UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Securing Local Network Access On Android

Diploma Thesis

Ioanna Gianni

Supervisor: Stamoulis Georgios

July 2024

UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Securing Local Network Access On Android

Diploma Thesis

Ioanna Gianni

Supervisor: Stamoulis Georgios

July 2024

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Εξασφαλίζοντας Την Ασφάλεια Πρόσβασης Σε Τοπικό Δίκτυο Στο Android

Διπλωματική Εργασία

Ιωάννα Γιάννη

Επιβλέπων: Σταμούλης Γεώργιος

Ιούλιος2024

Approved by the Examination Committee:

Supervisor **Stamoulis Georgios**
Professor, Department of Electrical and Computer Engineering, University of Thessaly

Member **Moondanos Ioannis**
Professor, Department of Electrical and Computer Engineering, University of Thessaly

Member **Mathy Vanhoef**
Professor, Department of Computer Science, KU Leuven

Acknowledgements

I would like to thank my supervisor, Professor Georgios Stamoulis, for his support throughout my academic journey and the completion of this thesis. His guidance and encouragement have been invaluable.

I am also profoundly grateful to Professor Ioannis Moodanos for his belief in my potential since the beginning and for his continuous support, which has been pivotal to my academic and personal growth.

I would like to extend my heartfelt gratitude to Professor Mathy Vanhoef for providing me with the remarkable opportunity to work under his guidance. This thesis would not have been possible without him.

I would like to express my appreciation to the PhD students, Jeroen Robben and Angelos Beitis, for their generous assistance and insightful guidance during my research. Their contributions have been significant to the success of this thesis.

Lastly, I would like to acknowledge my family for their unwavering support, enabling me to pursue my goals with confidence. To my friends and partner, thank you for making my student life unforgettable and filled with cherished memories.

DISCLAIMER ON ACADEMIC ETHICS AND INTELLECTUAL PROPERTY RIGHTS

«Being fully aware of the implications of copyright laws, I expressly state that this diploma thesis, as well as the electronic files and source codes developed or modified in the course of this thesis, are solely the product of my personal work and do not infringe any rights of intellectual property, personality and personal data of third parties, do not contain work / contributions of third parties for which the permission of the authors / beneficiaries is required and are not a product of partial or complete plagiarism, while the sources used are limited to the bibliographic references only and meet the rules of scientific citing. The points where I have used ideas, text, files and / or sources of other authors are clearly mentioned in the text with the appropriate citation and the relevant complete reference is included in the bibliographic references section. I also declare that the results of the work have not been used to obtain another degree. I fully, individually and personally undertake all legal and administrative consequences that may arise in the event that it is proven, in the course of time, that this thesis or part of it does not belong to me because it is a product of plagiarism».

The declarant

Ioanna Gianni

Diploma Thesis

Securing Local Network Access On Android

Ioanna Gianni

Abstract

Local network security has become increasingly important in the digital world in recent years. The number of gadgets we rely on for connectivity and all the smart IoT devices we have in our homes increases the risk that malicious actors will take advantage of weaknesses in our most reliable networks. Unauthorized access to local networks is a serious concern that can emerge through a variety of clever techniques that exploit protocols, web services and so on. These techniques describe how malicious actors try to break the security of a private network since the attack "comes" from the "outside". How easier would it be if an attack started from "inside"? In a space that is considered safe like the private network? That can be achieved if the malicious code, for example, was hidden inside an application downloaded from the app store. The user trusts the app since he downloaded it but also the rest of the network trusts whatever traffic comes from the "inside". This thesis is motivated by a gap in the Android ecosystem's approach to local network security. Android lacks a simple, comparable procedure to iOS, which has incorporated user permissions for apps requesting local network access. There are applications for Android that work as firewalls and give the user the power to control their applications' internet access but in those situations all of internet access is allowed or blocked. This project aims to provide an application that gives Android users more control over the security of their local networks. With the goal of keeping an eye on local network traffic in real time, this program notifies users when installed apps try to contact the network. Notifications that are easy to use enable users to decide whether to grant or restrict access, which in turn affects how secure their digital environment is.

