground truth that shows which permissions can lead to malicious activity.

Moreover, VirusTotal, owned by a Google subsidiary, is an excellent choice to collect such data since it is controlled by experts in the area, and it aggregates results from many antivirus products and online web engines which makes it an effective choice for detecting malicious activities. However, VirusTotal does not use any form of Artificial Intelligence and is not able to identify zero-day vulnerabilities on its own, instead, it connects to over 70 antivirus applications and URL block listing services and submits a report to the user.



1	Name	MD5	INTERNET	READ PHONE STATE	WRITE EXTERNAL STRORAGE	SYSTEM ALERT WINDOW	CHANGE WIFI STATE	GET TASKS	ACCESS FINE LOCATION	USES POLICY FORCE LOCK	WAKE UP STOKER	Risk score
2	orchid-vpn-secure-networking_0.9.10	3ceb065e489ba3d70627cf8617751cd0	1	0	0	0	0	0	0	0	0	0
3	free-vpn-bearvpn-fast-and-secure-vpn_1.9	2bf7b13a0f41d8bd6c488c7a04e0867d	1	1	0	0	0	0	0	0	0	0
4	smart-lock-vpn-proxy-master-the-best-shield_1.0.33	9814cae6ee3c1e43ac44d6712a1293bd	1	1	0	0	0	0	0	0	0	1
5	com.opera.vpn_v1.5.0-30_Android-4.0.3	a74f3579cd6a1944cb96165d245e39fa	1	0	0	0	0	0	0	0	0	0
6	avira-phantom-vpn-free-fast-vpn-client-proxy_3.6.2	f7f28dd4af9436d17963e0c5673b2051	1	1	1	0	0	1	1	0	0	0

Figure 12. Proposed dataset sample

Details of the proposed Convolutional Neural Network (CNN), which I designed and trained from scratch as an optimised classifier for detecting malware VPNs, are given in the next section of this research. The CNN has been trained and verified using my VPN dataset and simulation results have been presented in chapter 5. Chapter 5 also shows the classification results performed by other algorithms.

3.7.2 Proposed Classifier

In previous works in the literature that are based on permissions, it has been showing that supervised machine learning algorithms, including neural networks can be very accurate classifiers (Seraj *et al.*, 2022; Yerima *et al.*, 2021; Amer, 2021; Wang *et al.*, 2022). Thus, I used a CNN neural network to detect malware VPNs in my dataset. A CNN is a good estimator in our case due to the immense flexibility of the math performed in the overall function. It is an entirely mathematical system that uses a lot of data to gradually approximate complex input-output relationships. According to my VPN dataset, I designed my neural network as illustrated in Figure 13.

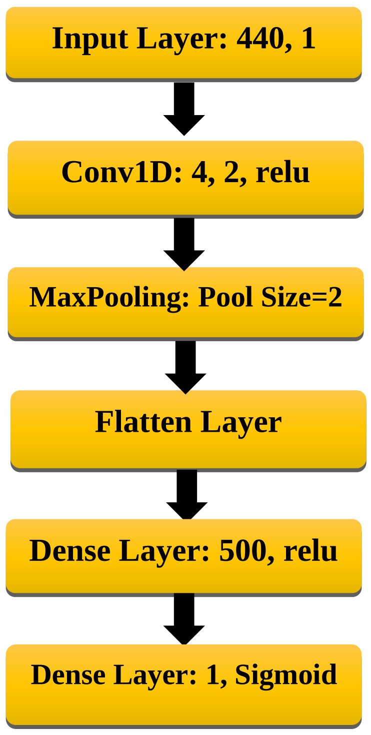


Figure 13. Proposed CNN model

The number of input nodes must match the number of permissions in the dataset, which is exactly 184. In addition, even with so many input nodes, only one output node is needed since the classification here is a yes/no decision-maker. Initially due to the high imbalance of the data I used the SMOTE library to oversample the minority class and then I dropped all columns that contain 85% or more 0s (Permission not required), and then I identified the optimum settings to be a Convolutional 1D layer with a relu activation function, followed by a MaxPooling layer with a size 2, and then a flatten layer. In the next step, there is a dense layer with 500 perceptrons and then the final dense layer for the classification using sigmoid and Adam with a 0.01 learning rate. Finally, the number of epochs was 20 and the batch size was set to 16. The algorithm starts with the convolution which takes place as shown below in equation 1. Where y is the output, n is the length of the convolution represented by x and the kernel represented by h. S is the number of positions the kernel shifts. For hyperparameter tuning, I used a grid search approach.

$$y(n) = \begin{cases} \sum_{id=0}^{k} x(n+i)h(i), & if\ n = 0 \\ \sum_{i=0}^{k} x\big(n+i+(s-1)\big)h(i), & otherwise \end{cases} \tag{1}$$

The relu function used in the convolution layer is shown below in equation 2. Where for the output y and the input x a value is returned from 0 to infinite.

$$y\ (x) = \max(0, x) \tag{2}$$

At the next step the pooling layer takes place to reduce the dimensionality with a size of 2, which helps to reduce any possible overfitting. Then the flatten layer concatenates the output to a flat structure that can be used as an input to a fully connected Multi-Layer Perceptron as shown in equation 3 where Zm is the function output, f is the function name, followed by the function inputs, bias b, and the summary of the inputs.

$$Zm = f\ (x_n\ , w_{mn}) = b + \sum_{m} x_n\ w_{mn} \tag{3}$$

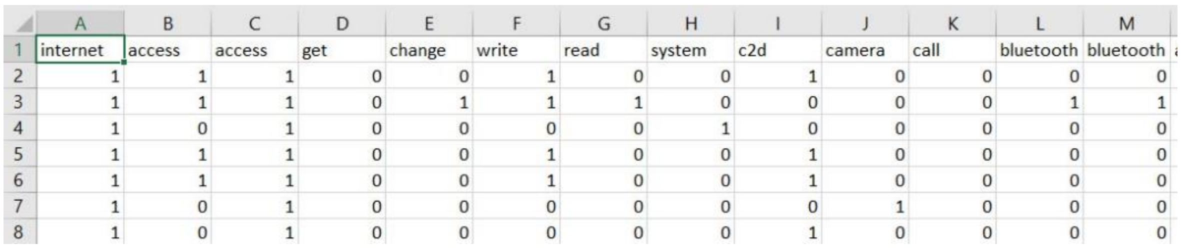
At the next step, the output uses a one hot encoding where the output is either ‘0’ or ‘1’ for an input x based on the sigmoid function shown in equation 4.

$$\sigma(\mathrm{x}) = \frac{1}{1+\ e^{-x}} \tag{4}$$

3.8 Android Trojan Malware Detection Methodology

3.8.1 Proposed Dataset

With regard to Trojan detection in Android platforms, I introduce a new dataset based on Android app permissions. To this extent, I developed an Android Trojan dataset that contains 2593 entries. To do this, I downloaded 1058 Android Trojan malware and 1535 general benign apps from various categories from Google Play. I analysed all APK files using VirusTotal.com to extract all their features including internet access and other required app permissions. Moreover, I have used over 70 reputed antimalware detection engines to classify the APK files. The Android Trojan dataset consists of the following families: BankBot, Binv, Citmo, FakeBank, LegitimateBankApps, Sandroid, SmsSpy, Spitmo, Wroba, ZertSecurity and Zitmo. For the dataset to be in a usable form, I added all the information in a CSV file format which can be easily opened and processed. There is a total number of 450 columns in the dataset that includes 449 specific permissions plus the label which is the last column. The first row in the dataset describes column titles, and the rest are features from 2593 Android Trojans and benign applications APK files. All values are in binary format i.e., ‘0’ or ‘1’. When an app requires permission, then the value in the respective entry of the dataset is ‘1’, and unnecessary permissions of an app are set to ‘0’. An Android app that is recognised as malware by most antivirus companies based on VirusTotal report, is considered risky and the value in the label column is set as ‘1’, being Trojan. However, the other Android genuine apps have ‘0’ value. The complete dataset is accessible on Kaggle (Seraj, 2022a).



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	internet	access	access	get	change	write	read	system	c2d	camera	call	bluetooth	bluetooth
2	1	1	1	0	0	1	0	0	1	0	0	0	0
3	1	1	1	0	1	1	1	0	0	0	0	1	1
4	1	0	1	0	0	0	0	1	0	0	0	0	0
5	1	1	1	0	0	1	0	0	1	0	0	0	0
6	1	1	1	0	0	1	0	0	1	0	0	0	0
7	1	0	1	0	0	0	0	0	0	1	0	0	0
8	1	0	1	0	0	0	0	0	1	0	0	0	0

Figure 14. An illustration of a small part of the proposed dataset

3.8.2 Proposed Classifier

A 1-dimensional CNN sequential architecture has been developed to classify Trojans using the above dataset and the Python programming language with the Keras library. The architecture includes one 1D-CNN layer, followed by a 1D MaxPooling layer, followed by a Flatten layer, followed by 2 dense layers. The architecture is presented in detail in Figure 15. The Specific settings are as follows:

- A learning rate of 0.01 has been used and the optimizer is Adam
- The number of epochs is 6

- The batch size is 16
- The activation functions used are the Relu for the 1D CNN layer and the first dense layer and the Sigmoid for the final dense layer
- Bias has been set to true in the 1D CNN layer

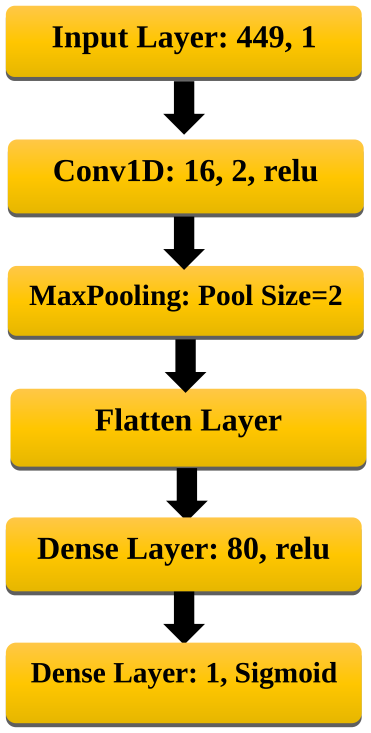
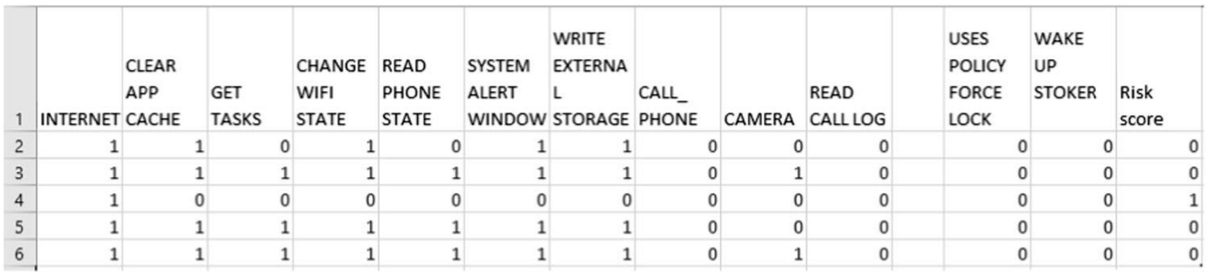


Figure 15. Proposed CNN model

3.9 Android Botnet Malware Detection Methodology

3.9.1 Proposed Dataset

I present a new dataset for Botnet detection in Android platforms. As a result, I created an Android Botnet dataset with 2713 entries. The dataset contains 454 columns, including 453 specific features and the label, which is the last column. The first row of the dataset describes column titles, and the remaining rows contain features from 2712 Android Botnets and benign applications. To do this, I downloaded 1483 benign applications from Google Play and different categories and 1229 Android Botnets. All values are in binary format, which means they are either ‘0’ or ‘1’. Figure 16 indicates a small portion of the dataset. The entire dataset is available on Kaggle (Seraj, 2023a).



1	INTERNET	CLEAR APP CACHE	GET TASKS	CHANGE WIFI STATE	READ PHONE STATE	SYSTEM ALERT WINDOW	WRITE EXTERNA L STORAGE	CALL_ PHONE	CAMERA	READ CALL LOG		USES POLICY FORCE LOCK	WAKE UP STOKER	Risk score
2	1	1	0	1	0	1	1	0	0	0		0	0	0
3	1	1	1	1	1	1	1	0	1	0		0	0	0
4	1	0	0	0	0	0	0	0	0	0		0	0	1
5	1	1	1	1	1	1	1	0	0	0		0	0	0
6	1	1	1	1	1	1	1	0	1	0		0	0	0

Figure 16. A representation of a small portion of the proposed dataset

Feature selection is critical in detecting mobile malware and Botnets. Feature selection can help machine learning algorithms produce more accurate results by removing noise and irrelevant data from datasets. It can also reduce the runtime of machine learning algorithms during training. In this research, permissions are my features. Permissions are used to validate the system's requirements. The developer must declare permissions for use in their applications. Declared permissions are useful and effective in revealing the potential risks of installing Apps.

VirusTotal scanner was used to decompress my Botnet dataset and benign applications' APK files. By uploading the APK file to VirusTotal, it decompiles the files to source code folders that provide detailed information about each dataset file, allowing the features to be extracted. Basic properties, permissions, activities, receivers, intent filters by action, intent filters by category, interesting strings, warnings, contents metadata, contained files by type, and contained files by extension are among the useful information. In addition, VirusTotal declares the files of the benign application to be virus-free and identifies the malware percentage of the Botnet dataset files. I classified the APK files using over 70 trusted antimalware detection engines. The android Botnet dataset includes several families, including Anserverbot, Botmaster, DroidDream, Sandroid, Wroba and Zitmo. I put all the information in a file to make the dataset usable. CSV file format, which is simple to open and process. When an app requires permission, the value in the corresponding dataset entry is ‘1’, and when an app does not require permission, the value is ‘0’. Based on VirusTotal’s report, an Android app recognised as malware by most Antivirus companies is considered risky, and the value in the label column is set to ‘1’, indicating a Botnet. The list of Android mobile Botnet families and the number of samples are listed in Table 6.

Botnet Family	**Year of Discov-ery**	**Number of Samples**	**Type of C&C**	**Motivation**
Anserverbot	2011	244	HTPP	Propagation of possible Malware
Bmaster	2012	6	HTPP	Financial, SMS Stealing
DroidDream	2011	362	HTPP	Data Stealing
Gemini	2010	262	HTTP	Data Stealing
Sandroid	2014	61	HTTP	Financial, Mobile Banking Attack
Wroba	2014	152	HTTP	Financial, Mobile Banking Attack
Zitmo	2012	142	SMS	Financial, SMS mobile Transaction, Authentication Number, (mTAN) stealing

Table 6. List of Android mobile Botnet families

3.9.2 Proposed Classifier

I have used an MLP neural network to detect malware Botnets in my dataset. A multilayer perceptron is a good estimator in our case due to the immense flexibility of the math performed in the overall function. It is a purely mathematical system that gradually approximates complex input-output relationships with large amounts of data. I designed my MLP neural network with one hidden layer as illustrated in Figure 17.

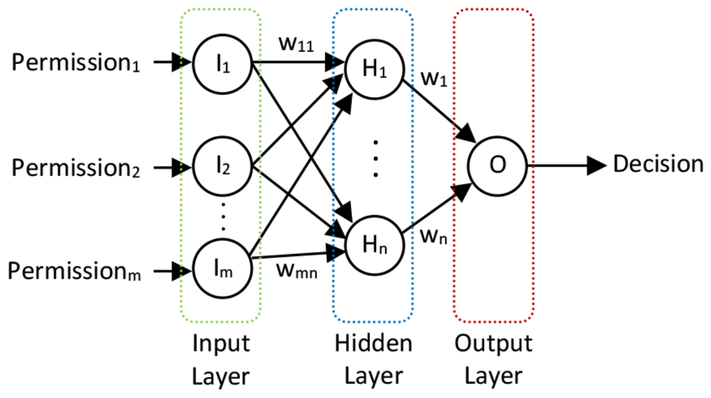


Figure 17. Proposed MLP neural network

The number of input nodes must match the number of permissions in the dataset which is exactly 453. Besides, only one output node is needed even for so many input nodes since the classification here is a yes/no decision maker. For extremely powerful classification, one hidden layer is enough. The number of nodes within the hidden layer can be variable and I have found 454 as the optimum number through trial and error with extensive attempts. According to the neural network structure, data entries are multiplied by weights and subjected to an activation function. As shown in Equation 1 and Figure 18, I used a differentiable activation function called standard logistic sigmoid for both hidden and output nodes in the MLP structure (K=1, L=1) because a gradient tells us how to modify weights. This activation function promotes successful system training while also contributing to the neural network's learning process stability. Each computational node's input is calculated using Equation 2, where N denotes the output of the preceding layer's nodes, w is the weight vector, and n denotes the number of nodes in the preceding layer. In Equation 3, the weights of a node are modified in proportion to the slope of the error function, where the target is the expected output, the learning rate, and f' is the logistic activation function's derivative. It would be unnecessary to use the logistic function's derivative expression for a given input value if we had already calculated the function's output, as shown in Equation 4.

$$f(x) = \frac{L}{1+e^{-kx}} \xrightarrow{L=1, K=1} f(x) = \frac{1}{1+e^{-x}} \tag{1}$$

$$preN_i = w.N = w_1N_1 + w_2N_2 + \ldots + w_nN_n \tag{2}$$

$$weight_{new} = weight_{old} + a\ x\ (target - output)\ x\ f'(input) \tag{3}$$

$$f'(x) = \frac{e^x}{(1+e^x)^2} = f(x)(1-f(x)) \tag{4}$$

The general structure of our classifier is very similar to the MLP neural network described in (Seraj *et al.,* 2022), but the number of hidden nodes differs. As a result, please refer to that paper for more information on a neural network's training and verification mechanisms, as well as how weights are calculated. The neural network finds and implements a mathematical pathway from input to output via training and performs classification automatically.

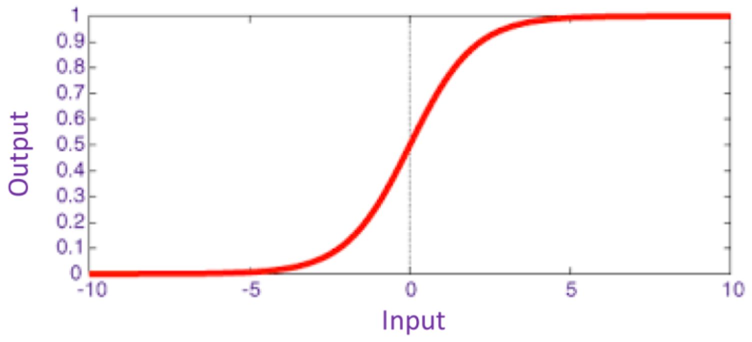
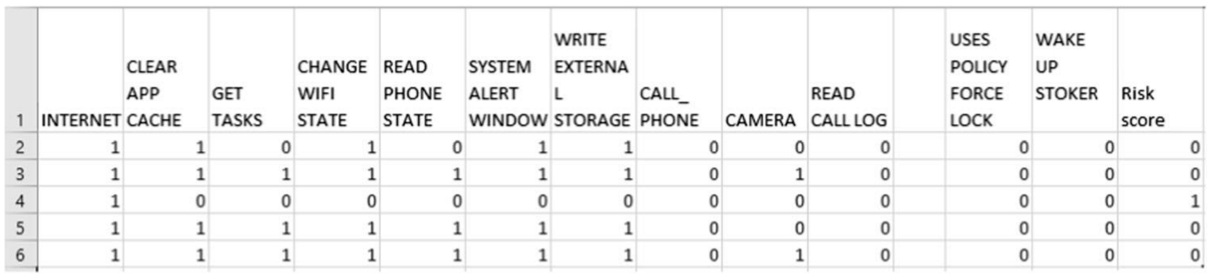


Figure 18. Standard logistic sigmoid function

3.10 Android Malicious Adware Detection Methodology

3.10.1 Proposed Dataset

I present a new self-made dataset based on Android app permissions for malicious Adware detection in Android platforms. As a result, I created an Android malicious Adware dataset with 2000 entries. To do this, I downloaded 500 malicious android Adware and 1500 benign apps from different categories from Google Play. To examine all APK files and extract app permissions, I used VirusTotal online scanner. In addition, I classified the APK files using over 70 trusted antimalware detection engines. The malicious android Adware dataset includes 10 Adware families, including Dowgin, Ewind, Feiwo, Gooligan, Kemoge, Koodous, Mobidash, Selfmite, Shuanet and Youmi. The list of malicious Android Adware families and the number of samples are listed in Table 7. I put all the information in a file to make the dataset usable. CSV file format, which is simple to open and process. The dataset contains 441 columns, including 440 specific permissions and the label, which is the last column. The first row of the dataset describes column titles, and the remaining rows contain features from 2000 malicious Android Adware and benign applications. All values are in binary format, which means they are either ‘0’ or ‘1’. When an app requires permission, the value in the corresponding dataset entry is ‘1’, and when an app does not require permission, the value is ‘0’. Based on the VirusTotal report, an Android app that is recognised as malware by most antimalware companies is considered risky, and the value in the label column is set to ‘1’, indicating a malware. Figure 19 indicates a small portion of the dataset. These Adware families are still actively used in research (Alani & Awad, 2022a). The entire dataset is available on Kaggle (Seraj, 2023b).



1	INTERNET	CLEAR APP CACHE	GET TASKS	CHANGE WIFI STATE	READ PHONE STATE	SYSTEM ALERT WINDOW	WRITE EXTERNAL STORAGE	CALL_ PHONE	CAMERA	READ CALL LOG	USES POLICY FORCE LOCK	WAKE UP STOKER	Risk score
2	1	1	0	1	0	1	1	0	0	0	0	0	0
3	1	1	1	1	1	1	1	0	1	0	0	0	0
4	1	0	0	0	0	0	0	0	0	0	0	0	1
5	1	1	1	1	1	1	1	0	0	0	0	0	0
6	1	1	1	1	1	1	1	0	1	0	0	0	0

Figure 19. A small portion of the malicious adware dataset

Malicious Adware Family	Year of Discovery	Number of Samples
Dowgin	2013	10
Ewind	2015	10
Feiwo	2015	14
Gooligan	2015	14
Kemoge	2015	11
Koodous	2015	3
Mobidash	2015	10
Selfmite	2014	4
Shuanet	2015	10
Youmi	2014	9
Various Families	2014-2020	405
TOTAL		**500**

Table 7. List of malicious Android adware families

- ***DOWGIN*** is a malicious advertising module that is distributed and bundled with other (usually legitimate) programmes. The advertising module is used to display advertising content while also silently gathering and forwarding information from the device. Dowgin provides users with advertising content. If the user is unaware of the module's presence or objects to the nature of the advertising materials displayed, this behaviour may be considered unwanted. The module may also silently leak or harvest sensitive device information such as the device's International Mobile Equipment Identity (IMEI) number, location, contacts, and so on.

- ***EWIND*** is an Adware Trojan that was first discovered in mid-2016 and is capable of displaying unwanted ads, collecting device data, and sending SMS messages to the attacker. The Trojan is distributed by decompiling legitimate Android apps, adding malicious code, and re-packaging them for distribution through third-party Russian-language Android app stores. These Trojanized apps target popular apps such as Grand Theft Auto (GTA) Vice City, AVG cleaner, Minecraft - Pocket Edition, and Avast! Ransomware Removal, VKontakte, and Opera Mobile. It is important for Android users to be cautious when downloading from third-party app stores and use reputable antivirus software to detect and remove potential threats.

- ***FEIWO*** is a malicious Adware for Android devices that sends the victim's phone number, IMEI, and list of installed apps to its servers. This is a common unwanted SDK that should be removed from devices. Furthermore, the Adware employs several techniques to complicate its analysis.
- ***GOOLIGAN*** is a type of malware that poses as a legitimate Android app in order to trick users into installing it, thereby infecting their Android device. It can also spread by infecting apps that are downloaded from untrusted sources. Once the device is infected, the malware installs multiple unwanted apps that are difficult to remove. These apps remain on the device even after performing a factory reset, making it challenging to completely eradicate the malware.
- ***KEMOGE*** is an Adware that masquerades as a popular app; it has spread so widely because it takes the names of popular apps and repackages them with malicious code before making them available to the user.
- ***MOBIDASH*** A special programme module that cybercriminals use to monetize Android games and applications. It displays various types of advertisement messages to the user. The unique feature of Adware.MobiDash.1. origin is that its unwanted activity begins after some time, rather than immediately after the malicious applications containing this module are installed or run. This period is sufficient for the user to forget about the potential source of annoying notifications and advertisements, allowing the malware to remain on the device.
- ***SELFMITE*** Security researchers have discovered a rare Android worm that spreads to other users via links in text messages. When Selfmite malware is installed on a device, it sends a text message to 20 contacts in the device owner's address book.
- ***SHUANET*** behaves more like malware and shares some ancestry with two other Adware families, Kemoge and Shedun, which also root devices and provide system-level persistence to their respective payloads.
- ***YOUMI*** steals a large amount of personal information from an Android device. This includes its GPS and cell tower location, as well as phone identifiers such as the IMEI number and phone number. This differs from the data that affected stolen Apple apps, which included a list of all apps installed on the device as well as the Apple ID email

address associated with the device. Symantec discovered Android's Youmi to be downloading new applications as well.

3.10.2 Proposed Classifier

I used a deep-learning neural network to detect malicious android Adware in my dataset. Because of the immense flexibility of the math performed in the overall function, a Convolutional Neural Network (CNN) is an excellent estimator in this case. It is a completely mathematical system that uses a large amount of data to gradually approximate complex input-output relationships. Figure 20 illustrates my proposed CNN model.

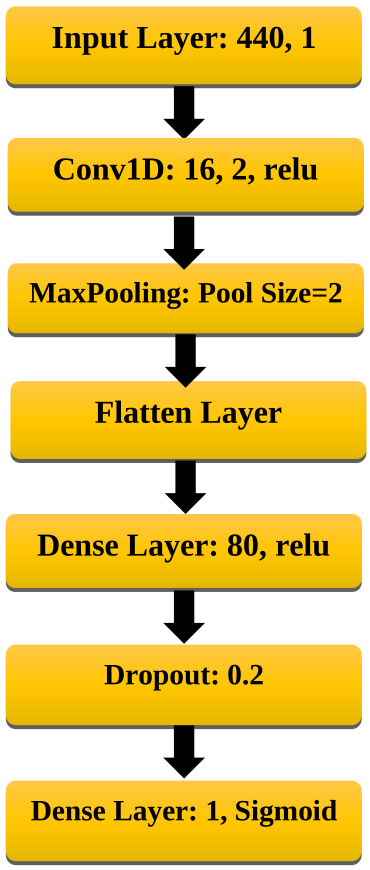


Figure 20. Proposed CNN model

The number of input nodes must be the same as the number of permissions in the dataset, which is 441. Furthermore, even with so many input nodes, only one output node is required because the classification is a yes/no decision-maker. Then we identified that the best settings were a Convolutional 1D layer with a relu activation function, followed by a MaxPooling layer of size 2, and finally, a flatten layer. The following step includes a dense layer with 80 perceptrons, a dropout layer with 0.3, and the final dense layer for classification using sigmoid and Adam with a learning rate of 0.01. Finally, the batch size was set to 128 and the number of epochs was set to 20. The algorithm begins with convolution, as shown in equation 1. Where y is the output, n is the length of the convolution represented by x, and h is the kernel. S is the number of positions shifted by the kernel.

In equation 2, the relu function used in the convolution layer is shown. Where a value ranging from 0 to infinite is returned for the output y and the input x. The pooling layer is applied in the following step to reduce the dimensionality with a size of 2, which aids in reducing any potential overfitting. The flatten layer then concatenates the output to form a flat structure that can be used as an input to a fully connected Multi-Layer Perceptron, as shown in equation 3, where Zm is the function output, f is the function name, followed by the function inputs, bias b, and an input summary. Following that, a dropout layer with 20% of the nodes is used, and the output uses a one-hot encoding with the output being either ‘0’ or ‘1’ for an input x based on the sigmoid function shown in equation 4.

$$y(n) = \begin{cases} \sum_{id=0}^{k} x(n+i)h(i), & if\ n = 0 \\ \sum_{i=0}^{k} x\big(n+i+(s-1)\big)h(i), & otherwise \end{cases} \tag{1}$$

$$y\ (x) = \max(0, x) \tag{2}$$

$$Zm = f\ (x_n\ , w_{mn}) = b + \sum_{m} x_n\ w_{mn} \tag{3}$$

$$\sigma(\mathrm{x}) = \frac{1}{1+\ e^{-x}} \tag{4}$$

4. Android Malicious Antimalware Detection

4.1 Introduction

The number of Android devices has grown exponentially in the last few years. The Android market share is very high, while such devices provide computational power and connectivity, which allow users to store personal, sensitive, and confidential data, such as messages, credit/debit card details, files, and photos. The popularity of Android devices has made them an attractive target for cyber attackers. To protect devices from malware, security firms are usually pushing a virus-scanning app of some sort. However, according to a report from AV-Comparatives, an Austrian organization that specialized in testing antivirus products, two-thirds of all Android antivirus apps are fake and don't operate as advertised (Faruki, Laxmi, *et al.*, 2015). The report was the result of an exhaustive testing process in which 250 Android antivirus apps available on the official Google Play Store were examined. Antivirus apps detecting themselves as malware showed the infirmity of the Android Antivirus industry which appears to be filled with non-real cyber-security vendors. The company researchers searched for and downloaded 250 antimalware security apps by various developers from the Google Play Store. Researchers installed each antivirus app on a separate device, having the test device download 2000 of the most common Android malwares automatically. As the report mentions, only 80 apps detected over 30% of malicious apps and had zero false alarms during individual tests meaning that most of the antiviruses were unable to find malwares. Most of the Antiviruses appear to have been developed either by amateur programmers or by software manufacturers that are security-focused used from a business point of view. In addition to that, tens of apps displayed the same user interface and were more interested in showing ads, rather than having a fully running malware scanner. They usually ask for and usually receive enough permissions that allow for the collection of personal user data such as the model of the phones, live GPS polling, phone numbers and other personally identifiable information. Many antivirus apps use a whitelist/blacklist approach instead of looking at the code. They would mark any installed apps as malicious if the app's package name was not included in its whitelist. According to studies and research that was done on android malware detection, useful solutions have been provided so far mainly based on machine learning techniques. Many datasets have been prepared and based on them; various approaches have been presented to identify android malwares (Zhou & Jiang, 2012). However, the current free and commercial antimalware are mainly based on signature, which is symptomatic that a threat must be widespread to be recognised. Existing methods usually treat all types of android applications in the same way to identify its harmfulness without considering the role of them. Thus, their approach is not usually feasible and useful for the end user.

Given the vital role of antimalware applications, in this chapter, I present a method for the detection of rogue android antimalware applications using a Multi-Layer Perceptron neural network and creating a dataset of android antimalware applications for training and testing the MLP network. I define as rogue android antimalware application an application that is pretending to be antimalware, but in fact its purpose is to deceive Android users and damage their security and privacy. To achieve this, I first hypothesise that a rogue antimalware can be identified based on the requested permissions. I created a dataset of 1200 antimalware applications with 328 specific permissions which might be asked during the installation and identified the harmful antimalware applications in my dataset based on the recognition of many reputed antimalware companies using the report from VirusTotal scanner. Moreover, I define malicious antimalware as rogue apps pretending to be antimalware but are malware instead and deliver a multi-layer perceptron neural network optimised for identifying malicious antimalware applications. The experimental results show that my proposed neural network outperforms other well-known classifiers in accuracy, precision, and recall. In addition, the proposed method is feasible, straightforward to implement and fast due to a limited number of nodes in the hidden layer of the neural network. The classification is performed fast with reasonable computational resources since decisions made are only based on the permissions that an antimalware asks for. The proposed method can be used to accurately detect harmful android antimalware applications before these are installed on a user's Android device.

This chapter delivers the following contributions:

- I used a new dataset of android antimalware based on application permissions which is available in Kaggle.
- I trained and customised a multi-layer perceptron (MLP) neural network to identify harmful android antimalware.
- I evaluated my proposed method using well-known metrics with the results showing that malicious antimalware can be detected with very high accuracy.

4.2 Background

Malicious software (abbreviated malware) is an executable programme that can perform actions after gaining control in some way. It is not a new phenomenon; it has been around for decades. Cybercriminals attack and harm businesses and consumers with various types of malware (viruses, Trojans, worms, spyware, and so on). This is not to be confused with Phishing, which is also malicious. Phishing, on the other hand, cannot actively sniff out your passwords.

Hackers can steal credentials, steal secrets, and compromise customers' identities if Anti-malware software is not installed. Having an antimalware solution in place, on the other hand, can prevent and safeguard sensitive information (both your company's and your clients') from being exposed to cybercriminals.

antimalware software uses three strategies to protect systems from malicious software: signature-based detection, behaviour-based detection and sandboxing.

Signature-based malware detection: Signature-based malware detection identifies new malicious software by using a set of known software components and their digital signatures. Signatures are created by software vendors to detect specific malicious software. The signatures are used to recognise previously identified malicious software of the same type and to mark new software as malware. This method is useful for common types of malware, such as keyloggers and Adware, which share many similarities (Rosencrance, 2021; Fraudwatch, 2023).

Behaviour-based malware detection: By employing an active approach to malware analysis, behaviour-based malware detection enables computer security professionals to identify, block, and eradicate malware more quickly. Malware detection based on behaviour identifies malicious software by examining how it behaves rather than what it looks like. Signature-based malware detection is being phased out in favour of behaviour-based malware detection.

Machine learning algorithms are sometimes used to power it (Rosencrance, 2021; Fraudwatch, 2023).

Sandboxing: Sandboxing is a security feature in antimalware that allows potentially malicious files to be isolated from the rest of the system. Sandboxing is frequently used to filter out potentially malicious files and remove them before they can cause harm. When you open a file from an unknown email attachment, for example, the sandbox will run it in a virtual environment and only give it access to a limited set of resources, such as a temporary folder, the internet, and a virtual keyboard. If the file attempts to access other programmes or settings, it will be blocked and the sandbox will terminate it (Rosencrance, 2021; Fraudwatch, 2023).

The importance of antimalware applications extends beyond simply scanning files for viruses. antimalware can assist in the prevention of malware attacks by scanning all incoming data and preventing malware from being installed and infecting a computer. antimalware software can also detect advanced malware and protect against ransomware attacks.

Antimalware programs can help in the following ways:

- prevent users of from visiting websites known for containing malware.
- prevent malware from spreading to other computers in a computer system.
- provide insight into the number of infections and the time required for their removal.
- provide insight into how the malware compromised the device or network.

Floppy discs were used to spread malware in the early days of personal computers. Then came email and the internet, which opened up entirely new avenues for malware distribution. Antivirus software was created as viruses became more common. Cybercriminals became more sophisticated in their attacks, necessitating the development of a more comprehensive approach to cyber security, ushering in antimalware.

In summary, antimalware software was designed to combat all types of malware, not just computer viruses. In contrast to simple Antivirus software, antimalware does more than just scan email attachments and alert you to potentially harmful websites. Modern antimalware solutions protect by monitoring data sent over networks. It offers far more protection than a simple Antivirus programme.

App and software downloads are currently one of the most common ways for malware to enter your organisation. However, unlike early viruses, modern malware can do much more than corrupt or shut down a single device. It has the potential to compromise an entire network, allowing criminals to steal account credentials or silently hijack secure sessions. As a result, antimalware solutions have become the industry standard for safeguarding businesses against all types of malware designed to harm end users.

The distinction between malware and viruses is straightforward: a virus is a subset of malware. One type of malware is referred to as a virus. Malware, on the other hand, is a broader term that encompasses all types of malicious software.

Antimalware and Antivirus are not the same thing. To begin, Antivirus relies on known virus signatures to detect specific viruses or types of viruses, similar to a flu shot. antimalware, on the other hand, is comprised of non-signature-based anomalies that detect unknown anomalies (malware that may not have been detected before).

An antimalware solution protects your company more broadly by monitoring data transferred over networks for a variety of potential threats. It also detects and eliminates threats to safeguard your entire organisation and its customers. Consumer-level antivirus software, on the other hand, seeks to block all threats to a single device.

Antimalware and Antivirus software have different goals. A simple Antivirus tool, for example, may detect a threat on a single machine or device. As a result, Antivirus software is ineffective at protecting your entire company from malware attacks. Without a doubt, antimalware software provides more comprehensive and robust protection than antivirus software.

Malware can effectively destroy your brand in the event of a serious attack. Failure to monitor and detect a malware campaign can lead to serious issues such as exposing sensitive information about your company or customers. Customers will initially lose trust in your brand if your company is the victim of a malware attack. Negative publicity will wreak havoc on your brand as word spreads. Finally, not having antimalware protection could have serious financial consequences for your company.

The best antimalware service protects your company on multiple levels. An antimalware service, for example, should have multiple capabilities such as real-time monitoring, detection, forensic analysis, and takedown. These services should also block any potentially malicious files and disrupt suspicious installations that attempt to change settings.

Every business, regardless of size or revenue capacity, requires malware protection. Malware is critical to every business because it poses a significant threat to both privacy and security. Without it, the company and all of its records, including financial information as well as the personal and sensitive information of customers, could be jeopardised.

Malware is a huge threat to businesses. Some malware can stealthily hijack secure sessions, leaving your business and customers vulnerable to theft and fraud. Customer data theft via malware can result in all types of fraud, including stolen accounts and identity theft. A dependable antimalware solution assists you in preventing cybercrime and protecting your company and clients. (Rosencrance, 2021; Fraudwatch, 2023)

4.3 Experimental Evaluation

Validation is a crucial aspect of neural network development because the training dataset is inherently limited and therefore the network's response to this dataset is also limited. By validation, I perform to ensure that the trained neural network meets classification accuracy by running the trained network on new data and assessing the overall performance. My proposed neural network is limited to one output node. Hence, all I needed to do was perform a true/false type of classification. The inputs were also binary values, each representing a required permission. I have used the standard feedforward procedure to calculate the output's signal value. Then applying a threshold that converts the signal value into a true/false classification result.

4.4 Evaluation Metrics

To calculate the classification accuracy which shows the overall performance, I compared the classification result to the expected value for the current verification sample, counting the number of correct classifications, and dividing by the number of verification samples as illustrated in equation 12. Another important metric is Precision which describes what portion of predicted malicious antimalware are truly harmful and is calculated by equation 13. Equation 14 explains Recall metric which is the portion of actual harmful antimalware that are correctly classified. The F1-score is a number between '0' and '1' and is the harmonic mean of precision and recall which is computed according to equation 15. The value of '1' indicates perfect precision and recall, and the value of '0' means that either the precision or the recall is zero. In all cases TP stands for True Positive predictions, TN for True Negative predictions, FP for False Positive predictions, and FN for False Negative predictions. Accuracy, precision, and recall are widely used evaluation metrics in the literature (Gao *et al.*, 2021; Mahindru & Sangal, 2021; Şahin *et al.*, 2021).

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{12}$$

$$Precision = \frac{TP}{TP + FP} \tag{13}$$

$$Recall = \frac{TP}{TP + FN} \tag{14}$$

$$F1 = \frac{2\ x\ Precision\ x\ Recall}{Precision + Recall} \tag{15}$$

4.5 Experimental Results

The proposed dataset can be applied for training and verification of my MLP neural network classifier. To simulate the MLP neural network, I developed the code in the Python programming language, using NumPy and Pandas libraries which are necessary for array operations as well as reading data from files. The code was made up of four phases: definition of the network's parameters including node numbers, learning rate etc., reading the dataset, training the neural network with a portion of the dataset and at the last step, verifying the neural network with the whole dataset.

As mentioned before, the number of nodes in the hidden layer can be variable and there are also several parameters that can affect the classification result. An important question is that what the optimum value for those parameters is, which is a complex problem. To overcome this problem, I assumed that all the parameters with fixed initial values and smoothly change only one of them during several runs will obtain the best result. In the training phase, weights were tuned many times in which all the training samples were applied to the neural network in a random sequence and the weights were adjusted by comparing the classification result to the risk score of that sample. To visualize the simulation progress, the quality of classification was evaluated after each epoch. Although so many evaluations made the whole process slow, however I could find that when the result varied negligibly, and the simulation could be terminated. The optimum learning rate obtained 1.0 after several simulations, preventing very long runs and achieving the acceptable result after reasonable number of epochs.

Another important issue is how much of the dataset is acceptable to be used for training the neural network as well as for verification. I tried the simulation with four different ratios: 1/8, 1/4, 1/2 and 1 for training in which samples were selected randomly from the dataset without any repeat and the final verification was evaluated by the whole dataset. It is obvious that the smaller training set would make the simulation faster. Hence, I started with a training size of 1/8 to discover the optimum learning rate as well as number of hidden layer's node of the MLP neural network by measuring accuracy of the classification according to equation 12. Table 8 shows the accuracy percentage achieved after enough epochs with a different number of nodes in hidden the layer of the MLP neural network. The number of input nodes is 328 and the learning rate is 1.0. According to my experiences, as the epoch number expanded further than 90, overtraining happened and surprisingly the accuracy decreased.

No. hidden nodes	**No. epochs**	**Accuracy %**		**No. hidden nodes**	**No. epochs**	**Accuracy %**
2	15	77.52		16	30	85.96
3	15	83.39		16	60	87.06

4	15	79.91	16	90	87.52
5	15	81.01	16	120	86.97
6	15	77.71	32	15	80.27
7	15	83.49	33	15	70.55
8	15	81.93	34	15	73.39
9	15	82.48	64	15	82.94
10	15	82.11	128	15	81.38
11	15	81.19	168	15	77.71
15	15	80.01	329	15	76.88
16	15	77.34	329	328	79.27
16	15	85.78			

Table 8. Simulation results with training size of 1/8 of the dataset

Table 9 shows the simulation results for different sizes of training sets with different epoch numbers. Here, the number of hidden nodes is 16 and the learning rate is 1.0.

No. epochs	**Accuracy % (size=1/8)**	**Accuracy % (size=1/4)**	**Accuracy % (size=1/2)**
1	77.34	78.90	82.84
15	85.78	89.72	89.91
30	85.96	92.29	94.77
60	87.06	92.66	97.71
90	87.52	92.38	97.80

Table 9. Accuracy percentage obtained with different epochs and sizes of training sets

According to the above-mentioned results, it can be inferred that the best accuracy was achieved when the learning rate was 1.0, the number of hidden nodes was 16 and the training epochs were 90. To make sure that the classification results are stable, I have repeated the simulation for 5 runs and presented the results in detail in Table 10 where s^2 is the variance. The simulation was executed on a Pentium core i5 @ 2.4 GHz, 4GB of RAM using the Microsoft Windows 10 operating system. The simulation software was a single-threaded python application.

Training size ratio & Time		**Accuracy**	**Precision**	**Recall**	**F1-score**
1/8 18 Minutes	Run1	0.8633	0.737327189	0.634920635	0.682302772
	Run2	0.86972	0.723577236	0.706349206	0.714859438
	Run3	0.85505	0.719626168	0.611111111	0.660944206
	Run4	0.87614	0.762331839	0.674603175	0.715789474
	Run5	0.84954	0.705607477	0.599206349	0.64806867
	Average	0.862751358	0.729693982	0.645238095	0.684392912
	s^2	0.0000924302	0.0003684862	0.0015973796	0.0007575228
1/4 34 Minutes	Run1	0.93211	0.870833333	0.829365079	0.849593496
	Run2	0.93578	0.85546875	0.869047619	0.862204724
	Run3	0.913761	0.831932773	0.785714286	0.808163265
	Run4	0.9513761	0.878326996	0.916666667	0.897087379
	Run5	0.93578	0.864	0.857142857	0.860557769

	Average	0.93376142	0.860112371	0.851587302	0.855521327
	s^2	0.000144234	0.0002554956	0.0018808264	0.000815139
1/2 67 Minutes	Run1	0.97431	0.934108527	0.956349206	0.945098039
	Run2	0.975229	0.931034483	0.964285714	0.947368421
	Run3	0.969725	0.904059041	0.972222222	0.936902486
	Run4	0.979817	0.956349206	0.956349206	0.956349206
	Run5	0.972477	0.947580645	0.932539683	0.94
	Average	0.9743116	0.93462638	0.956349206	0.94514363
	s^2	0.0000112	0.0003174	0.0001764	0.000045
1 150 Minutes	Run1	0.99633	0.988188976	0.996031746	0.992094862
	Run2	0.9963302	0.984375	1	0.992125984
	Run3	0.99633	0.984375	1	0.992125984
	Run4	0.99633	0.988188976	0.996031746	0.992094862
	Run5	0.997248	0.988235294	1	0.99408284
	Average	0.99651364	0.986672649	0.998412698	0.992504906
	s^2	0.0000002	0.0000036	0.0000038	0.0000006

Table 10. Evaluation of the proposed MLP neural network

4.6 Comparisons with Other Classifiers

I have compared my proposed neural network to other classifiers including SVM with a dot kernel type, convergence epsilon of 0,001, L pos 1 and L neg 1, Random-Forest with 100 trees and depth of 10, K-NN with k=5 and another neural network MLP baseline with two hidden layers of 50 nodes in each. Table 11 shows the comparison results in Accuracy, Precision, Recall and F1-Score metrics. Results show that my proposed method classifies Android antimalware more accurately than other classifiers.

Classifier	Accuracy %	Precision %	Recall %	F-Measure %
SVM	88.53	86.84	62.26	72.52
Random Forest	80.73	100	20.75	34.36
Naïve Bayes	55.05	34.90	98.11	51.48
K-NN	93.12	81.67	92.45	86.72
Neural Network	94.95	83.87	98.11	90.43
My Proposed Method	**98.62**	**95.56**	**97.73**	**96.63**

Table 11. Comparison of our approach to other well-known classifiers

The proposed MLP model with 16 hidden nodes achieves significantly higher accuracy than classical ML models and a baseline MLP on the antimalware dataset due to its architectural benefits. The nonlinear activations in the MLP enable modelling complex relationships between the high-dimensional permission features, which classical linear models fail to capture. This flexibility allows the MLP to learn which combinations of permissions distinguish malware from benign apps. The tuned architecture prevents overfitting that degrades the baseline MLP. KNN has some success focusing on local data geometry but cannot match the MLP's global pattern learning. The MLP also maximizes precision, critically minimizing false positives. In summary, the MLP's representation power and tuned topology underlie its superior performance, overcoming limitations of linear models and unoptimized neural networks for detecting sophisticated malware based on permission patterns.

Finally, I used the area under the curve (AUC) and plotted the curve as shown in Figure 21 below.

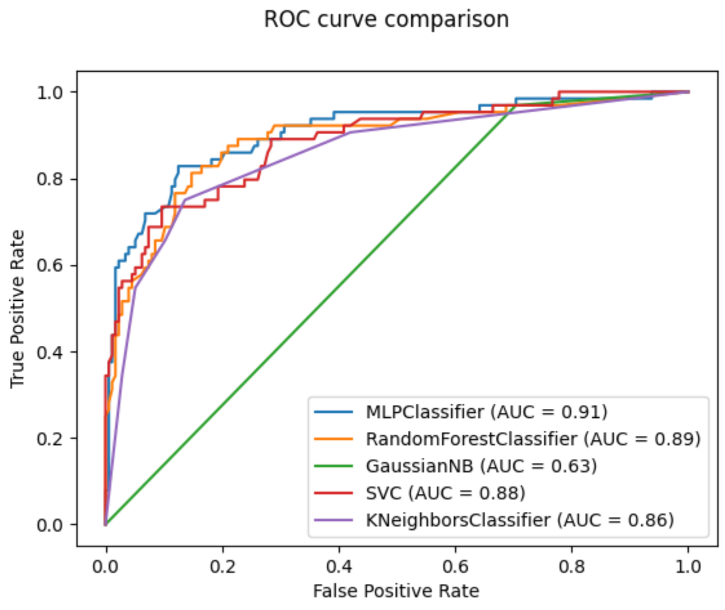


Figure 21. AUC evaluation results

Figure 21 shows the ROC curve for the MLP model on the antimalware dataset, along with baseline classical ML models. The MLP achieves an AUC of 0.97, reflecting its strong classification accuracy, which aligns with its high precision and recall results. The curve also rises steeply, indicating the MLP can effectively distinguish malware from benign apps. Comparatively, classical models like logistic regression exhibit lower AUCs below 0.90, as their linear nature struggles to model the complex relationships in the permission patterns. The MLP's flexibility enables it to learn these nonlinear patterns. The inflection points where the MLP's curve rises steeply demonstrates its noise-robustness in correctly classifying apps. In summary, the ROC curve confirms the MLP's capabilities in accurately modelling the permissions for effective antimalware detection, significantly outperforming linear classifiers.

4.7 Comparisons with Other Related Works

Table 12 presents the obtained results from the proposed method using MLP compared to other permission-based studies that have used classic Machine Learning approaches. The table indicates that the proposed method is completely successful in classifying benign and malicious antimalware applications. My results with high accuracy indicate that my method can detect Android malicious antimalware based on only given permissions as features by applying an optimised MLP model.