Keywords:

Security, Local Networks, Android, Applications, Local Communication, IoT, Smart Home

Διπλωματική Εργασία

Εξασφαλίζοντας Την Ασφάλεια Πρόσβασης Σε Τοπικό Δίκτυο Στο Android

Ιωάννα Γιάννη

Περίληψη

Η ασφάλεια των τοπικών δικτύων αποκτά ολοένα και μεγαλύτερη σημασία στον ψηφιακό κόσμο τα τελευταία χρόνια. Ο αριθμός των gadgets στα οποία βασιζόμαστε για τη συνδεσιμότητά μας και όλες οι έξυπνες συσκευές IoT που έχουμε στα σπίτια μας αυξάνουν τον κίνδυνο να εκμεταλλευτούν κακόβουλοι φορείς τις αδυναμίες των πιο αξιόπιστων δικτύων μας. Η μη εξουσιοδοτημένη πρόσβαση σε τοπικά δίκτυα είναι μια σοβαρή ανησυχία που μπορεί να προκύψει μέσω μιας ποικιλίας έξυπνων τεχνικών που στοχεύουν στην εκμετάλλευση πρωτοκόλλων, υπηρεσιών ιστού κ.ο.κ. Αυτές οι τεχνικές περιγράφουν τον τρόπο με τον οποίο κακόβουλοι παράγοντες προσπαθούν να παραβιάσουν την ασφάλεια ενός ιδιωτικού δικτύου, αφού η επίθεση ”έρχεται” από το ”εξωτερικό”. Πόσο πιο εύκολο θα ήταν αν μια επίθεση ξεκινούσε από το ”εσωτερικό”; Σε έναν χώρο που θεωρείται ασφαλής όπως το ιδιωτικό δίκτυο; Αυτό μπορεί να επιτευχθεί αν κακόβουλος κώδικας, για παράδειγμα, ήταν κρυμμένος μέσα σε μια εφαρμογή που κατεβαίνει από το κατάστημα εφαρμογών. Ο χρήστης εμπιστεύεται την εφαρμογή αφού την κατέβασε, αλλά και το υπόλοιπο δίκτυο εμπιστεύεται την όποια κίνηση προέρχεται από το ”εσωτερικό”. Η παρούσα διπλωματική έχει ως κίνητρο ένα κενό στην προσέγγιση του οικοσυστήματος Android για την ασφάλεια του τοπικού δικτύου. Το Android δεν διαθέτει μια απλή, συγκρίσιμη διαδικασία με το iOS, το οποίο έχει ενσωματώσει δικαιώματα χρήστη για εφαρμογές που ζητούν πρόσβαση σε τοπικό δίκτυο. Υπάρχουν εφαρμογές για το Android που λειτουργούν ως τείχη προστασίας και δίνουν στον χρήστη τη δυνατότητα να ελέγχει την πρόσβαση των εφαρμογών του στο διαδίκτυο, αλλά σε αυτές τις περιπτώσεις επιτρέπεται ή μπλοκάρεται όλη η πρόσβαση στο διαδίκτυο. Το έργο αυτό αποσκοπεί στην παροχή μιας εφαρμογής που δίνει στους χρήστες Android μεγαλύτερο έλεγχο της ασφάλειας των τοπικών τους δικτύων. Με στόχο την παρακολούθηση της κίνησης του τοπικού δικτύου σε πραγματικό χρόνο, το πρόγραμμα αυτό ειδοποιεί τους χρήστες όταν οι εγκατεστημένες εφαρμογές προσπαθούν να επικοινωνήσουν με το δίκτυο. Οι ειδοποιήσεις που είναι εύκολες στη χρήση επιτρέπουν στους χρήστες να αποφασίσουν αν θα χορηγήσουν

ή θα περιορίσουν την πρόσβαση, γεγονός που με τη σειρά του επηρεάζει το πόσο ασφαλές είναι το ψηφιακό τους περιβάλλον.