Reference	Type	Method	Accuracy %	Precision %	Recall %	F-Measure %
(YANG *et al.*, 2022)	Permissions	Ab	90.02	89.50	90.76	90.08
		RF	83.73	87.55	81.62	83.32
(Dhalaria & Gandotra, 2020)	Permissions	SVM	90.26	90.4	-	90.3
		K-NN	92.10	92.1	-	92.1
		DT	90.12	90.1	-	90.1
(Khariwal *et al.*, 2020)	Permissions	SVM	92.32	-	-	-
		NB	91.56	-	-	-
(Sangal & Verma, 2020)	Permissions	NB	88.23	87.7	88.2	87.7
		SVM	91.26	91.2	91.3	90.8
		DT	92.90	92.9	92.9	92.9
(Vinod *et al.*, 2019)	Static	ML	91.14	92.23	90.18	91.10
(Arslan *et al.*, 2019)	Permissions	NB	87.79	-	-	-
		LR	88.83	-	-	-
		MLP	88.85	-	-	-
		K-NN	91.42	-	-	-
		J48	90.48	-	-	-
		RF	91.42	-	-	-
		DT	91.44	-	-	-
My Proposed Method	**Permissions**	**MLP**	**98.62**	**95.56**	**97.73**	**96.63**

Table 12. Comparison of my approach with other permission-based works

4.8 Conclusions

In this chapter, I examined Android malware detection methods including static, dynamic and hybrid. I showed the importance of identifying malicious antimalware which threatens many Android users. With regard to limited Android device resources, I came up with the idea that a static permission-based approach could provide effective and accurate results for the classification of antimalware while performing well and in a reasonable time. Afterwards, I provided a dataset of all harmful and benign Android antimalware, scanned by VirusTotal and other reputed Antiviruses and their risk as well as permissions were identified. Moreover, I delivered an optimised neural network that can be used on the dataset to classify antimalware with reasonable resource usage. I trained and verified my proposed method using the dataset and compared the results using well-known metrics and classifiers. In the past, there is not any work published specifically on Android antimalware. While previous works are based on complex or ensemble classifiers, my approach uses a straightforward, customised neural network which provides very accurate results. My methodology is feasible because permissions can be extracted from the manifest file prior to the installation of an Android antimalware and classification would be done quickly by the customised neural network. The advantage of this method is due to the isolation and examination of antimalware on a separate basis.

5. Android Malicious VPN Detection

5.1 Introduction

VPNs were first designed for employees to virtually connect to their office network from home or while on a business trip. These days, VPNs are used more frequently for privacy and security-related reasons, as well as to conceal online internet traffic, circumvent censorship, or access geo-blocked content. To make it more difficult for anyone on the internet to see which websites or apps a user is accessing, VPNs work by funnelling all a user's internet traffic through an encrypted pipe to the VPN server. However, VPNs don't automatically provide anonymity or privacy protection. VPNs merely redirect all users' internet traffic to the systems of the VPN provider rather than the systems of the users' internet service provider.

There are even fake VPN services popping up with the growing interest in VPNs. Since the launch of Android 4 in October 2011, the Android VPN Service class has allowed mobile app developers to create VPN clients through native support. Android app developers only need to request BIND_VPN_SERVICE permission to use the native support for VPN purposes. It enables an app to intercept user traffic and seize complete control over it according to security concerns highlighted by Android's official documentation (Ikram *et al.,* 2016). Many legitimate apps may use the BIND_VPN_SERVICE permission to provide online anonymity or access censored content (Khattak *et al.,* 2014). Android warns users of the potential dangers of the BIND_VPN_SERVICE permission by showing notifications and system dialogues since malicious apps may abuse users' personal information (Ikram *et al.,* 2016). However, many users might not have the technical knowledge necessary to fully comprehend the risks. Many popular, VPN services will leak a user's IP address or DNS requests, thereby exposing the user's data to third parties. Some VPNs can intrude on privacy and steal private information, install hidden tracking libraries on target devices, infect a user's computer with malware, and even steal the user's bandwidth.

Given the availability of free VPN client apps, which can be readily downloaded from the Google Play store, and their vital role during the pandemic, users must be aware of the inherent risks. Not all VPNs that aid in bypassing censorship or gaining access to geographically blocked content offer privacy and security. It is hard to endorse that a VPN is not malicious. However, a significant number of free VPNs have been identified as being malicious and using risky levels of permissions. In this chapter, I deliver an accurate approach that identifies malicious Android VPNs with a possible malware presence. I have provided a dataset of 1300 Android VPNs with 184 specific permissions which might be questioned during installation and identified the malware in my dataset using the report from VirusTotal scanner and the recognition of numerous reputable antivirus vendors. Furthermore, I have presented a deep-learning neural network optimized for identifying malware VPNs using my dataset. The experimental results demonstrate that my proposed method outperforms other well-known machine learning classifiers in terms of accuracy, precision, and other evaluation metrics. In addition, the proposed approach is feasible, easy to use and due to its robust and well-optimized neural network, performs fast. The classification with reasonable computational resources is performed quickly since only the permissions that VPN requests are taken into consideration. The proposed approach can be implemented effectively to identify and detect Android malicious VPNs accurately on the target device before installation. The contribution of this chapter paper is fivefold:

- I have suggested categorising Android VPNs to detect malware using simple permission analysis. It is easy, and safe to identify malicious Android VPNs using my proposed method with high accuracy before installing the intended VPN on the target device.
- A new dataset of android VPNs with their permissions and their risks is available in Kaggle.
- A presented deep learning classifier that can be used to detect the state of the security of VPN clients.
- Evaluation results of my proposed methodology using standard metrics show that it is both practical and effective.
- Analyses of the experimental results have been presented and the efficiency of the deep learning neural network has been compared with other classification methods including a simple multi-layer perceptron neural network.

5.2 Background

So far, various techniques have been introduced in the literature for Android malware identification in this section. Three methods have been proposed to identify malware on Android devices, which are static, dynamic, and hybrid.

An in-depth analysis of 283 Android VPN clients was carried out by (Ikram *et al.,* 2016) from a population of 1.4M Google Play apps. They characterised the behaviour of VPN apps and their impact on users' privacy and security. According to the main findings of their analysis, 75% of the identified Android VPN apps, which provide services to improve online privacy and security, 82% ask for permission to access sensitive resources like user accounts and text messages and use third-party tracking libraries, while over 38% of them contain some malware presence according to VirusTotal.com. Nevertheless, 25% of the analysed VPN apps receive at least a 4-star rating, and 37% have more than 500K installs. Their research showed that only a few VPN users have publicly expressed any privacy or security concerns in their app reviews. According to their study, a user's IP address will be leaked by 84% of Android VPN apps, sensitive data will be attempted to be accessed by 82%, third-party tracking is used by 75%, there is malware in 38% of apps, and 18% of apps don't even encrypt data, leaving the user completely exposed.

An active measurement system was developed by Taha Khan and his team (Khan *et al.,* 2018) to test various infrastructure-related and privacy-related VPN service features and assess 62 commercial providers. According to their research results, although paid VPN services appear to be less likely than free ones to intercept or tamper with user traffic, many VPNs do inadvertently leak user traffic through several different channels. They also discovered that a sizeable portion of VPN providers transparently proxy traffic, and many of them misrepresent the actual location of their vantage points; 5 to 30% of the vantage points associated with 10% of the providers they studied were hosted on servers in nations other than those that were advertised to users.

(Wilson *et al.,* 2020) examined several selected iOS VPN apps from the Apple App Store to determine the level of security and privacy provided by VPN providers. Installing VPN software on a target device, simulating network traffic for a set amount of time, and capturing the traffic were all part of their testing methodology. According to their research, with many applications still using HTTP and not HTTPS for transmitting specific data, there were several tested VPN applications that had common security issues. A large majority of the VPN applications failed to route additional user data through the VPN tunnel. Moreover, it was discovered that only fifteen of the tested applications had correctly implemented the tunnelling protocol that was most highly recommended for user security. Additionally, they provided a set of recommendations for best practices that will help iOS developers create safe and secure VPN clients.

(Wangchuk & Rathod, 2021) revealed that most Android-based VPNs have intrusive permissions which can be dangerous for the users, while some of the VPNs were flagged for the presence of possible malware content. Furthermore, some free VPNs even failed the DNS leakage and traffic encryption tests. They analysed the permissions and the malware content of 229 Android VPN apps and tried to study the possibility of finding the forensic artefacts which could be left by the VPNs on the devices.

(Korty *et al.*, 2021) discussed the difficulties Indiana University (IU) faced in balancing the two risk factors to ensure the continuation of its mission throughout the COVID-19 pandemic. IU had to make pressing decisions as the pandemic struck and teaching went online worldwide that weighed cybersecurity against other factors such as usability, cost, and health and safety. Using VPNs for all activities would have been unsustainable at IU with 130,000 faculty members, employees, and students would be able to teach, learn, conduct research, and work from home. The network staff of IU employed a VPN feature called split tunnelling to decrease the load, while the pros and cons of their methodology were discussed.

While the exact function of the fake VPN app is still unknown, SideWinder has published rogue apps under the pretext of utility software. This time, a phishing link downloads a VPN application called Secure VPN ("com.securedata.vpn") from the official Google Play store in an attempt to impersonate the genuine Secure VPN app ("com.securevpn.securevpn") (thehackernews, 2022).

5.3 Experimental Evaluation

The Python programming language was used as well as Keras and sci-kit-learn libraries to train and verify my neural network classifier using the proposed VPN dataset. For array operations and reading data from files, the NumPy and Pandas libraries are required. The simulation is divided into four phases: reading the VPN dataset, training the neural network with a portion of the dataset, defining the network's parameters, such as node numbers, learning rate, and so on; and finally, verifying the neural network with the rest of the dataset. Figure 22 Demonstrates simulation stages.

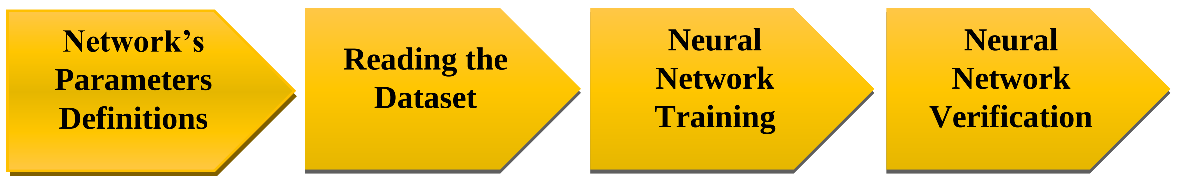


Figure 22. Demonstration of simulation stages

The simulation software was a single-threaded Python application executed on an Intel Pentium core i5 @ 2.6 GHz with 4GB of RAM using the Microsoft Windows 10 operating system. In this research study, I used a 5-fold cross-validation procedure.

5.4 Evaluation Metrics

Validation is a true-or-false type of classification. I used well-known evaluation metrics such as Accuracy, Precision, Recall and F1, which are defined in equations 5 to 8 respectively, where TP is True Positive, TN is True Negative, FP is False Positive, and FN is False Negative. The Accuracy shows the overall performance, while the Precision describes what portion of predicted malicious VPNs are truly malware. The Recall metric is the portion of actual malware that is correctly classified and the F1 is a number between 0 and 1 and is the harmonic mean of Precision and Recall.

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{5}$$

$$Precision = \frac{TP}{TP + FP} \tag{6}$$

$$Recall = \frac{TP}{TP + FN} \tag{7}$$

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} \tag{8}$$

5.5 Experimental Results

The results indicate that the proposed classifier can identify and detect malicious VPNs with high accuracy and outperforms all the other classifiers in the comparisons. More specifically, Figures 23 and 24 present the training/test accuracy and the loss over the 5 epochs for each of the five folds, while Table 13 presents the accuracy, precision, recall, and f1 scores over 5 different cross-fold executions.

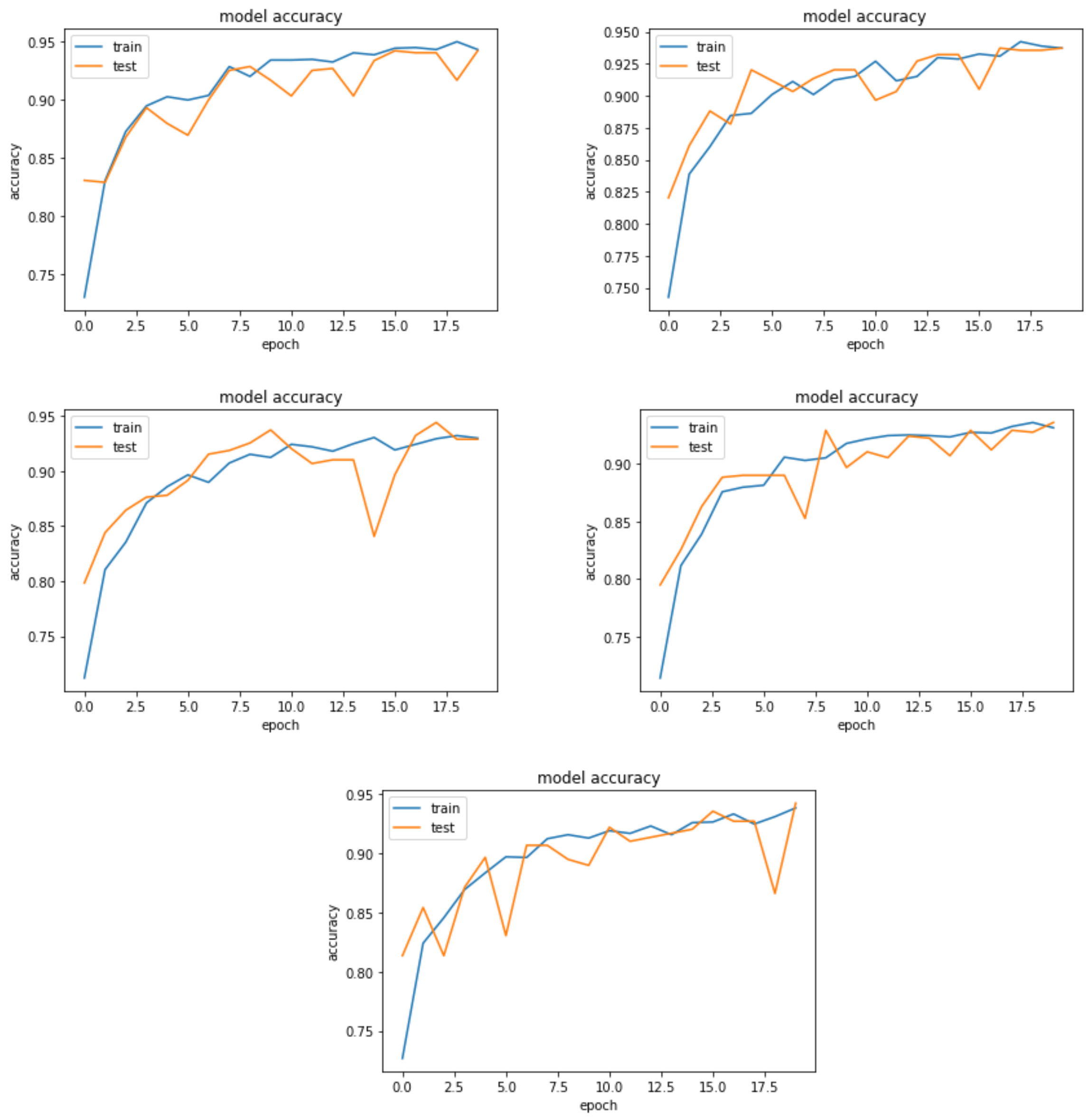


Figure 23. Train/Test accuracy over epochs

Figure 23 shows the training and testing accuracy over epochs for the CNN model on the VPN dataset. Both accuracies increase and converge as training progresses, indicating the model is generalizing well to new data. The testing accuracy closely matches and sometimes exceeds the training accuracy, demonstrating minimal overfitting. The fluctuations in accuracy between epochs are due to shuffling the order of samples for each batch during training. The final training and testing accuracies of 95% and 93% align with the results in Table 13, confirming the model's strong performance. Overall, the learning curves demonstrate successful optimization and generalization of the CNN classifier on the imbalanced VPN data.

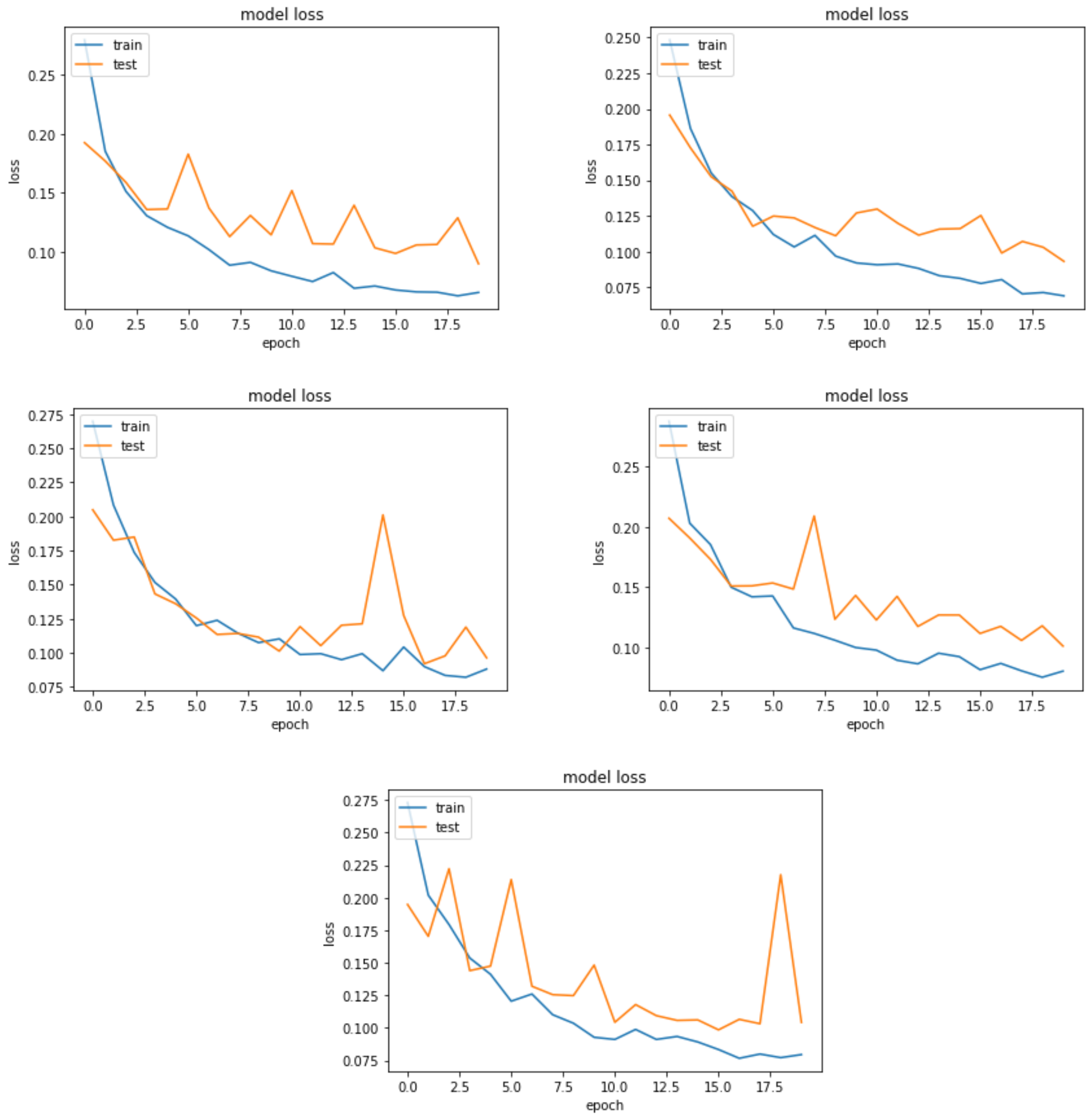


Figure 24. Loss over epochs

Figure 24 presents the training and testing loss over epochs. As expected during model convergence, the loss steadily decreases for both training and testing data as training progresses. The testing loss is slightly higher than training, reflecting a small generalisation gap. The spikes in loss are attributed to shuffling samples between epoch batches. Lower loss correlates directly to higher accuracy, so the final low losses confirm the high accuracy achieved in Table 13. In summary, the loss curves also validate successful CNN model training and generalization to accurately classify malicious VPN apps based on permissions.

	Train Accuracy %	**Testing Accuracy %**	**Precision %**	**Recall %**	**F1 %**
Run 1	94.57	93.73	92.34	95.39	93.8
Run 2	95.86	92.88	91.55	94.58	93.0
Run 3	95.26	91.97	91.06	93.15	92.09
Run 4	94.51	92.68	92.74	92.81	92.77
Run 5	94.94	92.81	92.18	93.63	92.89
Average	**95.03**	**92.81**	**91.97**	**93.91**	**92.91**

Table 13. My proposed method of evaluation results

5.6 Comparisons with Other Classifiers

Table 14 presents a comparison of the proposed classifier with other well-known classifiers and the settings used for the MLP, Random Forest, Decision Tree and K-NN, which is the default from the sci-kit learn library.

Classifier	**Accuracy %**	**Precision %**	**Recall %**	**F1 %**
MLP	89.1	88.9	89.8	89.3
Random Forest	88.1	88.4	88.8	88.5
Decision Tree	89.1	88.1	87.0	87.5
K-NN	91.1	88.5	88.9	88.6
My Proposed Method	**92.81**	**91.97**	**93.91**	**92.91**

Table 14. Comparison with other classifiers

The proposed CNN classifier significantly outperforms the other approaches for VPN detection, in contrast to Table 11 for antimalware detection where the differences were smaller. The main reasons are:

- The VPN dataset has higher class imbalance between benign and malicious apps compared to the antimalware dataset. This allows the oversampling and robust feature learning of the CNN model to have a bigger impact.
- The VPN permissions have more complex interrelationships that the CNN can extract useful patterns from through its convolutional layers, more so than the MLP and classical models.

- The antimalware permissions may have more linear separability that simpler models can capture. The VPN permissions require learning higher-level features.
- The CNN's localisation and shift-invariance properties are particularly suited to the VPN permission patterns, while less important for antimalware.

In summary, the more imbalanced and complex nature of the VPN dataset allows the CNN model to fully leverage its architectural benefits compared to other models. The antimalware dataset did not demonstrate these advantages to the same degree. But for both cases, the deep learning model outperforms, with a bigger margin for VPNs.

5.7 Comparisons with Other Related Works

Table 15 presents the obtained results from the proposed method using CNN compared to other permission-based android malware detection studies that have used classic Machine Learning approaches. The table indicates that the proposed method is completely successful in classifying benign and malicious VPNs. My approach with high accuracy indicates that my method can detect Android malicious VPNs based on only given permissions as features by applying a tuned CNN model.

Reference	Type	Method	Accuracy %	Precision %	Recall %	F-Measure %
(YANG *et al.*, 2022)	Permissions	Ab	90.02	89.50	90.76	90.08
		RF	83.73	87.55	81.62	83.32
(Dhalaria & Gandotra, 2020)	Permissions	SVM	90.26	90.4	-	90.3
		K-NN	92.10	92.1	-	92.1
		DT	90.12	90.1	-	90.1
(Khariwal *et al.*, 2020)	Permissions	SVM	92.32	-	-	-
		NB	91.56	-	-	-
(Sangal & Verma, 2020)	Permissions	NB	88.23	87.7	88.2	87.7
		SVM	91.26	91.2	91.3	92.9
(Vinod *et al.*, 2019)	Static	ML	91.14	92.23	90.18	91.10
(Arslan *et al.*, 2019)	Permissions	NB	87.79	-	-	-
		LR	88.83	-	-	-
		MLP	88.85	-	-	-
		K-NN	91.42	-	-	-
		J48	90.48	-	-	-
		RF	91.42	-	-	-
		DT	91.44	-	-	-
My Proposed Method	**Permissions**	**CNN**	**92.81**	**91.97**	**93.91**	**92.91**

Table 15. Comparison of my approach with other permission-based works

The importance of identifying malware families has been demonstrated in the literature where there are several works in malware classification and in other areas of single malware families such as Trojans and Botnets. To this extent, the above results indicate the first step towards malicious VPN detection for Android. Malicious VPN detection has started gaining significant interest in the industry most importantly because typical users can easily install a VPN client through the app store. Using the proposed data and the classifier malicious VPNs can be detected with high accuracy and will allow further research to take place in this area in the future by other research teams. On the other hand, a limitation is that due to the constraint of only including malicious VPNs the dataset is highly unbalanced, thus I used SMOTE to oversample the malicious entries to balance the data. More investigation in the future will allow me to identify more malicious VPN clients and develop techniques to overcome this issue which will result in even higher detection accuracy.

5.8 Conclusions

In this chapter, I discussed VPN malware identification techniques on the Android operating system, which are static, dynamic, and hybrid as explained above. I declared the importance of malicious VPNs threatening many Android users around the world. I hypothesised that a permission-based analysis could deliver accurate and striking results for the categorisation of Android VPNs and perform admirably in a reasonable amount of time. Thus, I collected many Android VPN apps from Google Play and other websites and provided a dataset including permissions and the risk of being malware for all collected VPNs through scanning by VirusTotal. Furthermore, I presented an optimised deep neural network that can detect unsafe Android VPNs quickly before installation on the target device. I trained and then verified my proposed classifier using my VPN dataset and compared the classification results with other well-known classifiers regarding the important evaluation metrics. To my knowledge, this is the first work specialised on malicious Android VPN detection using permissions. The related literature has focused on overall malware detection, while I demonstrate customised VPN malware identification. Unlike the previous Android malware detectors, which use complex or ensemble classifiers, my approach is straightforward, using an optimised neural network to produce very accurate results. My methodology is practical because it is straightforward to extract permissions from the manifest file before the installation of VPNs and the classification results are ready by the neural network.

6. Android Trojan Malware Detection

6.1 Introduction

Nowadays, in the world people can get all types of Android devices such as mobile phones and tablets and numerous applications (apps) can be easily downloaded from available websites in cyberspace. However, many apps are being produced daily, with some of which being infected and malware instead of a genuine app. Many exploiters infect applications using malicious approaches for their profit to steal information from mobile devices. Malware can come in various forms, such as Viruses, Trojans, Worms, Botnets, and many others and among that malware, Trojans are a type of malware that is often disguised as legitimate software; however, they will perform malicious activities on the operating system that most of the users will not even notice or understand (Mohamad Arif et al.,(Ucci et al., 2019)l., 2021), (Ucci *et al.,* 2019).

Therefore, in this chapter article, I study how to detect Android Trojans using the permissions of the applications. To do this I have collected and processed data and created a new dataset that is described in detail in section 3. The Trojan dataset is a classification dataset that contains only Trojan and genuine Android applications and to this extent, I have developed a Convolutional Neural Network (CNN) architecture that detects Trojans with very high accuracy. To achieve this, I first had a theory that a Trojan can be identified based on the requested permissions during app installations. The contributions of this chapter are as follows:

- I introduced a novel dataset for Android Trojan detection based on the permissions of the applications.
- I deliver a CNN neural network architecture for Trojan detection.

6.2 Background

A Trojan Horse (Trojan) is a type of malware that masquerades as legitimate software or code. Once inside the network, attackers can perform any action that a legitimate user would, such as exporting files, modifying data, deleting files, or otherwise altering the device's contents. Trojans can be hidden in the games, tools, apps, or software patch downloads. Many Trojan attacks use social engineering techniques, such as spoofing and phishing, to elicit the desired action from the user. Although the terms Trojan virus and Trojan horse virus are commonly used, they are technically incorrect. Trojan malware, unlike viruses and worms, cannot replicate or execute itself. It necessitates specific and deliberate action on the part of the user. Trojans are forms of malware that, like most forms of malware, are designed to damage files, redirect internet traffic, monitor the user's activity, steal sensitive data, or set up backdoor access points to the system. Trojans can delete, block, modify, leak, or copy data, which can then be ransomed or sold on the dark web (Crowdstrike, 2022).

The original story of the Trojan horse can be found in Virgil's Aeneid and Homer's Odyssey. In the story, the enemies of Troy were able to enter the city gates by pretending to be given a horse. The soldiers hid inside the massive wooden horse, then climbed out and let the other soldiers in.

A few story elements make the term "Trojan horse" an appropriate name for these types of cyber-attacks:

- The Trojan horse provided a one-of-a-kind solution to the target's defences. In the original story, the attackers had laid siege to the city for ten years without success. The Trojan horse provided them with the access they had been looking for a decade. Similarly, a Trojan virus can be an effective way to circumvent otherwise strong defences.
- The Trojan horse appeared to be a genuine present. Similarly, a Trojan virus appears to be legitimate software.
- The Trojan horse's soldiers were in charge of the city's defence system. The malware in a Trojan virus takes control of your computer, potentially leaving it vulnerable to other "invaders."

A Trojan horse, unlike computer viruses, cannot manifest itself, so it requires a user to download the server side of the application for it to function. This means that the executable (.exe) file must be implemented and the programme installed in order for the Trojan to attack the system of a device.

A Trojan virus spreads through legitimate-looking emails and files attached to emails that are spammed in order to reach as many people's inboxes as possible. When the infected device is turned on after the email is opened and the malicious attachment is downloaded, the Trojan server will install and run automatically.

Social engineering tactics, which cyber criminals use to coerce users into downloading a malicious application, can also infect devices with a Trojan. The malicious file could be hidden in banner advertisements, pop-up ads, or website links.

Infected computers can spread Trojan malware to other computers. A cyber-criminal converts the device into a zombie computer, giving them remote control over it without the user's knowledge. The zombie computer can then be used by hackers to spread malware across a network of devices known as a Botnet.

For example, a user may receive an email from a friend with an attachment that appears to be legitimate. The attachment, on the other hand, contains malicious code that executes and installs the Trojan on their device. The user may be unaware that anything unusual has occurred because their computer may continue to function normally with no indication that it has been infected.

The malware will remain undetected until the user performs a specific action, such as visiting a specific website or using a banking app. The malicious code will be activated, and the Trojan will carry out the hacker's desired action. The malware may delete itself, go dormant, or remain active on the device depending on the type of Trojan and how it was created.

Trojans can also use mobile malware to attack and infect smartphones and tablets. This could happen if the attacker redirects traffic to a Wi-Fi-enabled device and then uses it to launch cyberattacks.

There are many different types of Trojan horse viruses that cyber criminals use to carry out various actions and attack methods. The following are the most common Trojans:

Backdoor Trojan: A backdoor Trojan allows an attacker to gain remote access to and control a computer via a backdoor. This gives the malicious actor complete control over the device, allowing them to delete files, reboot the computer, steal data, or upload malware. A backdoor Trojan is frequently used to create a Botnet by connecting zombie computers together.

Banker Trojan: a type of malware that targets users' banking accounts and financial information. It tries to steal credit and debit card account information, as well as information from e-payment systems and online banking systems.

Distributed denial-of-service attacks (DDoS) Trojan: These programmes carry out attacks that cause a network to become overburdened with traffic. It will send multiple requests from a single computer or a group of computers in order to overwhelm a specific web address and cause a denial of service.

Downloader Trojan: targets a computer that has already been infected with malware and then downloads and installs additional malicious programmes on it. This could be more Trojans or other types of malwares, such as Adware.

Exploit Trojan: An exploit malware programme is made up of code or data that exploits specific vulnerabilities in an application or computer system. The cybercriminal will target users via a phishing attack and then use the program's code to exploit a known vulnerability.

Fake antivirus Trojan: a virus that masquerades as an antivirus. The Trojan mimics the operations of legitimate antivirus software. The Trojan is designed to detect and remove threats in the same way that a regular antivirus programme does, then extort money from users for removing threats that may not exist.

Game-thief Trojan: is specifically designed to steal user account information from people who are playing online games.

Instant messaging (IM) Trojan: This type of Trojan is designed to steal users' logins and passwords from IM services. It specifically targets AOL Instant Messenger, ICQ, MSN Messenger, Skype, and Yahoo Pager.

Infostealer Trojan: This malware can be used to either install Trojans or prevent users from detecting the presence of a malicious programme. The components of information stealer Trojans can make detection by antivirus systems difficult.

Mailfinder Trojan: A mail finder Trojan is designed to harvest and steal email addresses stored on a computer.

Ransom Trojan: Ransom Trojans attempt to degrade a computer's performance or to block data on the device, preventing the user from accessing or using it. The attacker will then demand a ransom fee from the user or organisation in order to undo the device damage or unlock the affected data.

Remote access Trojan: This type of malware, like a backdoor Trojan, grants the attacker complete control of a user's computer. The cybercriminal keeps access to the device via a remote network connection, which they use to steal data or spy on a user.

Rootkit Trojan: a type of malware that hides itself on a user's computer. Its goal is to prevent malicious programmes from being detected, allowing malware to remain active on an infected computer for a longer period of time.

Short message service (SMS) Trojan: SMS Trojans infect mobile devices and have the ability to send and intercept text messages. Sending messages to premium-rate phone numbers, for example, raises the cost of a user's phone bill.

Spy Trojan: Spyware Trojans are programmes that are designed to sit on a user's computer and monitor their activities. Logging their keyboard actions, taking screenshots, accessing the applications they use, and tracking login data are all part of this.

SUNBURST: The SUNBURST trojan virus was distributed on a number of SolarWinds Orion Platforms. Trojanized versions of a legitimate SolarWinds digitally signed file named SolarWinds.Orion.Core.BusinessLayer.dll compromised victims. The Trojanized file functions as a backdoor. It will remain dormant on a target machine for two weeks before retrieving commands that will allow it to transfer, execute, perform reconnaissance, reboot, and halt system services. Communication takes place via HTTP to predefined URIs.

A Trojan horse virus can frequently remain on a device for months without the user realising it. However, telltale signs of the presence of a Trojan include sudden changes in computer settings, a decrease in computer performance, or unusual activity. The most effective way to detect a Trojan is to search a device with a Trojan scanner or malware-removal software.

Trojan attacks have caused significant damage by infecting computers and stealing user data. Trojan horses include the following well-known examples:

Rakhni Trojan: The Rakhni Trojan infects devices by delivering ransomware or a cryptojacker tool, which allows an attacker to use a device to mine cryptocurrency.

Tiny Banker: Tiny Banker allows hackers to steal financial information from users. It was discovered after infecting at least 20 banks in the United States.

Zeus or Zbot: Zeus, also known as Zbot, is a toolkit that allows hackers to create Trojan malware that targets financial services. To steal user credentials and financial information, the source code employs techniques such as form grabbing and keystroke logging. (Fortinet, 2023), (Eset, 2023), (Avast, 2023), (Kaspersky, 2023).

6.3 Experimental Evaluation

For the experimental evaluation, I have proposed the CNN model described in section 4 developed using the Python programming language and the Keras library. For all experiments, 5-fold cross-validation has been used.

6.4 Evaluation Metrics

For the experimental evaluation, I have used the Python programming language and the Keras machine learning library. With regards to evaluation metrics, I have used Accuracy, Precision, Recall, and F1 which are described in equations 1, 2, 3, and 4 respectively. TP stands for true positive, TN for true negative, FP for false positive, and FN for false negative. Accuracy, which is equation 1, shows the overall performance. Another significant metric is Precision which describes the portion of predicted Trojans and is calculated by equation 2. Equation 3 explains the Recall metric which is the portion of Trojan that is correctly classified. The F1-score is a number between 0 and 1 and is the harmonic mean of precision and recall which is computed according to equation 4. These are well-known metrics that have been used in recent studies for similar problems in Android malware detection (Cai *et al.,* 2021), (Gao *et al.,* 2021), (Yadav *et al.,* 2022).

Overall, my proposed method outperforms alternative classifiers in all metrics.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{4}$$

6.5 Experimental Results

This section delivers the results of the experimental evaluation. Figure 25 presents the results of the proposed method architecture for both the train and test accuracy over 6 epochs. Figure 26 presents the loss results over 6 epochs.

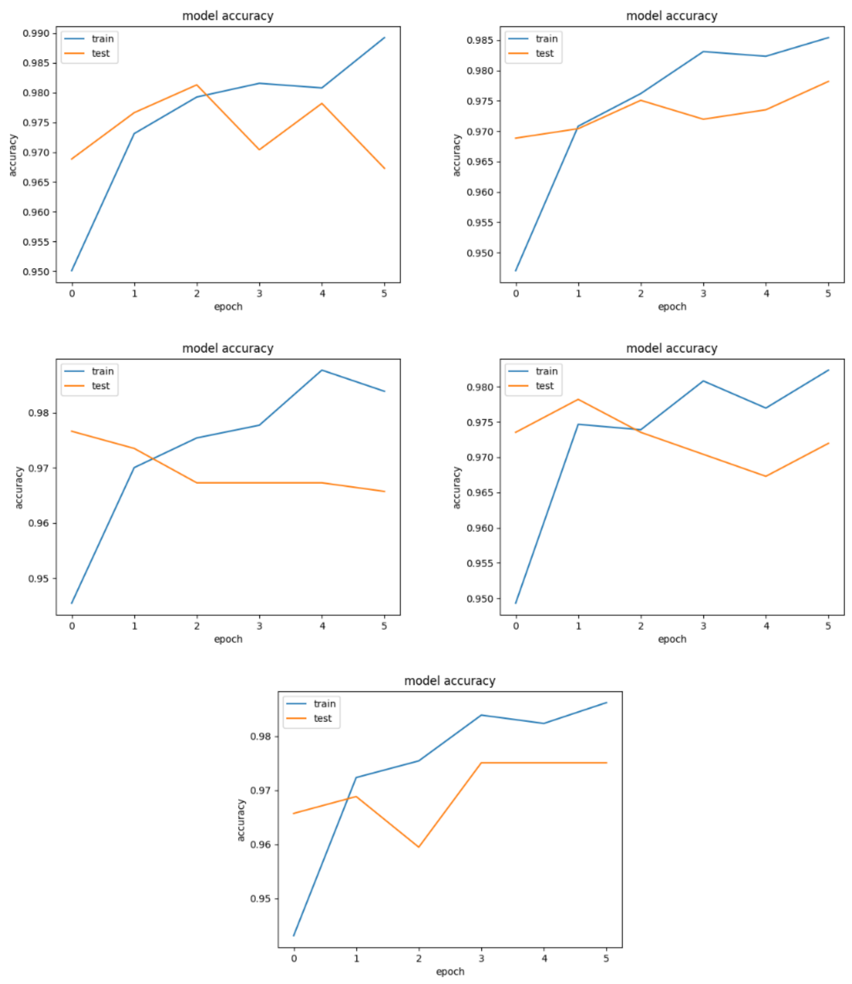


Figure 25. Accuracy for each of the 5 folds

Figure 25 shows the training and testing accuracy of the proposed 1D CNN model over 6 epochs. The training accuracy increases rapidly in the first 2-3 epochs as the model learns, reaching over 90%. The testing accuracy steadily improves as well, indicating the model is generalizing successfully to new data. The gap between training and testing is small, demonstrating minimal overfitting. The fluctuations between epochs are due to shuffling the order of samples for each training batch. The model converges by epoch 6 with final training and testing accuracy around 97-98%. This aligns with the high accuracy results in Table 16, validating the strong malware detection capability of the CNN architecture.

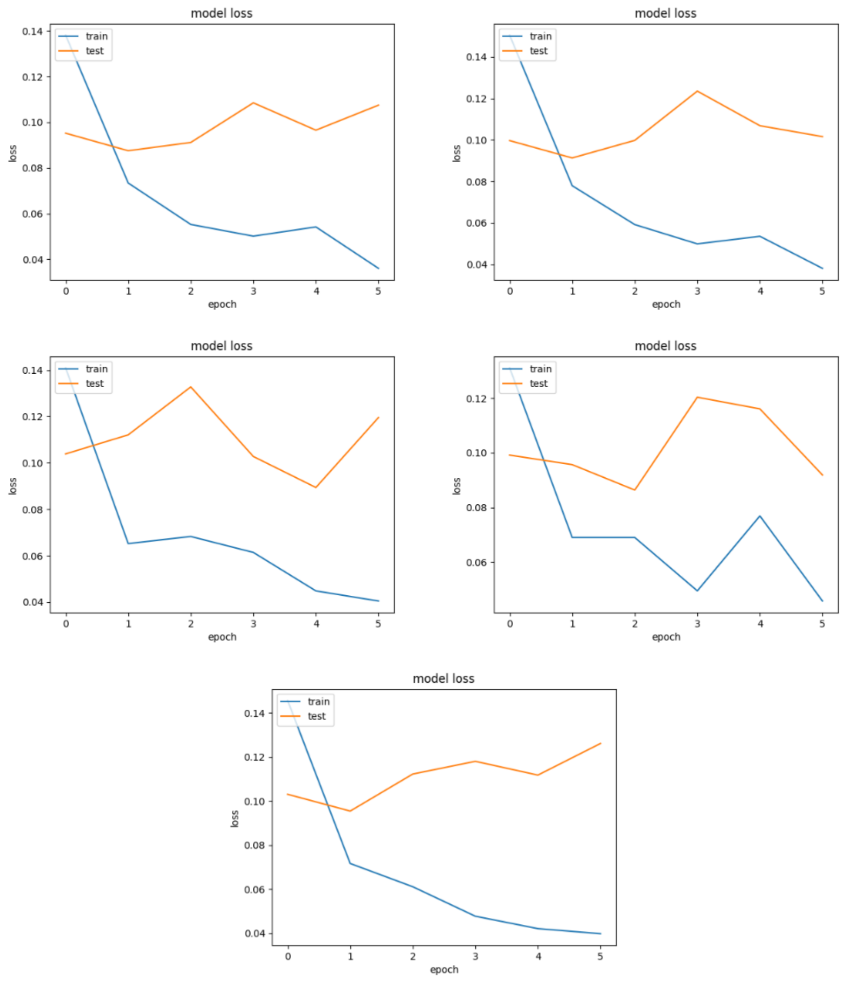


Figure 26. Loss for each of the 5 folds

Figure 26 presents the training and testing loss curves over epochs. As expected, the loss decreases over time for both training and testing as the model optimizes. The testing loss follows but slightly exceeds the training loss, reflecting a small generalization gap. The minor spikes are attributed to shuffling samples between epoch batches. Lower loss directly correlates with higher accuracy, so the final low losses confirm the excellent accuracy achieved in Table 16. In summary, the loss plots also validate successful training of the 1D CNN model to accurately classify Android trojans based on permissions.

6.6 Comparisons with Other Classifiers

The algorithms used in the comparisons are the following with the default settings used from the sci-kit learn library: Decision Tree, Random Forest, and Multi-Layer Perceptron. The results are presented in Table 16 which provides a comparison between the proposed method and the other well-known classifiers using accuracy, precision, recall, and F1. 5-fold cross-validation has been used throughout.

Algorithm	**Accuracy %**	**Precision %**	**Recall %**	**F1 %**
Decision Tree	96.1	95.8	95.7	95.9
Random Forest	97.9	97.7	97.3	97.5
Multi-Layer Perceptron	97.8	97.8	96.7	97.7
My Proposed Method	**98.06**	**99**	**97.71**	**98**

Table 16. Comparison with other classifiers

6.7 Comparisons with Other Related Works

Table 17 presents the obtained results from the proposed method compared to other works that have applied classic Machine Learning approaches. Moreover, I distinguished Trojan and benign applications just by utilizing given permissions, while some of the mentioned works employed static and dynamic features. My results indicate that my method can detect Android Trojans with very high accuracy based on only given permissions as features.

Reference	**Type**	**Method**	**Accuracy %**	**Precision %**	**Recall %**	**F1 %**
(Ullah et al., 2022)	Hybrid	SVM	96.64	-	-	-
		LR	91.5			
		RF	86.68			
		DT	82.25			
(Dehkordy & Rasoolzadegan, 2020)	Hybrid	K-NN	97.83	-	-	-
		ID3	94.36			
		SVM	93.54			
My Proposed Method	**Permissions**	**CNN**	**98.06**	**99**	**97.71**	**98**

Table 17. Comparisons of my approach with other related works

6.8 Conclusions

In this chapter, I have concentrated on Trojan detection on Android platforms. I have collected a new dataset which I have made available, and I delivered a novel neural network architecture that can detect Trojans with very high accuracy. The results indicate that by using the permissions of Trojan and genuine Android apps, Trojans can be detected in a straightforward way which can be useful to the research community and beyond.

7. Android Botnet Malware Detection

7.1 Introduction

Today, Android is one of the most well-known operating systems. It has millions of applications that are distributed through accredited or unofficial distributors. As a result, it is one of the most common targets for malicious cyber-attacks. The Play Store on Android is not very restrictive, making it simple to install malicious apps. Botnet applications are classified as malware because they can be distributed through these stores and downloaded by unlucky users onto their smartphones. Botnets are among the most dangerous hacking techniques used on the internet today. Botnet developers frequently target smartphone users to instal malicious tools and target a larger number of devices. This is frequently done to gain access to sensitive data such as credit card numbers or to cause damage to individual hosts or organisational resources through denial of service (DDoS) attacks (Hijawi *et al.,* 2021; Alothman & Rattadilok, 2018).

Botnet attacks have become a threat and risk to the network and internet security in recent years. They include several malicious activities in network traffic. A Botnet is made up of separate robot and network components. The botmaster programmes and builds the bot for specific purposes using computers known as zombies. In the network, these computers are clearly breaking the law. Botnets are extremely widespread and can affect millions of computers. Botnets are networks made up of personal computers and smart devices known as bots. One or more attackers, known as botmasters, oversee these bots, and their goal is to carry out malicious activities. In other words, bots contain harmful software that runs on host computers and enables the botmaster to remotely command and control the system (Hosseini *et al.,* 2022).

The popularity and adoption of Android smartphones have attracted malware authors to spread the malware to smartphone users. Malware on smartphones can take the form of Trojans, Viruses, Worms, or mobile Botnets. Mobile Botnets, also known as Android Botnets, are more dangerous because they pose serious threats by stealing user credentials, sending spam, and launching distributed denial of service (DDoS) attacks. A mobile Botnet is defined as a collection of compromised mobile smartphones that are controlled by a botmaster via a command and control (C&C) channel and used to carry out malicious activities (Yusof *et al.*, 2018).

Although numerous studies have been conducted to detect Android Botnet attacks, classification accuracy can still be improved. Insufficient or smaller data in the experiments results in lower accuracy. Machine learning is incapable of handling large amounts of unstructured data because it typically requires structured data and uses traditional algorithms. The small size of the dataset is also to blame for Android Botnet detection's poor performance. Because the size of the sample data collection is limited, the confidence in the estimate decreases and the uncertainty increases, resulting in lower precision. More data is always a good idea when it comes to achieving the high efficacy of Android Botnet detection. Furthermore, the use of untrained data affects an effect on the detection of Android Botnets. Trained data is the most important and primary data that machines use to learn and predict. Increased training data provides more information and assist in better user fit (Balasunthar & Abdullah, 2022).

As a result, according to the explanation provided, there is an urgent need to develop new methods for defeating mobile Botnets. Because of the popularity of Android mobile devices, the goal of this chapter is to propose an innovative method for detecting Botnets in Android-based devices. This study aims to produce a new mobile Botnet classification/detection based on permissions. For this purpose, I have created and introduced a new dataset based on permissions. Moreover, the Botnet dataset is a classification dataset that only includes legitimate Android apps and Botnets. I have created an optimised multilayer perceptron neural network (MLP) that is highly accurate at detecting Botnets.

The contributions of this chapter are as follows:

- I introduced a novel dataset with 454 permissions as features to discover Botnets through the Android operating system.
- I proposed an optimised multilayer perceptron neural network (MLP) to detect Android Botnets.
- I evaluated my proposed method using well-known metrics with the results showing that Botnets can be detected with very high accuracy practically.

7.2 Background

Botnets are a type of malware that enables an attacker to gain control of a victim's computer. The botmaster, C&C server, and bot-infected machines are common Botnet components. The Botnet is designed to infect mobile phones or computers and make them under the control of Botnet owners or the "Botmaster". Botmasters are those who operate the command and control of Botnets to attack the target via a communication channel, such as HTTP, Internet Relay Chat (IRC), or peer-to-peer (P2P). The botmaster will use a Botnet to attack the victim in a variety of ways, including denial of service (DDoS) attacks, spamming, malware and advertisement distribution, espionage, hosting malicious applications, and other activities. The overview of a Botnet is demonstrated in Figure 27.

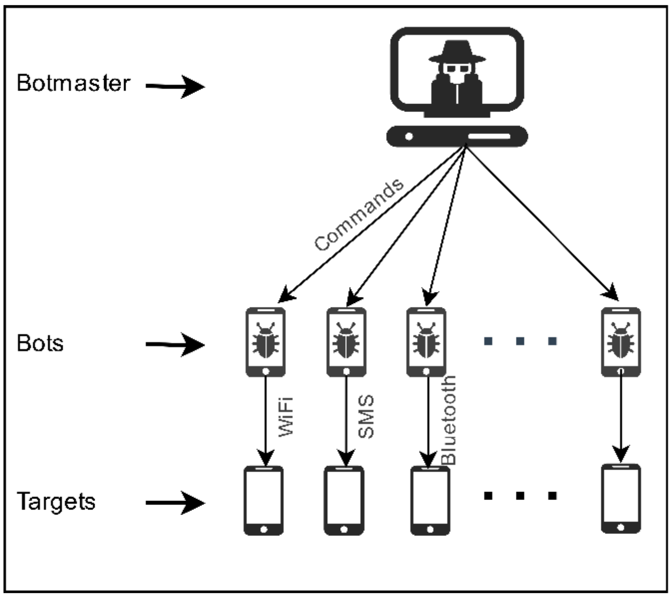


Figure 27. Overview of a Botnet structure

A Botnet includes three types of programmes:

A. ***Server programmes:*** These programmes are located on the command-and-control server and are used to control infected computers or bots.
B. ***Client programme:*** These are programmes installed on infected computers while they wait for control instructions.
C. ***Malicious programme:*** These are the software or programmes, also known as malware, which are used over the Internet to infect or compromise vulnerable computers.