Λέξεις-κλειδιά:

Ασφάλεια, Τοπικό Δίκτυο, Android, Εφαρμογή, Έξυπνο Σπίτι, Τοπική Επικοινωνία

Table of contents

List of figures

Abbreviations

VPN	Virtual Private Network
UDP	User Datagram Protocol
TCP	Transmission Control Protocol
XSS	Cross-Site Scripting
CSRF	Cross-Site Request Forgery
MAC	Medium Access Control
WiFi	Wireless Fidelity
SSID	Service Set IDentifier
UUID	Universally Unique IDentifier
DNS	Domain Name System
mDNS	multicast DNS
SSDP	Simple Service Discovery Protocol
UPnP	Universal Plug and Play
DHCP	Dynamic Host Configuration Protocol
SDK	Software Development Kit
LAN	Local Access Network

Chapter 1

Introduction

To secure network access, Android tries to ensure that all connections use encryption. This is based on the assumption that the network, including the first hop, may be under complete control of an adversary [11]. However, apps may be hostile themselves, and may abuse access to the user's local network to steal user information or to attack local devices. For instance, the Android app now called Trojan.AndroidOS.Switcher was caught brute-forcing the router's admin interface with as goal to change the local network's DNS server [12]. This enabled adversaries to subsequently intercept and profile most of the user's traffic. More generally, researchers have previously studied how malicious JavaScript can be used to scan and fingerprint the local network, and exploit vulnerable LAN devices[13]. Similarly, others have showed how browser-based methods can scan the local network to leak user information or perform attacks[14][15][16][17]. For instance, malicious JavaScript can discover and interact with local IoT devices that expose HTTP interfaces, potentially triggering malicious commands as well as fingerprinting the network[18].

Android apps with Internet permission can also access the local network, and hence perform similar attacks[19], but the precise security impact of this has not yet been studied. However, as reffered on [19] an Android application can perform various attacks such as: harvesting private data, making unwanted credit card charges, retrieving the location of users and so on. Interestingly, iOS 14, released in 2020, will warn users when an app is trying to access the local

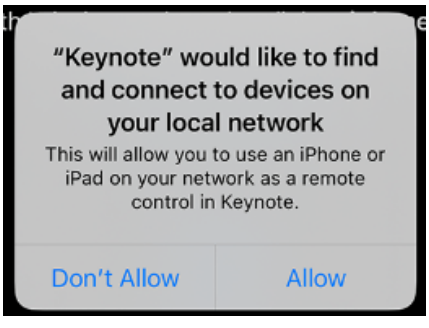

Figure 1.1: Local Network Access on iOS, Source: If an app would like to connect to devices on your local network [1]

network[1], as shown on Figure 1.1. However, it is
unclear how many mobile apps access the local network and for which reasons.

1.1 Thesis goal

This thesis studies the Android Permission Model and discusses the concerns regarding how much Local Network Security is taken into consideration.
The goal of this thesis is to create an application that reduces the hazards that come with other applications using and abusing local networks.
In order to strengthen security and user protection, the thesis suggests improvements to the Android Permission Model, particularly with regard to Internet access permissions.

1.1.1 Contribution

The contribution of the thesis can be summarized as:

1. A study was done in order to understand why there is security concern regarding the Local Network and a literature study was done on existing attacks

2. The Android Permission Model was studied and this thesis expresses the need to make network accessibility permissions more fine-grained

3. An application that aims to protect the Local Network is implemented. The application informs users of other apps that try to access the Local Network and gives the option for the user to choose how to handle that app.

1.2 Chapter Organization

The vulnerabilities of local networks are covered in Chapter 2. It is pointed out that cyber-criminals are particularly drawn to these networks. The chapter goes on to explain that devices which include poorly secured IoT devices are susceptible to assault leading data disclosure and taking control of the network without consent. Chapter 3 considers different weaknesses within a local network that can be exploited by attackers with emphasis on JavaScript role in performing these attacks.Discussed are methods inclusive of DNS rebinding, Cross-Site Scripting (XSS), and Cross-Site Request Forgery (CSRF) which shows how they can be used

for undermining the security of a network and obtaining secret data. It's the chapter that shows how an attacker can access the local network from the Internet to give an idea of how much easier it would be to have an attack happening from the internal network. Chapter 4 talk about how Android protects user's privacy and reviews the evolution of Android's permission model, describing the gradual improvements made to enhance user privacy and control. However, there are still considerable weaknesses within this system according to this chapter especially when it comes to network access permissions that continue being more relaxed if contrasted with those on iOS. The proposed solution and the application development and its functionalities are discussed in Chapter 5 where it's explained how the app monitors only the Local Network traffic coming from an Android phone and informs the user when other applications try to access it. Chapter 6 ends the thesis by summarizing the main discoveries and contributions. It elaborates more on the significance of total area security in android gadgets and gives suggestions on what can be done to the application going forward.