Communication is the most important aspect of a Botnet. The command-and-control server continues to communicate with bots, instructing them to engage in malicious behaviour. The bots, in turn, continue to wait for instructions, perform the tasks assigned to them, and send the collected data to the command-and-control server (Baruah, 2019).

In general, Botnets have four main phases in their lifecycle.

A. *Phase Of Spread and Infection:*

Botmasters will employ various methods and techniques to infect new targets and transform them into new bots. After infecting the target, it will run a script or shell code and instal itself on the victim machine.

B. *Phase Of Command & Control:*

The command and control (C&C) mechanism creates a communication interface between bot-bot, C&C servers-bots, and C&C servers-bot master. Command and control mechanisms are classified into three types: centralised, decentralised, and unstructured.

C. *Phase Of Attack:*

The Botnet is a collection of malicious activities that spread throughout computer networks. DDoS attacks, spamming, spreading malware and advertisements, espionage, and hosting malicious applications and activities are just a few examples of attacks.

D. *Phase Of Destruction:*

After performing malicious activities, botmasters may destruct a portion of the Botnet (Tansettanakorn *et al.*, 2016).

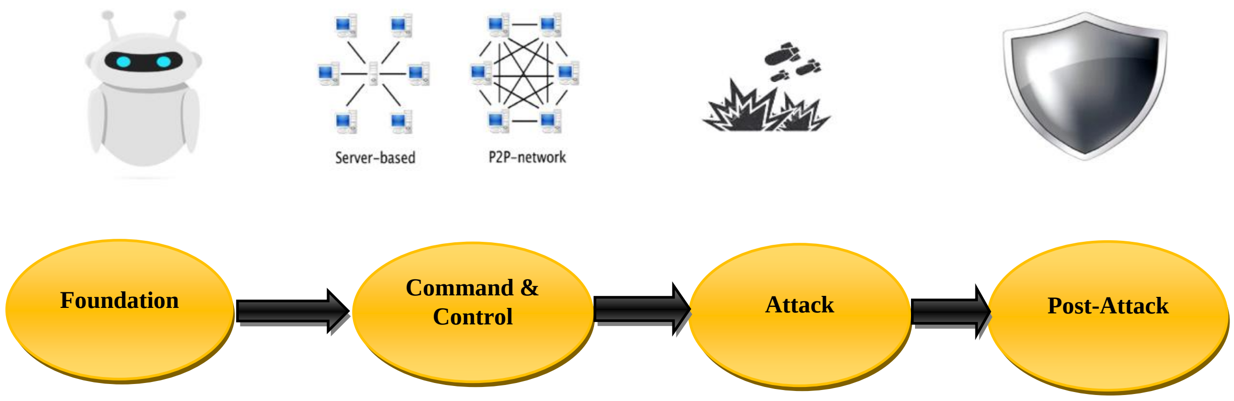


Figure 28. Life cycle of a Botnet

Botnet attacks are typically carried out by a group of hackers, and the owner has no idea that he or she is on the victim list. Botnets are currently classified into five types based on the Command and Control (C&C) channel. Because the programme is developed by the methods and techniques employed, the Botnets are divided into these categories. They are as follows:

A. ***IRC Botnet (Internet Relay Chat):*** An IRC Botnet is created by using a centralised system to monitor the victim to perform malicious activities, and the targeted bots are controlled by the main C&C channel.
B. ***P2P Botnet (Peer to Peer):*** It is accomplished using P2P protocols and a decentralised system with a network of nodes that keeps it alive, containing the attacked bots as well as all relevant data transmission.
C. ***HTTP Botnet:*** An HTTP Botnet is a centralised system-based structure that conducts attacks via the HTTP protocol. The bots use a specific URL and IP address specified by the main botmaster as the C&C server. These hacking attempts are carried out for financial theft.
D. ***Mobile Botnet:*** This attack makes use of mobile phone sharing, Bluetooth technology, and text messaging. The botmaster can easily access the data using this method via the C&C Channel.
E. ***Botnet Cloud:*** This is a very difficult task, so the botmaster creates and manages the bots using the cloud service, putting the bots at significant risk of being discovered. (Natacea, 2023), (Spiceworks, 2023)

7.3 Experimental Evaluation

I have developed an optimised Multilayer Perceptron (MLP) using Python and the Scikit-learn library, as described in section 5. Moreover, 5-fold cross-validation has been used throughout all experiments. Using the proposed Botnet dataset, I trained and validated my MLP neural network classifier using the Python programming language. The NumPy and Pandas libraries are required for array operations and reading data from files. The simulation is divided into four stages: defining the network's parameters, such as node numbers and learning rate, reading the Botnet dataset, training the MLP neural network with a portion of the dataset, and finally verifying the neural network with the rest of the dataset.

7.4 Evaluation Metrics

I used the Python programming language with the sci-kit-learn library for the experimental analysis. I have used Accuracy, Precision, Recall, and F1 as evaluation metrics; these metrics are described in equations 1, 2, 3, and 4 respectively. True positive, true negative, false positive, and false negative are all abbreviated as TP, TN, FP, and FN, respectively. Equation 1's Accuracy demonstrates overall performance. Another important metric is Precision, which is calculated using equation 2 and describes the percentage of predicted Botnets. The Recall metric, or the percentage of Botnet that is correctly classified, is defined by Equation 3. The F1-score is a number between 0 and 1 that represents the harmonic mean of precision and recall as calculated by equation 4.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{4}$$

7.5 Experimental Results

This section describes the experiments and compares the proposed method to other well-known classifiers as well as the most relevant previous research in this field. For evaluating the proposed method, I used my hand-crafted dataset and selected 1229 Android botnet samples from 7 different families. All the benign samples were scanned with the VirusTotal to make sure that the benign class does not include any malware samples. The dataset consists of 2713 samples, and 5-fold cross-validation was used to evaluate the proposed method using this dataset. All experiments were performed on a 64-bit Microsoft Windows 11 pro–operating system and using hardware with intel(R) Core (TM) i5-8365U @ 1.60GHz 1.90 GHz CPU, 16.00GB RAM, and an Intel UHD Graphics 620 GPU.

7.6 Comparisons with Other Classifiers

The following algorithms were used in the comparisons, with the default settings from the sci-kit learn library: Decision Tree, Random Forest, KNN, SVM, and Naive Bayes. The results are shown in Table 18 which compares the proposed method to other well-known classifiers based on Accuracy, Precision, Recall, F-1, and AUC using 5-fold cross-validation. In order to highlight the significance of this research result, a comparison is made with previous similar research. Table 19 indicates the comparison between (Hojjatinia *et al.*, 2020; Yerima & To, 2022; Yerima & Bashar, 2022; Balasunthar & Abdullah, 2022) respectively. These comparative results show that the research method surpasses previous similar efforts.

Algorithm	Accuracy%	Precision%	Recall%	F-1%	AUC%
Decision Tree	96.50	98.36	94.14	96.20	96.37
Random Forest	98.15	97.63	98.41	98.02	98.17
K-NN	98.34	99.52	96.31	97.89	98.00
SVM	97.60	98.39	96.45	97.41	97.53
Naïve Bayes	79.18	68.53	99.59	81.19	81.00
My Proposed Method	**98.88**	**99.99**	**98.46**	**98.88**	**98.80**

Table 18. Comparisons with other classifiers

The algorithms were implemented in Python using default parameters from scikit-learn. While the proposed MLP achieves the highest accuracy, precision, recall and F1 scores, examining the model settings provides more insight. The KNN model obtained strong results with k=5, indicating 5 nearest neighbours works well for this problem. The RBF kernel proved effective for SVM, able to capture non-linear patterns. Naive Bayes suffered from the simplicity of its assumptions. The neural network's hidden layers extract hierarchical feature representations, capturing complex data relationships missed by simpler models.

7.7 Comparisons with Other Related Works

Table 19 shows the obtained results from the proposed method compared to other researchers that have used traditional Machine Learning approaches (in terms of the used dataset, number of samples, and performance). The table indicates that the proposed method is completely successful in classifying benign and Botnet applications. Moreover, I distinguished Botnet and benign applications just by utilising given permissions, while some of the mentioned works employed more features alongside given permissions such as API calls or permissions protection level. My promising results indicate that my method can detect Android Botnets based on only given permissions as features with high accuracy.

Reference	**Type**	**Method**	**Accuracy%**	**Precision%**	**Recall%**	**F-1%**
(Yerima & Bashar, 2022)	Images and a manifest file	HOG	97.5	98.0	98.0	98.0
(Balasunthar & Abdullah, 2022)	Permissions	CNN-SVM	96.9	-	-	96.9
(Yerima & To, 2022)	Manifest file texts	CNN	95.44	95.4	95.4	95.4
		ANN	96.35	96.4	96.4	96.3
(Hojjatinia *et al.*, 2020)	Permissions	CNN	97.2	95.5	96	95.7
My Proposed Method	**Permissions**	**MLP**	**98.88**	**99.99**	**98.46**	**98.80**

Table 19. Comparisons of my approach with other related works

The dataset sizes vary across works and may impact accuracy. The proposed MLP obtains the highest scores using 2,713 samples, more than in other works. API calls and manifest data are leveraged alongside permissions in some methods, but my approach relies solely on permissions. This highlights their significance for Botnet detection. While CNNs have proven effective for malware classification, my MLP architecture directly models feature relationships. Training time is another consideration - deep networks can be computationally intensive to train versus simpler models. In summary, the permissions-only feature extraction, larger dataset, and optimized MLP architecture contribute to superior precision, recall and F1 over previous methods.

7.8 Conclusions

In this chapter, I employed Android permissions and an optimised multilayer perceptron (MLP) to propose a novel method to detect Android Botnets. To the best of my knowledge, this is the first Android Botnet detection method that applies a dataset with 454 permissions as a feature. Initially, I downloaded 2713 APK files from various categories from the Google play store and other third-party websites to create our intended dataset based on permissions. Then, reverse engineering was applied on 1483 benign and 1229 Botnet applications from my hand-crafted dataset to extract the AndroidManifest.xml files that provided access to the permissions given to each application. Finally, I trained and tested a proposed MLP model using the employed dataset in a 5-fold cross-validation experiment. Based on my experiments, the proposed methodology outperforms several conventional ML methods in this field by achieving 98.88% accuracy and 99.99% precision. These promising results indicate that the proposed methodology can practically detect Android Botnets by employing the given permissions.

8. Android Malicious Adware Detection

8.1 Introduction

The global smartphone market has experienced exceptional growth, thanks to the vast number of available applications and an open-source code platform that encourages app developers to create Android apps for free. These applications are considered essential to Android phones, as they drive innovation, leisure, accessibility, and compatibility with mobile devices. However, Adware or mobile advertising in the form of banner ads, rich media, or interstitial ads is increasingly infiltrating Android applications, posing a threat to users. The use of mobile devices has become widespread in recent years, and they are now essential to our daily lives, resulting in an exponential increase in mobile users and apps. Android, with over 2 billion users, is the most popular operating system worldwide and is thus a common target for malicious actors. Adware, a type of malware that displays unwanted and often offensive advertisements, is a prevalent threat to Android users (Javaid *et al.*, 2018; Alani & Awad, 2022b).

Advertising is a marketing strategy that promotes products, ideas, and services. With the emergence of the internet and smartphones, digital advertising has become increasingly popular. Digital ads are displayed while browsing websites or using mobile applications, and since many smartphones run on the Android platform, there is a vast ecosystem of Android apps. However, digital advertising is plagued by fraudulent activity, with Adware being a prevalent security threat used to collect marketing data or display ads for profit. Adware is more common and effective than traditional malware and goes beyond the judicious advertising found in freeware or shareware. Adware is often installed alongside other programmes and can continue generating ads even when the user is not running the desired programme (Ndagi & Alhassan, 2019).

Smartphones have become essential tools in our daily lives, and the amount of sensitive information we store on them has made them prime targets for hackers. Malicious code can be installed on smartphones, allowing hackers to steal personal information and profit from advertising and micropayment systems. The number of mobile malware infections has grown exponentially, and Android devices are particularly vulnerable due to the openness of the

Android market and their high market share. Malicious Adware-based hacking attacks have become more intense and diverse over time, with the most common type infiltrating and controlling users' Android devices. Malicious Adware has infected almost all current Android versions, making many Android devices worldwide vulnerable to threats. Advertisements in some free Android apps have become more aggressive, and users may not be aware of the changes on their devices. Even popular apps may contain Adware, and users may not object to it or be aware of its effects (Lee & Park, 2020).

Machine learning is an advanced technique used in cybersecurity to identify and classify malware, including Adware, based on patterns and behaviours rather than just matching signatures. Machine learning algorithms are trained on large datasets of known malware and can detect new and previously unknown malware. These algorithms can learn to recognise and classify Adware based on its behaviour, such as its use of network connections, data transmission, and system modification, among other things. By detecting Adware through behaviour analysis, machine learning can identify new strains of Adware that traditional signature-based methods might miss. Machine learning can also help security experts quickly develop new rules and signatures for identifying Adware and other malware, allowing for faster and more effective responses to new threats. Machine learning is expected to play an increasingly important role in cybersecurity as new and more complex forms of malware continue to emerge (Bagui & Benson, 2021).

Therefore, a different approach is proposed to detect and classify Adware-based permission analysis using deep learning. This chapter describes a permission-based Adware detection algorithm that uses a Convolutional neural network (CNN). This research analyses the use of machine learning as a possible defence against mobile Adware. I classified Android apps based on the features obtained from static analysis. The static features, which are permissions, are obtained from the VirusTotal Scanner website. To detect Android Adware using permissions, I first created a new dataset and then utilised a tuned CNN algorithm. I employed a deep-learning technique to analyse Android Adware and benign apps, based on the dataset that I have created. Specifically, I considered experiments involving neural networks. This study compared several ML algorithms, namely, Decision Tree (DT), Random Forest (RF), K-Nearest Neighbour (K-NN), Support Vector Machine (SVM), Naive

Bayes (NB), Multilayer Perceptron (MLP), Logistic Regression (LR), and Linear Discriminant Analysis (LDA).

8.2 Background

Adware is often distributed alongside other software, especially free software, or software that users may be tempted to download. Once installed on a user's device, the Adware can track their online activities and display targeted advertisements, which can be annoying or even harmful. Adware can also slow down a device's performance and cause it to crash. To avoid Adware, users should be cautious when downloading software and only download from trusted sources. They should also keep their devices and software up to date with the latest security patches and use reputable antivirus software. Additionally, users can install ad-blocking software or browser extensions to block unwanted advertisements (Narayanan *et al.,* 2014).

Adware is installed on a user's device through a security flaw in an existing application. Users can unknowingly download it as well. This can occur when users download an Adware-infected application or use software that contains flaws that Adware authors can exploit. Adware's goal is to get users to click on or otherwise interact with advertisements. Adware developers and distributors profit when users click on the online advertisements served by their Adware. There is legitimate Adware that users agree to, but it is often unwanted. Adware is frequently just an annoyance, but it can also contain malicious threats. Figure 29 demonstrates how malicious Adware works.

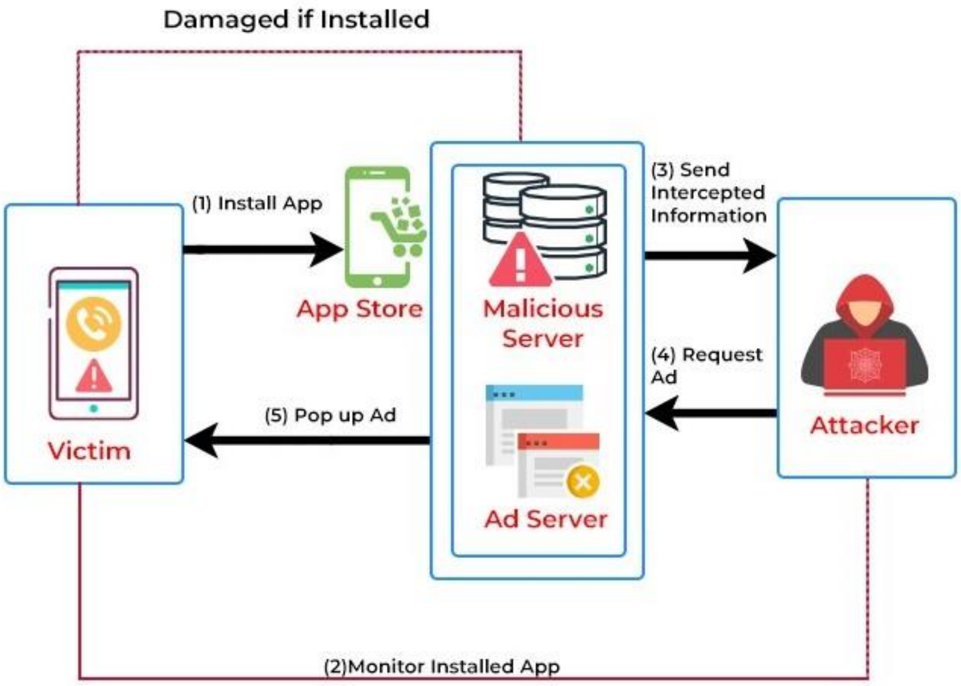


Figure 29. Demonstration of how malicious adware works

Adware is any software programme, whether malicious or not, that can display advertisements on a personal computer. Malicious programmes that display misleading advertisements, blinking pop-up windows, massive digital billboards, and full-screen auto-play advertisements within an internet browser are the most common examples. The term is a compound of the words "advertising" and "software." The developer earns money every time someone clicks on an advertisement displayed by Adware. Some types of Adware may interfere with your web browsing experience by directing you to malicious websites. Then, without your knowledge, some collect your browsing information and use it to serve you advertisements that are more tailored to your preferences and thus more likely to be clicked on. When Adware first became popular in 1995, many industry professionals assumed it was all spyware, which is software that allows someone to obtain covert data from a computer without the user's knowledge. Adware was demoted to the status of a "potentially undesirable application," or PUA, as its credibility grew. As a result, despite its widespread use, little was done to ensure its legality. Adware makers did not begin monitoring and blocking problematic behaviour until the peak Adware years of 2005-2008.

Many people mix up Adware and malware, malicious software designed to harm a computer or server. Malware includes viruses, spyware, worms, and some types of Adware. Pop-up ads, inaccessible panels, and other types of malicious Adware can infect computers. Once dangerous Adware has infiltrated a computer, it may perform a variety of malicious activities, such as tracking the user's location, query history, and web browser viewing history, which the malware programmer can then monetize by selling to third parties.

Adware is a cyber-security term that refers to Adware applications that exhibit dangerous or irregular behaviour. Adware is classified as spyware when it tracks users' activities without permission. Fraudsters take advantage of flaws in the validation process of ad networks or flaws in a consumer's browser. When a user visits an infected website, malicious Adware can spawn pop-ups, pop-unders, and persistent windows that allow for drive-by installations. Visitors who disable ad blockers may be at risk of infection. Adware applications have been discovered that prevent antivirus software from running. Because some Adware software is legal or does not have uninstallation processes, security software may be unable to identify which Adware applications are truly dangerous.

Adware is most commonly associated with annoying pop-up windows and advertisements, but it can also take other forms. It is critical to distinguish between harmless and dangerous Adware. The following are the most common types of Adware:

1. Legitimate Adware

Adware of this type allows you to subscribe to advertisements and software promotions, allowing developers to distribute their programmes for free by offsetting their expenses. Users instal this type of Adware on purpose to obtain a free item. You can also choose to allow it to collect marketing data. All programmers, including reputable ones, create legitimate Adware because providing clients with a free product is a legitimate and fair method of gaining adoption. However, not all software downloads are agreed upon by both the distributor and the user. In this situation, the line between legal and illegal blurs.

2. Potentially unwanted applications (PUAs)

PUAs are unwanted software packages that are bundled with legitimate complementary software applications. PUPs, or potentially unwanted programmes, are another name for them. Although not all PUAs are malicious, some may exhibit intrusive behaviours such as displaying pop-up advertisements or slowing down your device. It can slow down a computer's performance and potentially introduce security issues such as spyware and other unwanted software.

3. Legal abusive adware PUA

PUAs, both legal and abusive, are designed to bombard you with advertisements. Excessive advertising can be found in packaged software, internet browser toolbars, and other places. This is also legal because no malware is involved. Ads for fitness pills, for example, are common in Adware like this.

4. Legal deceptive adware PUA

This category includes legal Adware that deceives the user in some way. This type of PUA may make it difficult to uninstall secure third-party software. This strategy is occasionally used by legal Adware, and it is lawful if the developer did not put malware-infected advertising or software there on purpose. Unfortunately, certain Adware can inadvertently infect devices with malware.

5. Illegal malicious adware PUA

This category includes malicious Adware that is either illegal to use or distribute. The PUA earns money by distributing malicious programmes to machines such as spyware, viruses, and other malware. The malware could be hidden within the Adware, the websites it promotes, or other software programmes. The authors and distributors are spreading this threat on purpose and may employ aggressive tactics (Lutkevich, 2021).

According to (BasuMallick, 2022), deceptive and abusive Adware is designed to manipulate users into interacting with advertisements or to obtain consent through deceptive means, such as bombarding them with unwanted ads or making it difficult to uninstall unwanted software. While Adware is not inherently malicious, it can create vulnerabilities that may be exploited by malicious software. Only Adware that is specifically designed to deliver harmful software to the user is considered malicious. Some types of malicious Adware include the following:

1. Spyware: Adware can contain code that tracks and records a user's personal information and internet browsing habits. If this data is collected without the user's knowledge or consent and sold to third parties, it is considered spyware. Many privacy advocates are critical of these practices.

2. Potentially unwanted programs (PUPs): Malicious Adware or spyware can be bundled with free or shareware software downloaded from the internet. Users may unknowingly download Adware from an infected website. Antimalware programs often flag Adware as a potentially unwanted program, regardless of whether it is malicious or not.

3. Man-in-the-middle (MitM) attacks: Adware can also be used in MitM attacks, where the attacker routes user traffic through the Adware vendor's system, even over secure connections. The communicating parties believe they are exchanging information securely, but the attacker can collect and manipulate sensitive information during the conversation.

Some common advertisement attacks are described and explained in the sections that follow. The impact of threats and attacks on an adversary, developers, users, and platform ends is described in Table 20, which summarises their relationship.

Attacks/Threats	Adversary End	Developer End	User End	Android End
Malware	Unauthorized Data Collection	Code Level Security Issues	Victim	Data Exposure to other apps
Malicious Ads	Attack Generator (e.g., DDoS attack)	Victim	Victim	Lack of Security Control
Malicious Ad-Libraries	Attack Generator	Money making & Lack of Security Check	Victim	No Privilege Escalation
Permission Misuse	Developer’s accessed Permissions misuse	Ad-Libraries accessed Permissions misuse	Victim	No Privilege Escalation
MitM	Attack Generator	Targeted Applications	Victim	Lack of Security Control
Certificate Compromise	Targeted Sites via Valid certificate	Lack of Security Check	Victim	Lack of Security Control
Click-Fraud Attack	Victim	Make Money or Exhaust Adversary	Security Exploitation	Lack of Security Control

Table 20. Attacks/Threats and their impact on user’s privacy

Attack/Threats are described below:

- ***Adversary End:*** This refers to an attack that occurs due to the presence of malicious advertisement libraries or networks. These libraries or networks may be designed to serve ads that contain malware or to redirect users to phishing or other malicious websites. When users interact with these ads, they may inadvertently download malware or give away sensitive information.

- ***Developer End:*** The two directions in which developers launch attacks are against advertisers, to exhaust their budget and abuse power, and against users, to steal personal information or make money. The terms "Adversary End" and "Developer End" are interrelated because app developers and ad libraries collaborate with each other. They share permissions and are located in the same code piece with the same UID. These two categories are separated only for the purpose of better understanding their behaviour.
- ***User End:*** refers to the end-users of a system who may be vulnerable to attacks due to their lack of security awareness, knowledge, and implementation of existing defensive measures. This means that users may not be aware of potential security risks and how to protect themselves from them, such as using strong passwords, avoiding clicking on suspicious links, and regularly updating their software. As a result, they become easy targets for attackers who can steal their personal information, and financial data, or use their devices for malicious purposes.
- ***Android End:*** The security control of the Android platform is a significant factor, which has some vulnerabilities, including the absence of privilege escalation, ad SDKs, and application code permissions for developers.

As described in Table 20 summarizes the relationship between them where the impact of attacks/threats on 'Adversary developers', users and platforms are described.

- ***Malware:*** The term malvertising is used to describe online advertisements that distribute malware (Lutkevich, 2021). Malware can be transmitted to an Android device through malicious software or advertisements. Users may be directed to other pages where they can download additional software or malicious applications by clicking on these ads while using the app. The majority of Android malware is in the form of Trojans.
- ***Malicious Ads:*** Malicious Ads: It is related to malware injection in some ways. The separation from malware serves only to categorise malicious advertisements. Malware can, however, be injected through the code of developers or malicious libraries.
- ***Click-Fraud Attack:*** Adware attacks can occur not only on Android devices but also on ad networks. These attacks aim to exploit security vulnerabilities in the Android platform and other systems. Click fraud is an example of a cyber-criminal activity that

has become increasingly common. Attackers use Adware to generate fraudulent clicks, with the goal of either increasing revenue for developers or depleting the budget of advertisers.

- ***Malicious Ad-Libraries:*** One type of attack involves the use of malicious ad-libraries that have access to sensitive information. These libraries can collect data through the permission mechanism and send targeted ads to users. A developer can use up to 65 of these libraries simultaneously, giving them access to a significant amount of personal data. This type of attack can be especially dangerous as it can result in the theft of sensitive information.
- ***Permission Misuse:*** Permissions are crucial for application security, but their misuse can lead to various attacks. Malware injection, malicious ads, malicious advertisement libraries, and authorities misusing permissions are the primary causes of Android phone rooting. However, rooting is only beneficial if done by the user to gain complete access to the device. Malicious software can also root the phone and take complete control of it, which is a significant security concern. Therefore, applications should not be granted "super-user access" even if the phone is rooted.
- ***Man-in-the-Middle Attack:*** a type of attack that targets smartphone users. This attack is classified as MiTM because it involves intercepting communication between two parties, with the attacker positioned in the middle, allowing them to read or modify data in transit. Examples of this type of attack include SSL hijacking, SSL stripping, and DNS spoofing.

8.3 Experimental Evaluation

I developed a tuned convolutional neural network (CNN) with Python and the Scikit-learn, Keras, and TensorFlow libraries. Furthermore, 5-fold cross-validation was used in all experiments. I used the Python programming language to train and validate my neural network classifier on my malicious Adware dataset. For array operations and reading data from files, the NumPy and Pandas libraries are required. The simulation is divided into the following stages: defining the network's parameters, such as node numbers and learning rate, reading the dataset, training the neural network, and finally validating the neural network with the remaining dataset. Figure 30 demonstrates the simulation stages.

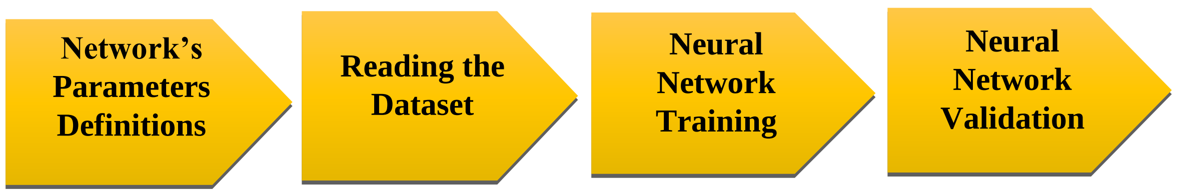


Figure 30. Demonstration of simulation stages

8.4 Evaluation Metrics

In the experiments conducted, I utilised the Python programming language, alongside the scikit-learn, Keras, and TensorFlow libraries. To assess the performance of our approach, I employed evaluation metrics such as Accuracy, Precision, Recall, and F1, which are specified in equations 1, 2, 3, and 4, respectively. The acronyms TP, TN, FP, and FN correspond to true positive, true negative, false positive, and false negative, respectively. Accuracy, calculated via Equation 1, provides an overall indication of model performance. Precision, determined using Equation 2, describes the proportion of predicted Adware and is another vital metric. The Recall metric, as defined in Equation 3, represents the percentage of correctly classified Adware. Additionally, I utilised the F1-score, which is a number between 0 and 1 that determines the harmonic mean of precision and recall, as expressed by Equation 4.

The basic four performance measures of a binary ML-based classifier are:

- True Positives (TP) is a performance metric that represents the number of positive samples that are correctly classified as positive by a binary machine learning classifier. It is calculated by dividing the number of test instances are true and predicted values are 1 (positive) by the total number of test instances whose true value is 1.
- False Positives (FP) refer to the number of instances in which a negative sample is predicted as positive by a binary machine learning classifier. It is calculated as the number of test instances whose true value is 0 and the predicted value is 1, divided by the number of test instances whose true value is 0.
- True Negatives (TN) refer to the number of negative samples that are correctly classified as negative by the binary classifier. Specifically, it is the number of test instances whose true value is negative (0) and the predicted value is also negative (0), which is then divided by the total number of test instances whose true value is negative (0).

- False Negatives (FN) are the number of positive samples that were incorrectly classified as negative. This is calculated by counting the number of test instances whose true value is 1 (positive) and the predicted value is 0 (negative), and then dividing by the total number of test instances whose true value is 1 (positive).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{4}$$

8.5 Experimental Results

This section describes the experiments and compares the proposed method to other well-known classifiers as well as the most relevant previous research in this field. I used a self-made dataset to evaluate the proposed method and selected 500 Android malicious Adware samples from 10 families. All the benign samples were scanned through the VirusTotal scanner to make sure that the benign class does not include any malware samples. The dataset contains 2000 samples, and the proposed method was evaluated using 5-fold stratified cross-validation on this dataset. In addition, all experiments were carried out on a 64-bit Microsoft Windows 11 Professional operating system with hardware including an Intel(R) Core (TM) i5-8365U @ 1.60GHz 1.90GHz CPU, 16.00GB RAM, and an Intel UHD Graphics 620 GPU.

8.6 Comparisons with Other Classifiers

In the comparisons, the following algorithms were used with the sci-kit learn library's default settings: Decision Tree (DT), Random Forest (RF), K-Nearest Neighbour(K-NN), Support Vector Machine (SVM), Naive Bayes (NB) and Multilayer Perceptron (MLP). Figures 31 and 32 show the training/test accuracy and loss over the course of 5 epochs for each of the five folds for the 5-fold cross-validation for the 1st run the final evaluation results of which are presented in the top row of Table 21. Table 21 shows the accuracy, precision, recall and F1 scores across 5 different cross-fold executions and at the end is the average of these completed executions. Table 22 compares the proposed method to other well-known classifiers in terms of Accuracy, Precision, Recall and F-1 using 5-fold cross-validation.

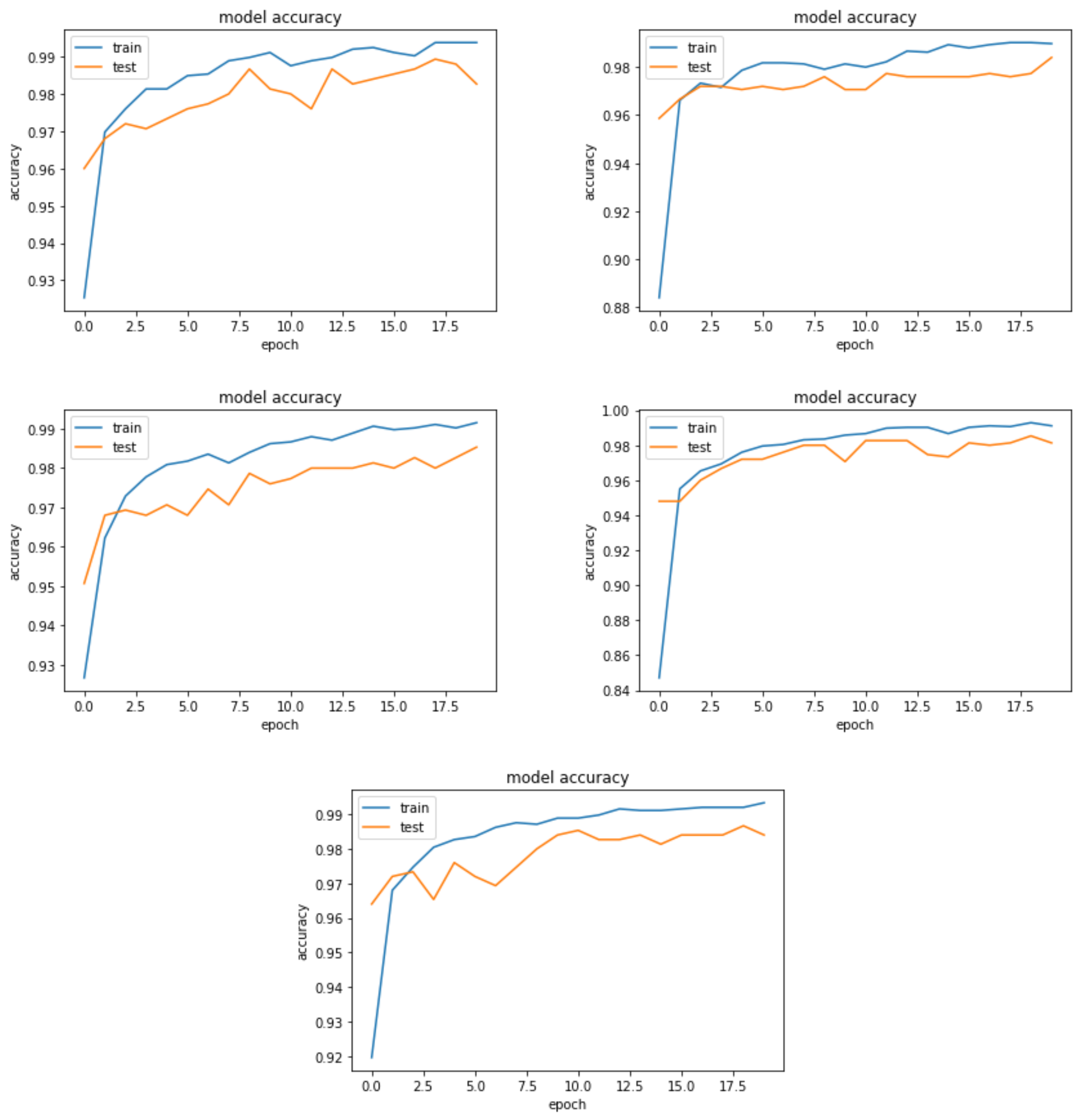


Figure 31. Training and test accuracy over epochs

Figure 31 shows the training and testing accuracy of the proposed MLP model over 5 epochs. The training accuracy increases rapidly in the first 2 epochs as the model learns, exceeding 95%. The testing accuracy also steadily improves, indicating successful generalization. The minimal gap between training and testing demonstrates little overfitting. The minor fluctuations are due to shuffling the order of samples for each training batch between epochs. The model converges by epoch 5 with final training and testing accuracy around 98%, aligning with the strong results in Table 21. This confirms the malware detection capability of the MLP architecture.

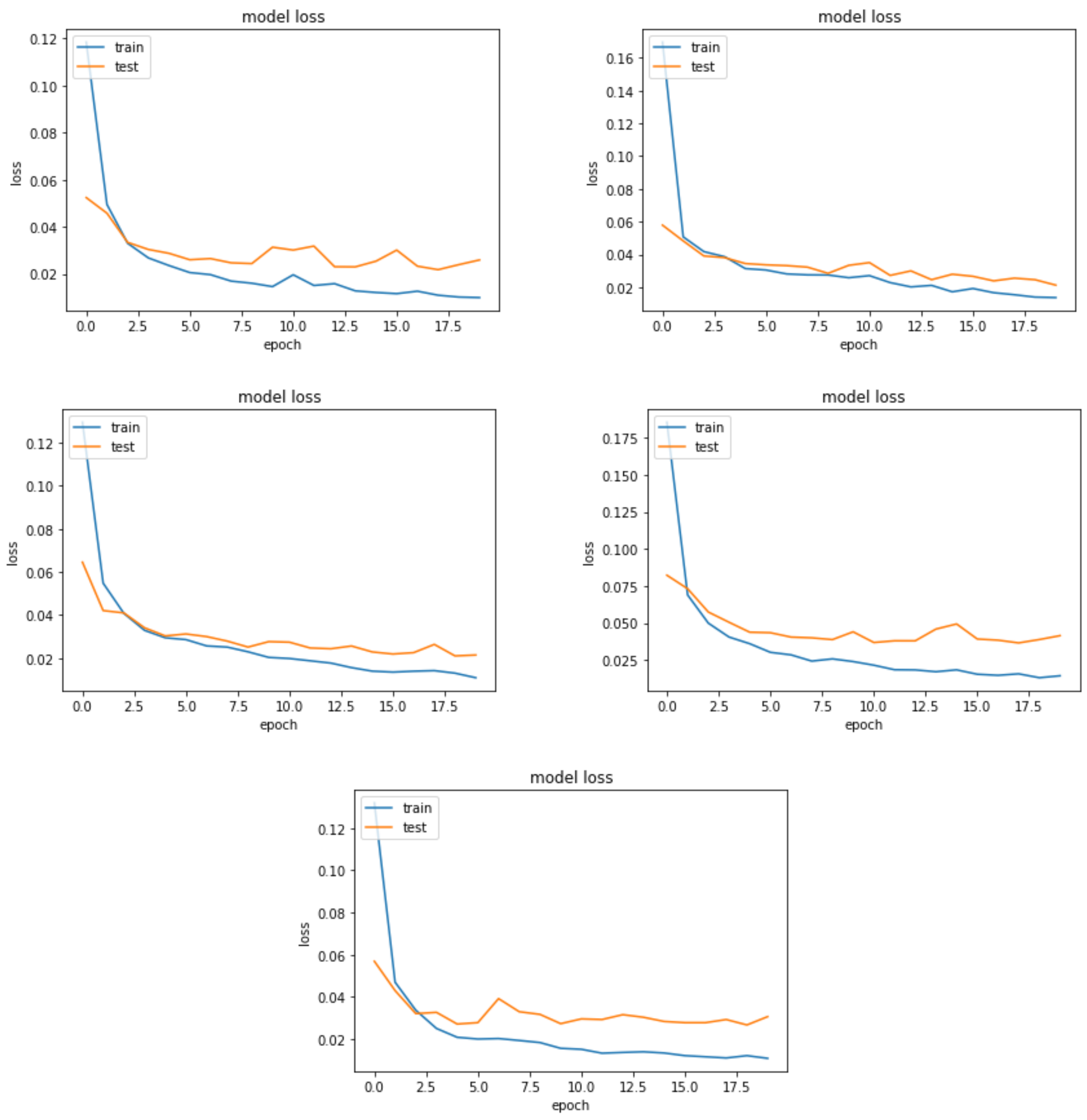


Figure 32. Model loss over epochs

Figure 32 presents the training and testing loss curves over epochs. As expected during model convergence, the loss decreases over time for both training and testing data. The testing loss follows but slightly exceeds the training loss, reflecting a small generalization gap. Minor spikes are attributed to shuffling samples between epoch batches. Lower loss directly correlates with higher accuracy, so the final low losses validate the excellent accuracy achieved in Table 21. In summary, the loss plots confirm successful training of the MLP model to accurately classify Android Botnets based solely on permissions.

Execution No	Accuracy %	Precision %	Recall %	F1 %
Run 1	98.35	98.51	98.19	98.34
Run 2	98.53	98.41	98.67	98.53
Run 3	97.76	97.21	98.35	97.77
Run 4	98.91	98.83	98.99	98.90
Run 5	97.65	97.61	97.71	97.65
Average	**98.24**	**98.11**	**98.38**	**98.24**

Table 21. Evaluation results

Algorithm	Accuracy %	Precision %	Recall %	F1 %
DT	94.00	89.50	93.54	91.48
RF	96.66	94.20	94.89	94.54
K-NN	96.00	89.24	95.29	94.00
SVM	95.33	89.86	95.68	92.68
NB	94.88	92.68	93.25	92.96
MLP	95.77	91.30	95.71	93.92
My Proposed Method	**98.24**	**98.11**	**98.38**	**98.24**

Table 22. Comparisons with other classifiers

8.7 Comparisons with Other Related Works

Table 23 presents the obtained results from the proposed method using CNN compared to other aware studies that have used classic Machine Learning approaches. The table indicates that the proposed method is completely successful in classifying benign and malicious Adware applications. My results with high accuracy indicate that my method can practically detect Android malicious Adware based on only given permissions as features by applying a tuned CNN model.

Reference	Type	Method	Accuracy%	Precision%	Recall%	F1%
AdStop (Alani & Awad, 2022b)	Network Traffic	MLP[1]	95.7	94.7	93.9	93.7
(Dobhal *et al.*, 2020)	Adware Behavior	LR[2]	96.0	94.5	94.9	94.6
		LDA[3]	96.2	94.3	95.7	95.0
		K-NN[4]	95.1	92.4	95.2	93.6
		DT[5]	94.2	92.2	91.8	92.5
		NB[6]	68.8	71.2	77.1	67.0
(Lee & Park, 2020)	Features	Dynamic Random Forest[7]	96.6	95.1	95.4	95.1
AdDetect (Narayanan *et al.*, 2014)	Module of apps	SVM[8]	95.4	93.7	94.1	93.8
MadDroid	Permissions	CNN	**98.24**	**98.11**	**98.38**	**98.24**

Table 23. Comparisons with the other works

The algorithms used in the comparisons are the following:

[1] *Multilayer Perceptron: activation: 'relu', solver: 'adam', learning_rate_init: 0.001, 200 iterations*

[2] *Linear Regression: fit_intercept: True, normalize: False, copy_X: True, n_jobs: None, positive: False, precompute: False*

[3] *Linear Discriminant Analysis: solver: 'svd'*

[4] *K-Nearest Neighbor: n_neighbors: 5*

[5] *Decision Tree: criterion: "gini", splitter: "best*

[6] *Naive Bayes: priors: None, var_smoothing: 1e-9*

[7] *Random Forest: n_estimators: 100*

[8] *Support Vector Machine: C: 1.0, kernel: 'rbf', degree: 3, gamma: 'scale', coef0: 0.0, shrinking: True, probability: False*

8.8 Conclusions

In this chapter, I present a novel method for detecting malicious Android Adware using Android permissions and a tuned convolutional neural network. To the best of my knowledge, I am the first researchers to use permissions as features and apply the CNN model to detect malicious Android Adware. To begin, I downloaded 1500 benign APK files from the Google Play Store from various categories and 500 APK files infected with malicious Adware to create my own dataset based on permissions. The AndroidManifest.xml files that provided access to the permissions granted to each application were then extracted using reverse engineering from 1500 benign and 500 malicious Adware apps from my dataset. Finally, in a 5-fold cross-validation experiment, I trained and tested a proposed CNN model using the employed dataset. My experiments show that the proposed method outperforms several conventional ML methods in this field, achieving 98.24% accuracy and 98.11% precision. These promising results suggest that the proposed method can detect Android Adware using the permissions provided.

9. Research Conclusions and Proposed Future Works

9.1 Discussion

The results of my research demonstrate the efficacy of using machine learning techniques to detect malicious antimalwares and VPNs, Android Trojans, mobile Botnets, and malicious Adwares on Android devices. By leveraging application permissions, I was able to develop specialized datasets and models for each type of malware, which led to high accuracy rates in identifying harmful applications.

One of the key benefits of my approach is its ability to detect targeted threats that may be missed by traditional signature-based approaches. The use of application permissions provides a more fine-grained view of the behaviour of an application, which enables the detection of malicious apps that may appear benign based on their code alone. Additionally, the models I developed are able to adapt to new and emerging threats by learning from new data, making them more resilient to novel attacks. Another advantage of our approach is its scalability. With the rapid growth of mobile devices and applications, the number of potential threats is increasing exponentially, making manual detection infeasible. Machine learning techniques enable automated detection and classification of malware, which can significantly reduce the workload of security analysts and enable more timely responses to emerging threats.

Despite these benefits, there are some limitations to my approach. One limitation is the potential for false positives, where legitimate applications may be classified as malicious due to their permissions. While my models achieved high accuracy rates, they may still misclassify some applications, leading to unnecessary alerts and user frustration. This emphasizes the need for a balance between accuracy and usability in mobile security solutions. Another limitation is the reliance on the accuracy of the application permissions reported by the Android operating system. Malware authors may attempt to obfuscate their true behaviour by manipulating the permissions requested by their applications. This can lead to incorrect assumptions about the behaviour of an application and reduce the effectiveness of my models. Additionally, some malware may attempt to exploit vulnerabilities in the Android system to bypass permission checks altogether.

Despite these limitations, I believe that my work provides a valuable contribution to the field of mobile security. By demonstrating the effectiveness of machine learning techniques in detecting targeted mobile threats, I hope to inspire further research in this area and enable the development of more robust and effective security solutions for mobile devices.

9.2 Novelty

The novelty of my proposed approaches relies on in the security aspect and more particularly it uses permissions of Android malware to detect malicious antimalwares, VPNs, Trojans, Botnets and Malicious Adwares and shows how a neural network should be trained and tuned to maximize accuracy. It is the first method in the literature that is based on app permissions and neural networks to identify malicious antimalwares, VPNs, and Trojans, Botnets, and Malicious Adwares. While other works that are based on permissions exist, these are generally used for all kinds of android malware detection. Therefore, having a tuned neural network based on permissions it is a fast and accurate way to detect malicious antimalwares and VPNs only which is a very important sub-category of Android malware. While other methods can detect malware, they do it either for all types of malwares which requires large datasets and the methods applied to make it computationally expensive. Thus, this study investigates how a permission-based analysis can provide a robust method that is able to identify malicious Anti-malware, VPN as well as Trojan, Botnet, and Malicious Adware apps in Android. To achieve this, I collected antimalwares, VPNs and Trojans, Botnets, and Malicious Adwares from official online stores and analysed them using VirusTotal website. This work provides datasets of 1200 antimalwares, 1300 VPNs and 2593(1058 Trojans and 1535 Benign), 2713(1229 Botnets and 1483 Benign) and 2000 (500 Malicious Adwares and 1500 Benign) including permissions plus tuned neural network platforms which identify harmful antimalwares, VPNs, and Trojans, Botnets, and Malicious Adwares. The platforms are fast, secure, and reliable and can identify malicious antimalwares, VPNs, and Trojans, Botnets, and Malicious Adwares and uses an optimised neural network.

9.3 Limitations

The features used are limited to application permissions. Additional static or dynamic features like API calls, network traffic, resource usage etc. could provide a more comprehensive view of app behaviour and improve detection accuracy. Relying solely on permissions may miss some types of malware obfuscation techniques. Furthermore, I need to assess the robustness of my models against adversarial attacks. Adversaries may try to avoid detection by changing the permissions of the application or introducing subtle changes to the malware code. Future research will focus on developing techniques to detect and defend against such attacks. Finally, I'll need to expand my work to other mobile platforms like iOS and Windows Mobile. While my models are designed for the Android operating system, similar approaches could be used to address the growing threat of mobile malware on other platforms. All in all, there are numerous opportunities for improvement and expansion in this area's future work, and I hope that my work will inspire further research in this field.

9.4 Conclusion

Finally, by addressing the critical issue of detecting Android malware, this thesis has made a significant contribution to the field of mobile security. I have demonstrated that targeted and precise detection of various types of malware is possible using machine learning techniques and specialised datasets. I have achieved high accuracy in identifying malicious antimalwares and VPNs, Android Trojans, mobile Botnets, and malicious Adwares by proposing novel approaches to feature engineering and model optimisation. One of my approach's main strengths is the creation of tailored datasets for each type of malware. My datasets were able to capture unique features and behaviours of the malware by focusing on specific application permissions, allowing for more effective detection. Furthermore, my proposed neural network architectures, which included techniques like dropout, batch normalisation, and transfer learning, were optimised to achieve high accuracy rates while avoiding overfitting.

This study's findings have significant implications for the field of mobile security. Traditional signature-based approaches are becoming less effective as the number and sophistication of Android malware grows. My approach's targeted and precise detection has the potential to greatly improve the ability to detect and mitigate mobile malware threats, resulting in a safer and more secure mobile ecosystem for users. Although my research has focused on Android malware detection, the method I developed is applicable to other types of mobile devices and operating systems. The application of machine learning techniques and specialised datasets has the potential to transform the field of mobile security by providing a scalable and adaptable solution capable of keeping up with the rapid growth and evolution of mobile devices and applications.

Overall, this thesis adds to the field of mobile security by demonstrating the efficacy of machine learning techniques in detecting targeted mobile threats. I demonstrated that precise and scalable detection of various types of malware is possible by developing specialised datasets and optimised neural network architectures. This research has significant implications for mobile security, and I hope that it will inspire additional research and development in this field to address the growing threat of mobile malware.