Chapter 2

Understanding Local Networks and Their Attractiveness to Hackers

2.1 Introduction

Organizations and individual communications rely heavily on local networks, which are vulnerable to cyberattacks due to their handling of sensitive data. Devices with different security postures, particularly IoT devices with inadequate security, are present in these networks, which leads to vulnerabilities. This chapter investigates the motivations of cybercriminals as well as the vulnerabilities present in local networks.

2.2 Definition

A local network is a collection of devices connected together in one physical location, such as a building, office, or home. They communicate with each other through common methods, often using Ethernet cables or Wi-Fi signals.

In this network community, each device is assigned a unique address, called an IP address, which fits within certain private ranges. These ranges include numbers like 10.0.0.0 to 10.255.255.255, 172.16.0.0 to 172.31.255.255, and 192.168.0.0 to 192.168.255.255. These addresses can be thought as individual house numbers; they help devices locate and communicate with each other securely within the local network, without needing to involve the larger internet. This keeps everything private and within the borders of the network.

Figure 2.1: Example of a Local Network

Local Networks are known for their ability to transfer data quickly and with minimal delay. This makes them ideal for applications that require fast communication and immediate response times. Unlike Wide Area Networks (WANs) that span large distances and connect different local networks, local networks are more intimate and compact, making them easier to manage and troubleshoot.

A typical local network setup includes essential hardware such as routers, switches, and access points. The router plays the role of the gatekeeper, managing traffic between the local network and the internet. Meanwhile, switches and access points help direct data traffic internally, ensuring everything flows smoothly within the network.

2.3 Vulnerabilities on Local Networks

Inside the simple communications and the ease of integrating IoT devices into the networked world of smart homes lies a mesh of complex security and privacy issues that lurk deep within local networks. Using extensive research that clarifies the workings and consequences of these attacks, this chapter explores the subtle weaknesses and exploitation strategies that jeopardize home networks' security. This had been supported by the paper of Aniketh Girish et al. [20], where their study described by length the threats and findings about Local Network and the users' privacy.

2.3.1 The motivation to gain access in a Local Network

At the heart of the smart home ecosystem are devices that connect across local networks These devices range from simple printers to complex voice assistants. These protocols, which are meant to make device detection and interoperability easier, unintentionally let a host of security flaws and privacy invasions go through.
The IoT devices in a home/private network are vulnerable to cyber-attacks. Several factors contribute to the vulnerability of IoT devices: Firstly, most IoT devices are built with a focus on functionality and cost-effectiveness, without security in mind. Their firmware and software can be vulnerable, there are no regular patches or some devices rely on outdated software components. Moreover, there is lack of encryption and manufactures provide these devices with weak Default settings and passwords that do not get changed by the later owners.[21] [22] As a result IoT devices are a target to attackers. By accessing the local network and subsequently those devices and the router, the attacker can take control of them, gain information on location, the model of device etc., that are helpful for selling later to advertisement companies, or just monitor all information that is exchanged in the home network. Moreover, it's easy to take control of devices like that and change settings, turn them off and on and use them as the attacker intends. [23]

2.3.2 Data harvesting on Local Network

Aniketh Girish et al. [20] expose how insecure our private networks are. As viewed on Chapter 4, Android has made changes on its Permission System in order to mitigate risks and control access to nearby devices over WiFi. These modifications, however, do not stop mobile apps from using mDNS and SSDP/UPnP to scan the local IP address space through side channels. From a local network, malicious actors can retrieve a wide range of important data, each of which has a distinct function in their toolbox for possible exploitation. The Snowden revelations showed how some ”harmless data” like device models or MACs facilitate user tracking and household profiling [24]. Take this example of ”harmless data” of the Local Network:

- **MAC Addresses**: These distinct codes are necessary for network device identification. These are used by malicious actors to follow devices over different WiFi networks and

get beyond security controls that rely on IP address variability.

- **Device Type and UUIDs**:An attacker can target a specific device for attack and continue surveillance over time by knowing the kind of equipment and its unique identification.
- **Geo-location Data**: Gives an insight on physical location of devices which can be used for harmful conduct or theft planning. Furthermore, it can help advertisers understand human behavior in terms of habits or simply know whether someone is at home or not.
- **WiFi SSID and BSSID**:Attackers may use these facts to indirectly reveal the device's location by mapping known Wi-Fi networks to geographical locate a device more precisely.
- **Device and Network Configuration Details**: Out-of-date or poorly protected components of a network may be exploited, leading to offensive actions or intrusion upon private lives with the help of personalized ads.

2.3.3 Examples

High-profile WiFi-enabled smart home devices, such as the WeMo Switch and motion sensor, baby monitoring devices [25][26], and smart light bulbs, have all been linked to reports of being vulnerable to local attacks[27]. In these cases, an adversary operating on the same local network has the ability to take control of the devices or steal private user data from them, such as live video streams. This is further supported by studies, which show that these devices have weak or nonexistent authentication measures [28][29], making them easily vulnerable to local attackers.

Chapter 3

Attacks on Local Networks

3.1 Introduction

In the previous chapter, we elaborated on vulnerabilities related to IoT devices with sensitive data within local networks. In light of these vulnerabilities, local networks become an ideal target for cybercriminals with malicious intent to exploit such devices. In this chapter, we will survey existing research that studies how an adversary can abuse access to the local network with Javascript. Initially the study is done in web browser attacks to showcase the ability for an attacker to access sensitive information from the global network from where the private network is supposed to be protected. Then narrowing down to similar attacks coming from phones which are attacks that come from the 'inside', through which local networks can be compromised, enabling the exploitation of IoT devices.

3.2 Attacks using JavaScript

JavaScript is a powerful and pervasive language for web client-side programming, offering dynamic and interactive elements to create a more interactive user interface. Conceived to increase the aesthetic and functional quality of websites, JavaScript in a short time became an instrument for cyber attackers to exploit its powers and widespread adoption for ill gains. This programming language, which is integrated into web browsers and the majority of development environments, offers many breeding grounds for the initiation of cyber attacks across various platforms.

The first use of JavaScript in such malicious activity is due to the fact that it has universal

presence inside web browsers. Since JavaScript is a client-side scripting language, it runs on the user's browser; as a result, an attacker can deliver malicious scripts directly to the individuals without having to compromise the server-side infrastructure. This direct access to the users' browsers allows a variety of exploits—from stealing cookies and capturing keystrokes to more complex attacks like Cross-Site Scripting [30].

For example, Cross-Site Scripting is the insertion of malicious JavaScript codes on web pages that are viewed by other users. These scripts, when executed, will be able to hijack user sessions, deface websites, or redirect the victim to a malicious website. This attack abuses the trust a user has in a given website and uses JavaScript as a vehicle to break it. Moreover, the ability to seamlessly interact with web content and user sessions, using JavaScript, makes it the perfect candidate for such exploits.

Even more, JavaScript's flexibility and the complexity of its ecosystems have made it possible to use for malicious purposes. Many frameworks and libraries of the language can be manipulated to create obfuscated code, hence difficult to detect and analyze, thereby going unnoticed. Attackers exploit these characteristics in order to create complex, veiled code that circumvents security measures and delivers harmful payloads with the greatest efficiency possible.

3.2.1 Examples of Web attacks using Javascript

DNS Rebinding

An attacker can't access a private network or server and make changes. The only person that is able to roam in the local network freely is a user that is part of the local network or has the permissions to access a private network/server. An attacker needs to find a way to take advantage of that information to gain access. For that they can use DNS Rebinding to abuse victims' browsers as proxies to increase the surface of the attack to private networks. DNS rebinding abuses the privilege that a domain owner has complete control of their DNS records.

To explain DNS rebinding technique the above image is used. Basically, the victim Alex and the attacker Bob that wants to gain access and extract information from Private Web Server with IP address 192.0.0.1. The private web server is accessible only by Alex. Bob creates a malicious website attack.com that hosts in the (malicious) web server with IP 5.6.7.8. Bob controls the malicious web server and the (malicious) DNS resolver with IP 1.2.3.4.