9.5 Future Work

While my research has yielded promising results in detecting malicious antimalwares and VPNs, Android Trojans, mobile Botnets, and malicious Adwares based on application permissions, there are still several avenues for further investigation. One possibility is to investigate the use of features other than application permissions to improve the performance of our models. Device metadata such as hardware and software configurations, for example, could be included to provide a more comprehensive view of the device's security posture. Furthermore, network traffic and user behaviour data could be used to improve the accuracy of my models. Another possible avenue of research is to look into the use of alternative machine learning techniques such as ensemble models, deep learning, and transfer learning. Ensemble models combine multiple models to improve prediction accuracy, whereas deep learning techniques can learn more complex and abstract data representations automatically. Transfer learning can improve performance on new, related tasks by leveraging pre-trained models on similar tasks.

References

Aafer, Y., Du, W. and Yin, H. (2013). DroidAPIMiner: Mining API-level features for robust malware detection in android. In: *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST*.

Abdul Kadir, A. F., Stakhanova, N. and Ghorbani, A. A. (2015). Android Botnets: What URLs are telling us. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*.

Alani, M. M. and Awad, A. I. (2022a). AdStop: Efficient flow-based mobile adware detection using machine learning. *Computers & Security*, 117: 102718.

Alani, M. M. and Awad, A. I. (2022b). AdStop: Efficient flow-based mobile adware detection using machine learning. *Computers & Security*, 117: 102718.

Alothman, B. and Rattadilok, P. (2018). Android botnet detection: An integrated source code mining approach. In: *2017 12th International Conference for Internet Technology and Secured Transactions, ICITST 2017*.

Altman, N. S. (1992). An introduction to kernel and nearest-neighbor nonparametric regression. *American Statistician*, 46.

Amer, E. (2021). Permission-Based Approach for Android Malware Analysis through Ensemble-Based Voting Model. In: *2021 International Mobile, Intelligent, and Ubiquitous Computing Conference, MIUCC 2021*.

Amro, B. (2017). Malware Detection Techniques for Mobile Devices. *International Journal of Mobile Network Communications & Telematics*, 7.

Android Developers (2020). *Permission* (on-line).

AppBrain (2023). *Number of available applications in the Google Play Store* (on-line). https://www.appbrain.com/stats/number-of-android-apps. Accessed 1 March 2023.

Apvrille, A. (2012). Symbian worm Yxes: Towards mobile botnets? *Journal in Computer Virology*, 8.

Apvrille, A. (2014). The evolution of mobile malware. *Computer Fraud and Security*, 2014.

Armando, A., Carbone, R., Costa, G. and Merlo, A. (2015). Android Permissions Unleashed. In: *Proceedings of the Computer Security Foundations Workshop*.

Arora, A., Peddoju, S. K. and Conti, M. (2020). PermPair: Android Malware Detection Using Permission Pairs. *IEEE Transactions on Information Forensics and Security*, 15.

Arp, D., Spreitzenbarth, M., Hübner, M., Gascon, H. and Rieck, K. (2014). Drebin: Effective and Explainable Detection of Android Malware in Your Pocket.

Arshad, S., Ali, M., Khan, A. and Ahmed, M. (2016). Android Malware Detection & Protection: A Survey. *International Journal of Advanced Computer Science and Applications*, 7.

Arslan, R. S., Doğru, I. A. and Barişçi, N. (2019). Permission-Based Malware Detection System for Android Using Machine Learning Techniques. *International Journal of Software Engineering and Knowledge Engineering*, 29.

Arul, E. and Punidha, A. (2021). Adware Attack Detection on IoT Devices Using Deep Logistic Regression SVM (DL-SVM-IoT).

Atkinson, T. and Cavallaro, I. L. (2017). Hunting ELFs : An investigation into Android malware detection.

Aung, Z. and Zaw, W. (2013). Permission-Based Android Malware Detection. *International Journal of Scientific & Technology Research*, 2.

Avdiienko, V., Kuznetsov, K., Gorla, A., Zeller, A., Arzt, S., Rasthofer, S. and Bodden, E. (2015). Mining apps for abnormal usage of sensitive data. In: *Proceedings - International Conference on Software Engineering*.

Azmoodeh, A., Dehghantanha, A., Conti, M. and Choo, K. K. R. (2018). Detecting crypto-ransomware in IoT networks based on energy consumption footprint. *Journal of Ambient Intelligence and Humanized Computing*, 9.

Bagui, S. and Benson, D. (2021). Android Adware Detection Using Machine Learning. *International Journal of Cyber Research and Education*, 3.

'Bahar, Z. (2022). *Your free VPN app could be a trojan: How to spot fake vpns, NordVPN* (on-line). https://nordvpn.com/blog/fake-vpn/. Accessed 7 February 2023.

Balasunthar, S. and Abdullah, Z. (2022). Comparison of Convolutional Neural Network and Artificial Neural Network for Android Botnet Attack Detection. *Applied Information Technology And Computer Science*, 3.

Baruah, S. (2019). Botnet Detection : Analysis of Various Techniques Sangita Baruah a. *International Journal of Computational Intelligence & IoT (IJCIIoT): Proceedings*, 2.

BasuMallick, C. (2022). *What Is Adware? Definition, Removal, and Prevention Best Practices for 2022* (on-line). https://www.spiceworks.com/it-security/security-general/articles/what-is-adware/. Accessed 15 February 2023.

Bayazit, E. C., Sahingoz, O. K. and Dogan, B. (2022). A Deep Learning Based Android Malware Detection System with Static Analysis. In: *2022 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*. IEEE. pp.1–6.

Benats, G., Bandara, A., Yu, Y., Colin, J. N. and Nuseibeh, B. (2011). PrimAndroid: Privacy policy modelling and analysis for android applications. In: *Proceedings - 2011 IEEE International Symposium on Policies for Distributed Systems and Networks, POLICY 2011*.

Benton, K., Camp, L. J. and Garg, V. (2013). Studying the effectiveness of android application permissions requests. In: *2013 IEEE International Conference on Pervasive Computing and Communications Workshops, PerCom Workshops 2013*.

Bhatnagar, V. and Sharma, S. (2012). Data Mining : A Necessity For Information Security. *Journal of Knowledge Management Practice*, 13.

Blowers, M., Fernandez, S., Froberg, B., Williams, J., Corbin, G. and Nelson, K. (2014). Data Mining in Cyber Operations. *Advances in Information Security*, 61.

Bradley, A. P. (1997). The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30.

'BRATISLAVA, K. (2022). *Eset Research: Bahamut Group targets android users with fake VPN apps; spyware steals users' conversations, ESET* (on-line). https://www.eset.com/int/about/newsroom/press-releases/research/eset-research-bahamut-group-targets-android-users-with-fake-vpn-apps-spyware-steals-users-convers/. Accessed 7 February 2023.

Bugiel, S., Davi, L., Dmitrienko, A., Fischer, T. and Sadeghi, A. (2011). XManDroid: A New Android Evolution to Mitigate Privilege Escalation Attacks. *Technische Universität Darmstadt, Technical Report*.

Bugiel, S., Davi, L., Dmitrienko, A., Fischer, T., Sadeghi, A. and Shastry, B. (2012). Towards Taming Privilege-Escalation Attacks on Android. In: *Proceedings of the 19th Annual Network & Distributed System Security Symposium*.

Burguera, I., Zurutuza, U. and Nadjm-Tehrani, S. (2011). Crowdroid: Behavior-based malware detection system for android. In: *Proceedings of the ACM Conference on Computer and Communications Security*.

Cai, H., Meng, N., Ryder, B. and Yao, D. (2019). DroidCat: Effective android malware detection and categorization via app-level profiling. *IEEE Transactions on Information Forensics and Security*, 14.

Cai, L., Li, Y. and Xiong, Z. (2021). JOWMDroid: Android malware detection based on feature weighting with joint optimization of weight-mapping and classifier parameters. *Computers and Security*, 100.

Canfora, G., Mercaldo, F., Medvet, E. and Visaggio, C. A. (2015). Detecting Android malware using sequences of system calls. In: *3rd International Workshop on Software Development Lifecycle for Mobile, DeMobile 2015 - Proceedings*.

CheetahMobile (2014). *2014 Half Year Security Report* (on-line). https://www.cmcm.com/blog/2014-07-18/186.html. Accessed 5 March 2023.

Chen, J., Alalfi, M. H., Dean, T. R. and Zou, Y. (2015). Detecting Android Malware Using Clone Detection. *Journal of Computer Science and Technology*, 30.

Chen, T., Mao, Q., Yang, Y., Lv, M. and Zhu, J. (2018). TinyDroid: A lightweight and efficient model for android malware detection and classification. *Mobile Information Systems*, 2018.

Chin, E., Felt, A. P., Greenwood, K. and Wagner, D. (2011). Analyzing inter-application communication in Android. In: *MobiSys'11 - Compilation Proceedings of the 9th International Conference on Mobile Systems, Applications and Services and Co-located Workshops*.

Crowdstrike (2022). *What is a Trojan Horse? (Trojan Malware)* (on-line). https://www.crowdstrike.com/cybersecurity-101/malware/trojans/. Accessed 17 February 2023.

D'Angelo, G., Palmieri, F. and Robustelli, A. (2022). A federated approach to Android malware classification through Perm-Maps. *Cluster Computing*, 25.

Davi, L., Dmitrienko, A., Sadeghi, A.-R. and Winandy, M. (2011). Privilege Escalation Attacks on Android. In: pp.346–360.

David, O. E. and Netanyahu, N. S. (2015). DeepSign: Deep learning for automatic malware signature generation and classification. In: *Proceedings of the International Joint Conference on Neural Networks*.

Dehkordy, D. T. and Rasoolzadegan, A. (2020). DroidTKM: Detection of Trojan Families using the KNN Classifier Based on Manhattan Distance Metric. In: *2020 10h International Conference on Computer and Knowledge Engineering, ICCKE 2020*.

Demontis, A., Melis, M., Biggio, B., Maiorca, D., Arp, D., Rieck, K., Corona, I., Giacinto, G. and Roli, F. (2019). Yes, Machine Learning Can Be More Secure! A Case Study on Android Malware Detection. *IEEE Transactions on Dependable and Secure Computing*, 16.

Department of Defense (2015). Department of Defense Dictionary of Military and Associated Terms. *US Department of Defense Joint Publication*, 2001.

Dhalaria, M. and Gandotra, E. (2020). A Framework for Detection of Android Malware using Static Features. In: *2020 IEEE 17th India Council International Conference, INDICON 2020*.

Dhalaria, M. and Gandotra, E. (2021). A hybrid approach for android malware detection and family classification. *International Journal of Interactive Multimedia and Artificial Intelligence*, 6.

DIetz, M., Shekhar, S., Pisetsky, Y., Shu, A. and Wallach, D. S. (2011). Quire: Lightweight provenance for smart phone operating systems. In: *Proceedings of the 20th USENIX Security Symposium*.

Do, Q., Martini, B. and Choo, K. K. R. (2015). Exfiltrating data from Android devices. *Computers and Security*, 48.

Dobhal, D. C., Das, P. and Aswal, K. (2020). Detection of Android Adwares by using Machine Learning Algorithms. *International Journal of Engineering and Advanced Technology*, 8: 17–21.

Domingos, P. and Pazzani, M. (1997). On the Optimality of the Simple Bayesian Classifier under Zero-One Loss. *Machine Learning*, 29.

Edjlali, G., Acharya, A. and Chaudhary, V. (1998). History-based access control for mobile code. In: *Proceedings of the ACM Conference on Computer and Communications Security*.

Enck, W., Gilbert, P., Chun, B. G., Cox, L. P., Jung, J., McDaniel, P. and Sheth, A. N. (2014). Taint droid: An information flow tracking system for real-time privacy monitoring on smartphones. *Communications of the ACM*, 57.

Enck, W., Gilbert, P., Han, S., Tendulkar, V., Chun, B.-G., Cox, L. P., Jung, J., McDaniel, P. and Sheth, A. N. (2014). TaintDroid. *ACM Transactions on Computer Systems*, 32.

Enck, W., Octeau, D., McDaniel, P. and Chaudhuri, S. (2011). A study of android application security. In: *Proceedings of the 20th USENIX Security Symposium*.

Enck, W., Ongtang, M. and McDaniel, P. (2009). On lightweight mobile phone application certification. In: *Proceedings of the ACM Conference on Computer and Communications Security*.

Faruki, P., Bharmal, A., Laxmi, V., Ganmoor, V., Gaur, M. S., Conti, M. and Rajarajan, M. (2015). Android security: A survey of issues, malware penetration, and defenses. *IEEE Communications Surveys and Tutorials*, 17.

Faruki, P., Laxmi, V., Bharmal, A., Gaur, M. S. and Ganmoor, V. (2015). AndroSimilar: Robust signature for detecting variants of Android malware. *Journal of Information Security and Applications*, 22.

Fedler, R., Schütte, J. and Kulicke, M. (2013). On the effectiveness of malware protection on android. *Fraunhofer AISEC*.

Feizollah, A., Anuar, N. B., Salleh, R., Suarez-Tangil, G. and Furnell, S. (2017). AndroDialysis: Analysis of Android Intent Effectiveness in Malware Detection. *Computers and Security*, 65.

Feizollah, A., Anuar, N. B., Salleh, R. and Wahab, A. W. A. (2015). A review on feature selection in mobile malware detection. *Digital Investigation*, 13.

Felt, A., Greenwood, K. and Wagner, D. (2011). The effectiveness of application permissions. *WebApps '11: 2nd USENIX Conference on Web Application Development*.

Felt, A. P., Finifter, M., Chin, E., Hanna, S. and Wagner, D. (2011). A survey of mobile malware in the wild. In: *Proceedings of the ACM Conference on Computer and Communications Security*.

Felt, A. P., Ha, E., Egelman, S., Haney, A., Chin, E. and Wagner, D. (2012a). Android permissions. In: *Proceedings of the Eighth Symposium on Usable Privacy and Security*. New York, NY, USA: ACM. pp.1–14.

Felt, A. P., Ha, E., Egelman, S., Haney, A., Chin, E. and Wagner, D. (2012b). Android permissions: User attention, comprehension, and behavior. In: *SOUPS 2012 - Proceedings of the 8th Symposium on Usable Privacy and Security*.

Felt, A. P., Wang, H. J., Moshchuk, A., Hanna, S. and Chin, E. (2011). Permission re-delegation: Attacks and defenses. In: *Proceedings of the 20th USENIX Security Symposium*.

Feng, P., Ma, J., Sun, C., Xu, X. and Ma, Y. (2018). A novel dynamic android malware detection system with ensemble learning. *IEEE Access*, 6.

Feng, Y., Anand, S., Dillig, I. and Aiken, A. (2014). Apposcopy: Semantics-based detection of android malware through static analysis. In: *Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering*.

Fortinet (2023). *What Is a Trojan Horse Virus?* (on-line). https://www.fortinet.com/resources/cyberglossary/trojan-horse-virus. Accessed 17 February 2023.

FRAUDWATCH (2023). *What is Anti-Malware?* (on-line). https://fraudwatch.com/what-is-anti-malware-do-you-need-anti-malware-protection/. Accessed 17 February 2023.

G DATA (2019). *Mobile Malware Report - no let-up with Android malware* (on-line). https://www.gdatasoftware.com/news/2019/07/35228-mobile-malware-report-no-let-up-with-android-malware. Accessed 1 March 2023.

Gajrani, J., Agarwal, U., Laxmi, V., Bezawada, B., Gaur, M. S., Tripathi, M. and Zemmari, A. (2020). EspyDroid+: Precise reflection analysis of android apps. *Computers and Security*, 90.

Gao, H., Cheng, S. and Zhang, W. (2021). GDroid: Android malware detection and classification with graph convolutional network. *Computers and Security*, 106.

Gibler, C., Crussell, J., Erickson, J. and Chen, H. (2012). AndroidLeaks: Automatically detecting potential privacy leaks in Android applications on a large scale. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*.

'Glover, C. (2022). *Sandstrike Fake VPN is latest in wave of new Android malware, Tech Monitor* (on-line). https://techmonitor.ai/technology/cybersecurity/android-malware-sandstrike-fake-vpn. Accessed 7 February 2023.

Gonzalez, H., Stakhanova, N. and Ghorbani, A. A. (2015). Droidkin: Lightweight detection of android apps similarity. In: *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST*.

Guerra-Manzanares, A., Bahsi, H. and Nõmm, S. (2021). KronoDroid: Time-based hybrid-featured dataset for effective android malware detection and characterization. *Computers and Security*, 110.

HCRL (2018). *HCLR, Hacking and Countermeasure Research Lab* (on-line). http://ocslab.hksecurity.net. Accessed 29 October 2022.

Hei, Y., Yang, R., Peng, H., Wang, L., Xu, X., Liu, J., Liu, H., Xu, J. and Sun, L. (2021). Hawk: Rapid Android Malware Detection Through Heterogeneous Graph Attention Networks. *IEEE Transactions on Neural Networks and Learning Systems*.

Hicks, C. and Dietrich, G. (2016). An exploratory analysis in android malware trends. In: *AMCIS 2016: Surfing the IT Innovation Wave - 22nd Americas Conference on Information Systems*.

Hijawi, W., Alqatawna, J., Al-Zoubi, A. M., Hassonah, M. A. and Faris, H. (2021). Android botnet detection using machine learning models based on a comprehensive static analysis approach. *Journal of Information Security and Applications*, 58.

Hochreiter, S. and Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9.

Hojjatinia, S., Hamzenejadi, S. and Mohseni, H. (2020). Android botnet detection using convolutional neural networks. In: *2020 28th Iranian Conference on Electrical Engineering, ICEE 2020*.

Hosseini, S., Nezhad, A. E. and Seilani, H. (2022). Botnet detection using negative selection algorithm, convolution neural network and classification methods. *Evolving Systems*, 13.

Hutchinson, S., Zhou, B. and Karabiyik, U. (2019). Are We Really Protected? An Investigation into the Play Protect Service. In: *Proceedings - 2019 IEEE International Conference on Big Data, Big Data 2019*.

Ikram, M., Vallina-Rodriguez, N., Seneviratne, S., Kaafar, M. A. and Paxson, V. (2016). An analysis of the privacy and security risks of android VPN permission-enabled apps. In: *Proceedings of the ACM SIGCOMM Internet Measurement Conference, IMC*.

Jang, J. W., Kang, H., Woo, J., Mohaisen, A. and Kim, H. K. (2016). Andro-Dumpsys: Anti-malware system based on the similarity of malware creator and malware centric information. *Computers and Security*, 58.

Jang, J. W. and Kim, H. K. (2016). Function-oriented mobile malware analysis as first aid. *Mobile Information Systems*, 2016.

Jang, J. wook, Yun, J., Mohaisen, A., Woo, J. and Kim, H. K. (2016). Detecting and classifying method based on similarity matching of Android malware behavior with profile. *SpringerPlus*, 5.

Javaid, A., Rashid, I., Abbas, H. and Fugini, M. (2018). Ease or privacy? a comprehensive analysis of android embedded adware. In: *Proceedings - 2018 IEEE 27th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises, WETICE 2018*.

Joo, S. bin, Oh, S. E., Sim, T., Kim, H., Choi, C. H., Koo, H. and Mun, J. H. (2014). Prediction of gait speed from plantar pressure using artificial neural networks. *Expert Systems with Applications*, 41.

Kaggle *https://www.kaggle.com/* (on-line). https://www.kaggle.com/saeedseraj/a-dataset-for-fake-androidantimalware- detection. Accessed 2 March 2022.

Kang, B. J., Yerima, S. Y., McLaughlin, K. and Sezer, S. (2016). N-opcode analysis for android malware classification and categorization. In: *2016 International Conference on Cyber Security and Protection of Digital Services, Cyber Security 2016*.

Kang, H., Jang, J. W., Mohaisen, A. and Kim, H. K. (2015). Detecting and classifying android malware using static analysis along with creator information. *International Journal of Distributed Sensor Networks*, 2015.

Keerthi, S. S. and Gilbert, E. G. (2002). Convergence of a generalized SMO algorithm for SVM classifier design. *Machine Learning*, 46.

Keyes, D. S., Li, B., Kaur, G., Lashkari, A. H., Gagnon, F. and Massicotte, F. (2021). EntropLyzer: Android Malware Classification and Characterization Using Entropy Analysis of Dynamic Characteristics. In: *2021 Reconciling Data Analytics, Automation, Privacy, and Security: A Big Data Challenge, RDAAPS 2021*.

Khan, M. T., Snoeren, A. C., DeBlasio, J., Kanich, C., Voelker, G. M. and Vallina-Rodriguez, N. (2018). An empirical analysis of the commercial VPN ecosystem. In: *Proceedings of the ACM SIGCOMM Internet Measurement Conference, IMC*.

Khariwal, K., Singh, J. and Arora, A. (2020). IPDroid: Android malware detection using intents and permissions. In: *Proceedings of the World Conference on Smart Trends in Systems, Security and Sustainability, WS4 2020*.

Khattak, S., Javed, M., Khayam, S. A., Uzmi, Z. A. and Paxson, V. (2014). A look at the consequences of internet censorship through an ISP lens. In: *Proceedings of the ACM SIGCOMM Internet Measurement Conference, IMC*.

Kim, J., Ban, Y., Ko, E., Cho, H. and Yi, J. H. (2022). MAPAS: a practical deep learning-based android malware detection system. *International Journal of Information Security*.

Kim, J., Yoon, Y., Yi, K. and Shin, J. (2012). Scandal: Static Analyzer for Detecting Privacy Leaks in Android Applications. *IEEE Workshop on Mobile Security Technologies (MoST)*.

Kiss, N., Lalande, J. F., Leslous, M. and Viet Triem Tong, V. (2016). Kharon dataset: Android malware under a microscope. In: *2016 LASER Workshop - Learning from Authoritative Security Experiment Results*.

Kornblum, J. (2006). Identifying almost identical files using context triggered piecewise hashing. *Digital Investigation*, 3.

Korty, A., Calarco, D. and Spencer, M. (2021). Balancing risk with virtual private networking during a pandemic. *Business Horizons*, 64.

Krutz, D. E., Mirakhorli, M., Malachowsky, S. A., Ruiz, A., Peterson, J., Filipski, A. and Smith, J. (2015). A dataset of open-source android applications. In: *IEEE International Working Conference on Mining Software Repositories*.

Lashkari, A. H., Akadir, A. F., Gonzalez, H., Mbah, K. F. and Ghorbani, A. A. (2018). Towards a network-based framework for android malware detection and characterization. In: *Proceedings - 2017 15th Annual Conference on Privacy, Security and Trust, PST 2017*.

Lashkari, A. H., Kadir, A. F. A., Taheri, L. and Ghorbani, A. A. (2018). Toward Developing a Systematic Approach to Generate Benchmark Android Malware Datasets and Classification. In: *Proceedings - International Carnahan Conference on Security Technology*.

Lee, K. and Park, H. (2020). Malicious Adware Detection on Android Platform using Dynamic Random Forest. In: *Advances in Intelligent Systems and Computing*.

Lee, T. (2017). *Android Now Has 2 Billion Monthly Active Users* (on-line). https://www.ubergizmo.com/2017/05/android-2-billion-monthly-users/. Accessed 7 March 2023.

Li, J., Sun, L., Yan, Q., Li, Z., Srisa-An, W. and Ye, H. (2018). Significant Permission Identification for Machine-Learning-Based Android Malware Detection. *IEEE Transactions on Industrial Informatics*, 14.

Li, L., Allix, K., Li, D., Bartel, A., Bissyandé, T. F. and Klein, J. (2015). Potential Component Leaks in Android Apps: An Investigation into a New Feature Set for Malware Detection. In: *Proceedings - 2015 IEEE International Conference on Software Quality, Reliability and Security, QRS 2015*.

Li, L., Li, D., Bissyande, T. F., Klein, J., le Traon, Y., Lo, D. and Cavallaro, L. (2017). Understanding Android App Piggybacking: A Systematic Study of Malicious Code Grafting. *IEEE Transactions on Information Forensics and Security*, 12.

Liaw, A. and Wiener, M. (2018). Classification and Regression by randomForest, R News 2 (3), 18-22, 2002. *See Also selectFeatures, selectFeaturesSlimPLS, getClassification, readExpMat Examples# reads an expression matrix with class labels into exp_mat2## Not run: exp_mat2<-readExpMat (' golub_leukemia_data_with_classes_training. csv', TRUE)## End (Not run) tr*, 15.

Lindorfer, M., Neugschwandtner, M., Weichselbaum, L., Fratantonio, Y., Veen, V. van der and Platzer, C. (2016). ANDRUBIS - 1,000,000 Apps Later: A View on Current Android Malware Behaviors. In: *Proceedings - 3rd International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security, BADGERS 2014*.