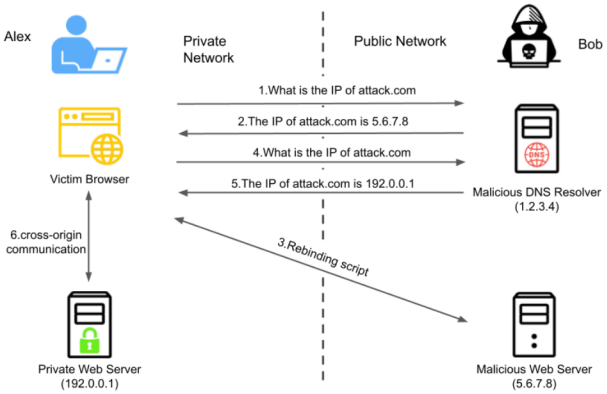

Figure 3.1: Mechanism of DNS Rebinding, Source: DNS Rebinding [2]

When attack.com needs to be translated into IP address, this DNS resolver will be reached (the nameserver record of the domain points to 1.2.3.4) and will respond with 5.6.7.8. Upon accessing attack.com, Alex's browser initiates a DNS request to Bob's resolver, which subsequently obtains the malicious server's address, 5.6.7.8. The malicious script on Bob's website tries to start a second DNS resolution for its own domain when it loads in Alex's browser. But instead, the resolver will return 192.0.0.1 this time. Attack.com is so bouncing back to the intended IP address. The malicious script can then continue to send requests to attack.com, which will ultimately lead to the private server. If the malicious website is open on the victim's browser, it can read the returned secrets and exfiltrate stolen data because Alex's browser will not identify these requests as cross-origin.

Several techniques exist that prevent DNS rebinding attacks. DNS pinning aids by saving the initial resolved IP address of a site, still it is not really useful because cached data is reset after a small time. DNSSEC (Domain Name System Security Extensions) authenticate DNS entries thus stopping tampering although it is not good in cases where the attacker owns the domain name. Firewall rules, such as those configured with iptables, can detect and block these attacks at the network entry point, though users on VPNs remain vulnerable due to encrypted DNS queries. There is also Fail-rebind, a browser plug-in designed to detect and prevent DNS rebinding attacks more robustly than other methods. [31]

Cross-Site Scripting

To defeat SOP, other ways except DNS rebinding exist, like taking advantage of Cross-Site Scripting (XSS) vulnerability or leveraging Cross-Site Request Forgery (CSRF). The XSS vulnerability can be found in web applications, and it allows attackers to inject malicious

scripts into content from, under normal circumstances, trusted websites. This altered content is then executed by the victim's browser and it can lead to stolen cookies, or other sensitive information stored by the browser. Cross-Site Scripting (XSS) can facilitate local network attacks by leveraging the victim's browser as a bridge to access internal network resources like scan the network, acees the router and so on. The reason SOP is not triggered is because SOP is designed to prevent scripts from one origin from accessing resources from another origin, but it does not protect against malicious scripts running from within the same origin.

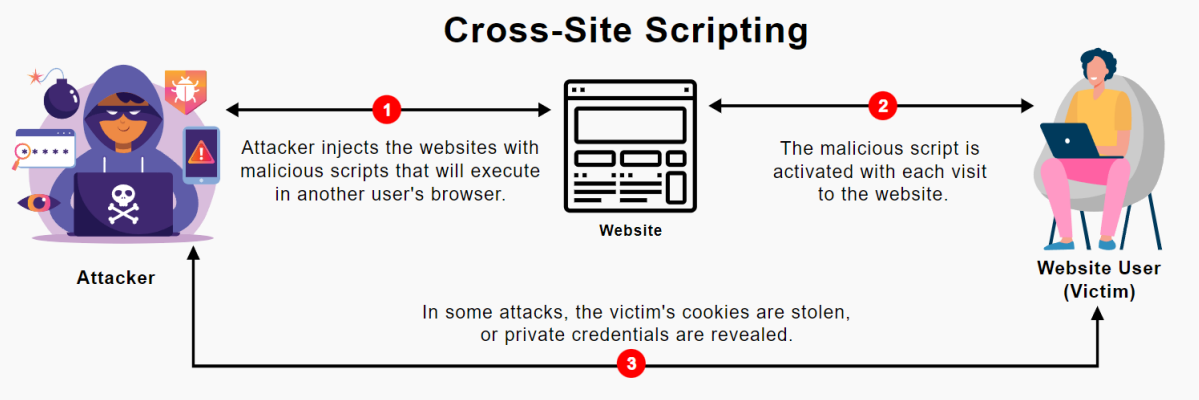

Figure 3.2: Mechanism of Cross-Site Scripting, Source: Cross-Site Scripting (XSS) Writeup [3]