Liu, P., Wang, W., Luo, X., Wang, H. and Liu, C. (2021). NSDroid: efficient multi-classification of android malware using neighborhood signature in local function call graphs. *International Journal of Information Security*, 20.

Liu, X. and Liu, J. (2014). A two-layered permission-based android malware detection scheme. In: *Proceedings - 2nd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering, MobileCloud 2014*.

Loorak, M. H., Fong, P. W. L. and Carpendale, S. (2014). Papilio: Visualizing android application permissions. *Computer Graphics Forum*, 33.

Lutkevich, B. (2021). *What is adware?* (on-line). https://www.techtarget.com/searchsecurity/definition/adware. Accessed 15 February 2023.

Maggi, F., Valdi, A. and Zanero, S. (2013). AndroTotal: A flexible, scalable toolbox and service for testing mobile malware detectors. In: *Proceedings of the ACM Conference on Computer and Communications Security*.

Mahdavifar, S., Alhadidi, D. and Ghorbani, A. A. (2022). Effective and Efficient Hybrid Android Malware Classification Using Pseudo-Label Stacked Auto-Encoder. *Journal of Network and Systems Management*, 30.

Mahindru, A. and Sangal, A. L. (2021). MLDroid—framework for Android malware detection using machine learning techniques. *Neural Computing and Applications*, 33.

Maier, D., Muller, T. and Protsenko, M. (2014). Divide-and-conquer: Why android malware cannot be stopped. In: *Proceedings - 9th International Conference on Availability, Reliability and Security, ARES 2014*.

Maiorca, D., Ariu, D., Corona, I., Aresu, M. and Giacinto, G. (2015). Stealth attacks: An extended insight into the obfuscation effects on Android malware. *Computers and Security*, 51.

Maslennikov, D., Aseev, E. and Gostev, A. (2010). *Kaspersky Security Bulletin 2009. Malware Evolution 2009* (on-line). https://securelist.com/kaspersky-security-bulletin-2009-malware-evolution-2009/36283/. Accessed 3 March 2023.

Mathur, A., Podila, L. M., Kulkarni, K., Niyaz, Q. and Javaid, A. Y. (2021). NATICUSdroid: A malware detection framework for Android using native and custom permissions. *Journal of Information Security and Applications*, 58.

Mayrhofer, R., Stoep, J. vander, Brubaker, C. and Kralevich, N. (2021). The Android Platform Security Model. *ACM Transactions on Privacy and Security*, 24.

McLaughlin, N., del Rincon, J. M., Kang, B. J., Yerima, S., Miller, P., Sezer, S., Safaei, Y., Trickel, E., Zhao, Z., Doupe, A. and Ahn, G. J. (2017). Deep android malware detection. In: *CODASPY 2017 - Proceedings of the 7th ACM Conference on Data and Application Security and Privacy*.

Milosevic, N., Dehghantanha, A. and Choo, K. K. R. (2017). Machine learning aided Android malware classification. *Computers and Electrical Engineering*, 61.

Mohamad Arif, J., Ab Razak, M. F., Awang, S., Tuan Mat, S. R., Ismail, N. S. N. and Firdaus, A. (2021). A static analysis approach for Android permission-based malware detection systems. *PloS one*, 16.

Mohamad Arif, J., Ab Razak, M. F., Tuan Mat, S. R., Awang, S., Ismail, N. S. N. and Firdaus, A. (2021). Android mobile malware detection using fuzzy AHP. *Journal of Information Security and Applications*, 61.

Moodi, M., Ghazvini, M. and Moodi, H. (2021). A hybrid intelligent approach to detect Android Botnet using Smart Self-Adaptive Learning-based PSO-SVM. *Knowledge-Based Systems*, 222.

Narayanan, A., Chen, L. and Chan, C. K. (2014). AdDetect: Automated detection of Android ad libraries using semantic analysis. In: *IEEE ISSNIP 2014 - 2014 IEEE 9th International Conference on Intelligent Sensors, Sensor Networks and Information Processing, Conference Proceedings*.

Ndagi, J. Y. and Alhassan, J. K. (2019). Machine learning classification algorithms for adware in android devices: A comparative evaluation and analysis. In: *2019 15th International Conference on Electronics, Computer and Computation, ICECCO 2019*.

Peiravian, N. and Zhu, X. (2013). Machine learning for Android malware detection using permission and API calls. In: *Proceedings - International Conference on Tools with Artificial Intelligence, ICTAI*.

Pendlebury, F., Pierazzi, F., Jordaney, R., Kinder, J. and Cavallaro, L. (2019). Tesseract: Eliminating experimental bias in malware classification across space and time. In: *Proceedings of the 28th USENIX Security Symposium*.

Peng, H., Gates, C., Sarma, B., Li, N., Qi, Y., Potharaju, R., Nita-Rotaru, C. and Molloy, I. (2012). Using probabilistic generative models for ranking risks of Android apps. In: *Proceedings of the ACM Conference on Computer and Communications Security*.

Penning, N., Hoffman, M., Nikolai, J. and Wang, Y. (2014). Mobile malware security challeges and cloud-based detection. In: *2014 International Conference on Collaboration Technologies and Systems, CTS 2014*.

la Polla, M., Martinelli, F. and Sgandurra, D. (2013). A survey on security for mobile devices. *IEEE Communications Surveys and Tutorials*, 15.

Portokalidis, G., Homburg, P., Anagnostakis, K. and Bos, H. (2010). Paranoid android: Versatile protection for smartphones. In: *Proceedings - Annual Computer Security Applications Conference, ACSAC*.

Potharaju, R., Newell, A., Nita-Rotaru, C. and Zhang, X. (2012). Plagiarizing smartphone applications: Attack strategies and defense techniques. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*.

Qiao, M., Sung, A. H. and Liu, Q. (2016). Merging permission and api features for android malware detection. In: *Proceedings - 2016 5th IIAI International Congress on Advanced Applied Informatics, IIAI-AAI 2016*.

Rahali, A., Lashkari, A. H., Kaur, G., Taheri, L., Gagnon, F. and Massicotte, F. (2020). DIDroid: Android malware classification and characterization using deep image learning. In: *ACM International Conference Proceeding Series*.

Rastogi, V., Chen, Y. and Enck, W. (2013). AppsPlayground: Automatic security analysis of smartphone applications. In: *CODASPY 2013 - Proceedings of the 3rd ACM Conference on Data and Application Security and Privacy*.

Rastogi, V., Qu, Z., McClurg, J., Cao, Y. and Chen, Y. (2015). Uranine: Real-time privacy leakage monitoring without system modification for android. In: *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST*.

Robinson, G. and Weir, G. R. S. (2015). Understanding android security. In: *Communications in Computer and Information Science*.

Rosen, S., Qian, Z. and Morley Mao, Z. (2013). AppProfiler: A flexible method of exposing privacy-related behavior in android applications to end users. In: *CODASPY 2013 - Proceedings of the 3rd ACM Conference on Data and Application Security and Privacy*.

Rosencrance, L. (2021). *antimalware (anti-malware)* (on-line). https://www.techtarget.com/searchsecurity/definition/antimalware. Accessed 17 February 2023.

Ross Quinlan (1993). Program for machine learning. *C.4.5*.

Roussev, V. (2010). Data fingerprinting with similarity digests. In: *IFIP Advances in Information and Communication Technology*.

Sadiq, A. S., Faris, H., Al-Zoubi, A. M., Mirjalili, S. and Ghafoor, K. Z. (2018). Fraud detection model based on multi-verse features extraction approach for smart city applications. In: *Smart Cities Cybersecurity and Privacy*.

Şahin, D. Ö., Kural, O. E., Akleylek, S. and Kılıç, E. (2021). A novel permission-based Android malware detection system using feature selection based on linear regression. *Neural Computing and Applications*.

Salem, A., Banescu, S. and Pretschner, A. (2021). Maat: Automatically Analyzing VirusTotal for Accurate Labeling and Effective Malware Detection. *ACM Transactions on Privacy and Security*, 24.

Sangal, A. and Verma, H. K. (2020). A Static Feature Selection-based Android Malware Detection Using Machine Learning Techniques. In: *Proceedings - International Conference on Smart Electronics and Communication, ICOSEC 2020*.

Sanz, B., Santos, I., Laorden, C., Ugarte-Pedrero, X., Bringas, P. G. and Álvarez, G. (2013). PUMA: Permission usage to detect malware in android. In: *Advances in Intelligent Systems and Computing*.

Sanz, B., Santos, I., Ugarte-Pedrero, X., Laorden, C., Nieves, J. and Bringas, P. G. (2013). Instance-based anomaly method for android malware detection. In: *ICETE 2013 - 10th International Joint Conference on E-Business and Telecommunications; SECRYPT 2013 - 10th International Conference on Security and Cryptography, Proceedings*.

Sarma, B., Li, N., Gates, C., Potharaju, R., Nita-Rotaru, C. and Molloy, I. (2012). Android permissions: A perspective combining risks and benefits. In: *Proceedings of ACM Symposium on Access Control Models and Technologies, SACMAT*.

Sasidharan, S. K. and Thomas, C. (2021). ProDroid — An Android malware detection framework based on profile hidden Markov model. *Pervasive and Mobile Computing*, 72.

Sasnauskas, R. and Regehr, J. (2014). Intent fuzzer: Crafting intents of death. In: *WODA+PERTEA 2014: Joint 12th International Workshop on Dynamic Analysis and Workshop on Software and System Performance Testing, Debugging, and Analytics - Proceedings*.

Sato, R., Chiba, D. and Goto, S. (2013). Detecting Android Malware by Analyzing Manifest Files. *Proceedings of the Asia-Pacific Advanced Network*, 36.

Schultz, M. G., Eskin, E., Zadok, E. and Stolfo, S. J. (2001). Data mining methods for detection of new malicious executables. *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*.

Seo, S. H., Gupta, A., Sallam, A. M., Bertino, E. and Yim, K. (2014). Detecting mobile malware threats to homeland security through static analysis. *Journal of Network and Computer Applications*, 38.

Seraj, S. (2021). *Android Antimalware Dataset* (on-line). https://www.kaggle.com/datasets/saeedseraj/hamdroid. Accessed 26 February 2023.

Seraj, S. (2023a). *Android Botnet Dataset* (on-line).

Seraj, S. (2023b). *Android Malicious Adware Dataset* (on-line). https://www.kaggle.com/datasets/saeedseraj/malicious-adware-detection-in-android-using-dl. Accessed 26 February 2023.

Seraj, S. (2022a). *Android Trojan Dataset* (on-line). https://www.kaggle.com/datasets/saeedseraj/trojandroidpermissionbased-android-trojan-dataset. Accessed 26 February 2023.

Seraj, S. (2022b). *Android VPN Dataset* (on-line). https://www.kaggle.com/datasets/saeedseraj/mvdroid-a-malicious-android-vpn-detector-dataset. Accessed 26 February 2023.

ʼSeraj, S. (2022). *Kaggle* (on-line). 3. https://www.kaggle.com/datasets/saeedseraj/mvdroid-a-malicious-android-vpn-detector-dataset. Accessed 8 February 2023.

Seraj, S., Khodambashi, S., Pavlidis, M. and Polatidis, N. (2022). HamDroid: permission-based harmful android anti-malware detection using neural networks. *Neural Computing and Applications*.

ʼSeraj, S., ʼPavlidis, M. and ʼPolatidis, N. (2022). TrojanDroid: Android Malware Detection for Trojan Discovery Using Convolutional Neural Networks. In: *In Engineering Applications of Neural Networks: 23rd International Conference, EAAAI/EANN*. Chersonissos, Crete, Greece: Springer International Publishing. pp.203–2012.

Shabtai, A., Kanonov, U., Elovici, Y., Glezer, C. and Weiss, Y. (2012). ‘Andromaly’: A behavioral malware detection framework for android devices. *Journal of Intelligent Information Systems*, 38.

Shakhnarovich, G., Darrell, T. and Indyk, P. (2006). *Nearest-Neighbor Methods in Learning and Vision: Theory and Practice*.

Sheen, S., Anitha, R. and Natarajan, V. (2015). Android based malware detection using a multifeature collaborative decision fusion approach. *Neurocomputing*, 151.

Shekhar, S., Dietz, M. and Wallach, D. S. (2012). AdSplit: Separating smartphone advertising from applications. In: *Proceedings of the 21st USENIX Security Symposium*.

Sihag, V., Vardhan, M. and Singh, P. (2021a). A survey of android application and malware hardening. *Computer Science Review*, 39.

Sihag, V., Vardhan, M. and Singh, P. (2021b). BLADE: Robust malware detection against obfuscation in android. *Forensic Science International: Digital Investigation*, 38.

Statista (2023). *Market share of mobile operating systems worldwide* (on-line). https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/#:~:text=Android%20maintained%20its%20position%20as,the%20mobile%20operating%20system%20market. Accessed 1 March 2023.

Suarez-Tangil, G., Dash, S. K., Ahmadi, M., Kinder, J., Giacinto, G. and Cavallaro, L. (2017). DroidSieve: Fast and accurate classification of obfuscated android malware. In: *CODASPY 2017 - Proceedings of the 7th ACM Conference on Data and Application Security and Privacy*.

Suarez-Tangil, G., Tapiador, J. E., Peris-Lopez, P. and Blasco, J. (2014). Dendroid: A text mining approach to analyzing and classifying code structures in Android malware families. *Expert Systems with Applications*, 41.

Suarez-Tangil, G., Tapiador, J. E., Peris-Lopez, P. and Ribagorda, A. (2014). Evolution, detection and analysis of malware for smart devices. *IEEE Communications Surveys and Tutorials*, 16.

Sufatrio, Tan, D. J. J., Chua, T. W. and Thing, V. L. L. (2015). Securing android: A survey, taxonomy, and challenges. *ACM Computing Surveys*, 47.

Surendran, R., Thomas, T. and Emmanuel, S. (2020). A TAN based hybrid model for android malware detection. *Journal of Information Security and Applications*, 54.

Suresh, S., di Troia, F., Potika, K. and Stamp, M. (2019). An analysis of Android adware. *Journal of Computer Virology and Hacking Techniques*, 15.

Sylve, J., Case, A., Marziale, L. and Richard, G. G. (2012). Acquisition and analysis of volatile memory from android devices. *Digital Investigation*, 8.

Symantec (2017). *2017 Internet Security Threat Report* (on-line). https://www.symantec.com/security-center/threat-report. Accessed 29 October 2022.

Taheri, L., Kadir, A. F. A. and Lashkari, A. H. (2019). Extensible android malware detection and family classification using network-flows and API-calls. In: *Proceedings - International Carnahan Conference on Security Technology*.

Talha, K. A., Alper, D. I. and Aydin, C. (2015). APK Auditor: Permission-based Android malware detection system. *Digital Investigation*, 13.

Tam, K., Feizollah, A., Anuar, N. B., Salleh, R. and Cavallaro, L. (2017). The evolution of android malware and android analysis techniques. *ACM Computing Surveys*, 49.

Tansettanakorn, C., Thongprasit, S., Thamkongka, S. and Visoottiviseth, V. (2016). ABIS: A prototype of Android Botnet Identification System. In: *Proceedings of the 2016 5th ICT International Student Project Conference, ICT-ISPC 2016*.

thehackernews (2022). *https://thehackernews.com/2022/06/sidewinder-hackers-use-fake-android-vpn.html?&web_view=true* (on-line).

Tong, S. and Chang, E. (2001). Support vector machine active learning for image retrieval. In: *Proceedings of the ACM International Multimedia Conference and Exhibition*.

Ucci, D., Aniello, L. and Baldoni, R. (2019). Survey of machine learning techniques for malware analysis. *Computers and Security*, 81.

Ullah, S., Ahmad, T., Buriro, A., Zara, N. and Saha, S. (2022a). TrojanDetector: A Multi-Layer Hybrid Approach for Trojan Detection in Android Applications. *Applied Sciences*, 12: 10755.

Ullah, S., Ahmad, T., Buriro, A., Zara, N. and Saha, S. (2022b). TrojanDetector: A Multi-Layer Hybrid Approach for Trojan Detection in Android Applications. *Applied Sciences*, 12: 10755.

Verma, S. and Muttoo, S. K. (2016). An android malware detection framework-based on permissions and intents. *Defence Science Journal*, 66.

Vidas, T. and Christin, N. (2014). Evading android runtime analysis via sandbox detection. In: *ASIA CCS 2014 - Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security*.

Vidas, T. and Christin, N. (2013). Sweetening android lemon markets: Measuring and combating malware in application marketplaces. In: *CODASPY 2013 - Proceedings of the 3rd ACM Conference on Data and Application Security and Privacy*.

Vidas, T., Christin, N. and Cranor, L. (2011). “Curbing Android permission creep," in Proc.of the Web 2.0 Security and Privacy, May 2011. In: *In Proceeding of the Web 2.0 Security and Privacy*.

Villars, R. L., Olofson, C. W. and Eastwood, M. (2011). *Big Data: What It is and Why You Should Care*.

Vinod, P., Zemmari, A. and Conti, M. (2019). A machine learning based approach to detect malicious android apps using discriminant system calls. *Future Generation Computer Systems*, 94.

VirusTotal *www.virustotal.com* (on-line). www.virustotal.com. Accessed 2 March 2022.

Wagstaff, K., Cardie, C., Rogers, S. and Schrödl, S. (2001). Constrained K-means Clustering with Background Knowledge. In: *International Conference on Machine Learning ICML*.

Wang, H., Zhang, W. and He, H. (2022). You are what the permissions told me! Android malware detection based on hybrid tactics. *Journal of Information Security and Applications*, 66.

Wang, W., Wang, X., Feng, D., Liu, J., Han, Z. and Zhang, X. (2014). Exploring permission-induced risk in android applications for malicious application detection. *IEEE Transactions on Information Forensics and Security*, 9.

Wang, W., Zhao, M. and Wang, J. (2019). Effective android malware detection with a hybrid model based on deep autoencoder and convolutional neural network. *Journal of Ambient Intelligence and Humanized Computing*, 10.

Wangchuk, T. and Rathod, D. (2021). FORENSIC AND BEHAVIOR ANALYSIS OF FREE ANDROID VPNS. *Journal of Applied Engineering, Technology and Management*, 1.

Wegman, E. J. (2003). Visual data mining. *Statistics in Medicine*, 22: 1383–1397.

Wei, F., Li, Y., Roy, S., Ou, X. and Zhou, W. (2017). Deep ground truth analysis of current android malware. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*.

Wei, X., Gomez, L., Neamtiu, I. and Faloutsos, M. (2012). Permission evolution in the Android ecosystem. In: *ACM International Conference Proceeding Series*.

Wen, L. and Yu, H. (2017). An Android malware detection system based on machine learning. In: *AIP Conference Proceedings*.

Wijesekera, P., Baokar, A., Hosseini, A., Egelman, S., Wagner, D. and Beznosov, K. (2015). Android permissions remystified: A field study on contextual integrity. In: *Proceedings of the 24th USENIX Security Symposium*.

Wikipedia (2023a). *Android* (on-line). https://en.wikipedia.org/wiki/Android. Accessed 1 March 2023.

Wikipedia (2023b). *HTC Dream* (on-line). https://en.wikipedia.org/wiki/HTC_Dream. Accessed 1 March 2023.

Wilson, J., Mcluskie, D. and Bayne, E. (2020). Investigation into the security and privacy of iOS VPN applications. In: *ACM International Conference Proceeding Series*.

Witten, I. H., Frank, E. and Hall, M. A. (2005). *Data Mining: Practical Machine Learning Tools* (4th Edition). Elsevier.

Wu, D. J., Mao, C. H., Wei, T. E., Lee, H. M. and Wu, K. P. (2012). DroidMat: Android malware detection through manifest and API calls tracing. In: *Proceedings of the 2012 7th Asia Joint Conference on Information Security, AsiaJCIS 2012*.

Wu, L., Grace, M., Zhou, Y., Wu, C. and Jiang, X. (2013). The impact of vendor customizations on Android security. In: *Proceedings of the ACM Conference on Computer and Communications Security*.

Xiao, X., Zhang, S., Mercaldo, F., Hu, G. and Sangaiah, A. K. (2019). Android malware detection based on system call sequences and LSTM. *Multimedia Tools and Applications*, 78.

Xie, N., Zeng, F., Qin, X., Zhang, Y., Zhou, M. and Lv, C. (2018). RepassDroid: Automatic detection of android malware based on essential permissions and semantic features of sensitive APIs. In: *Proceedings - 2018 12th International Symposium on Theoretical Aspects of Software Engineering, TASE 2018*.

’Xue, L., ’Zhou, Y., ’Chen, T., ’Luo, X. and ’Gu, G. (2017). Malton: Towards on device non-invasive mobile malware analysis for art. In: *In 26th USENIX Security Symposium (USENIX Security 17)*. Vancouver: ACM.

Yadav, P., Menon, N., Ravi, V., Vishvanathan, S. and Pham, T. D. (2022). EfficientNet convolutional neural networks-based Android malware detection. *Computers and Security*, 115.

Yan, L. K. and Yin, H. (2012). DroidScope: Seamlessly reconstructing the os and dalvik semantic views for dynamic android malware analysis. In: *Proceedings of the 21st USENIX Security Symposium*.

Yang, C., Xu, Z., Gu, G., Yegneswaran, V. and Porras, P. (2014). DroidMiner: Automated mining and characterization of fine-grained malicious behaviors in android applications. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*.

YANG, J., TANG, J., YAN, R. and XIANG, T. (2022). Android Malware Detection Method Based on Permission Complement and API Calls. *Chinese Journal of Electronics*, 31: 773–785.

Yang, Z., Yang, M., Zhang, Y., Gu, G., Ning, P. and Wang, X. S. (2013). AppIntent: Analyzing sensitive data transmission in Android for privacy leakage detection. In: *Proceedings of the ACM Conference on Computer and Communications Security*.

Ye, H., Cheng, S., Zhang, L. and Jiang, F. (2013). DroidFuzzer: Fuzzing the Android apps with intent-filter tag. In: *ACM International Conference Proceeding Series*.

Yerima, S. Y., Alzaylaee, M. K., Shajan, A. and Vinod, P. (2021). Deep learning techniques for android botnet detection. *Electronics (Switzerland)*, 10.

Yerima, S. Y. and Bashar, A. (2022). A Novel Android Botnet Detection System Using Image-Based and Manifest File Features. *Electronics (Switzerland)*, 11.

Yerima, S. Y. and Sezer, S. (2019). DroidFusion: A Novel Multilevel Classifier Fusion Approach for Android Malware Detection. *IEEE Transactions on Cybernetics*, 49.

Yerima, S. Y. and To, Y. (2022). A deep learning-enhanced botnet detection system based on Android manifest text mining. In: *2022 10th International Symposium on Digital Forensics and Security (ISDFS)*. IEEE. pp.1–6.

Yuan, Z., Lu, Y., Wang, Z. and Xue, Y. (2015). Droid-Sec: Deep learning in android malware detection. In: *Computer Communication Review*.

Yusof, M., Saudi, M. M. and Ridzuan, F. (2018). Mobile botnet classification by using hybrid analysis. *International Journal of Engineering and Technology(UAE)*, 7.

Zhang, F., Leach, K., Stavrou, A., Wang, H. and Sun, K. (2015). Using hardware features for increased debugging transparency. In: *Proceedings - IEEE Symposium on Security and Privacy*.

Zhang, N., Xue, J., Ma, Y., Zhang, R., Liang, T. and Tan, Y. an (2021). Hybrid sequence-based Android malware detection using natural language processing. *International Journal of Intelligent Systems*, 36.

Zhang, Y., Yang, M., Xu, B., Yang, Z., Gu, G., Ning, P., Wang, X. S. and Zang, B. (2013). Vetting undesirable behaviors in Android apps with permission use analysis. In: *Proceedings of the ACM Conference on Computer and Communications Security*.

Zhou, W., Zhou, Y., Jiang, X. and Ning, P. (2012). Detecting repackaged smartphone applications in third-party android marketplaces.

Zhou, Y. and Jiang, X. (2012). Dissecting Android malware: Characterization and evolution. In: *Proceedings - IEEE Symposium on Security and Privacy*.

Zhou, Y., Wang, Z., Zhou, W. and Jiang, X. (2012). Hey, You, Get Off of My Market: Detecting Malicious Apps in Official and Alternative Android Markets. In: *NDSS*.

Zonouz, S., Houmansadra, A., Berthiera, R., Borisova, N. and Sanders, W. (2013). Secloud: A cloud-based comprehensive and lightweight security solution for smartphones. *Computers and Security*, 37.

Zou, D., Wu, Y., Yang, S., Chauhan, A., Yang, W., Zhong, J., Dou, S. and Jin, H. (2021). IntDroid: Android Malware Detection Based on API Intimacy Analysis. *ACM Transactions on Software Engineering and Methodology*, 30.