To give an example, imagine a website that shows user comments without the necessary sanitization. An attacker might post a comment such as this one, which contains a malicious script:

```
<script>alert('XSS'Attack);</script>
```

Other users' browsers will run the script and display an alert box if this comment is shown to them without first filtering out the <script> tag. More maliciously, the script might compromise users' sessions by sending their cookies to the attacker.

The OWASP XSS Prevention Cheat Sheet [32] lists several means of defense that, if implemented jointly, can help in significantly reducing Cross-Site Scripting (XSS) vulnerabilities. Output Encoding is one of the most important defensive techniques since it ensures that information, which is incorporated into HTML, JavaScript, CSS or URLs, is interpreted as text instead of executable code. Moreover, when users are allowed to input HTML content it is extremely important to sanitize it. Using libraries such as DOMPurify will help in the removal of any harmful tags contained within the users provided HTML thus only safe material will be shown. Content Security Policy (CSP) offers more security by limiting the locations from which content might be loaded thereby putting up another barrier still in place

even in instances with some cross site scripting attacks. Input Validation makes certain that consumer inputs undergo good scrubbing prior application execution thus keep off harmful data.

Cross-Site Request Forgery

Cross-Site Request Forgery (CSRF) is an attack that tricks a victim to execute commands on a web application that is unauthorized [3]. That is possible because the victim is authenticated on there. It basically exploits the trust that a site has in the user's browser, making requests that appear legitimate. The most suitable example for this kind of attack is a banking application and money transfer. Think of a banking web app where users can send money by filling out a form. An email sent by the attacker to the victim that includes a link to a malicious website and an HTML form like the one below—which is hidden from the user—could result in a Cross-Site Request Forgery (CSRF) attack.

```
<form action="https://bank.com/transfer" method="POST"
style="display:none">
    <input type="hidden" name="amount" value="1000"/>
    <input type="hidden" name="account" value="attacker"/>
</form>
<script>document.forms[0].submit();</script>
```

Using the user's authentication cookies, this form would submit a transfer request to the bank's website if the user clicked the link while logged into their banking application. This request might be handled by the bank's website as a valid transaction, allowing money to be sent to the attacker's account. This occurs because of the bank's website using the user's cookies to make requests, without recognizing they were made by a third party.

One of the ways to defend against CSRF are the CSRF tokens. They function by being unique and secret generated by a server before inserting them into either a web page or an AJAX request. When a user submits the form or makes the request, the token is appended and transmitted to the server, where the system checks its validity against the stored session token. If the token is missing or incorrect, the server rejects the request, preventing unauthorized actions initiated by malicious websites. This mechanism ensures that only legitimate requests

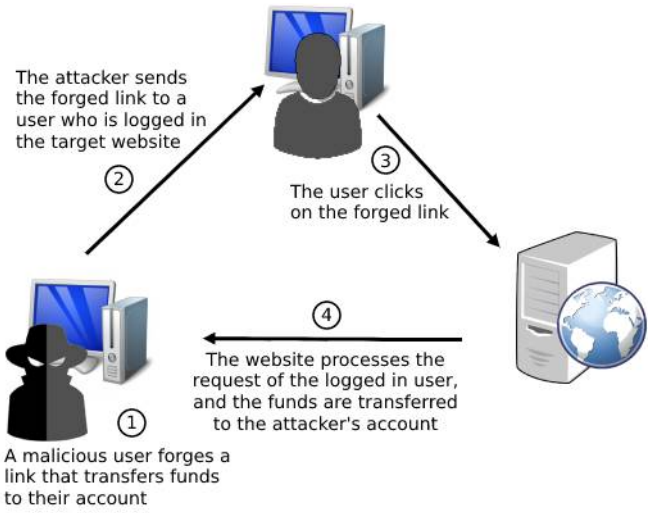

Figure 3.3: Mechanism of Cross-Site Request Forgery, Source: CSRF (Cross-Site Request Forgery) Explained [4]

from authenticated users are processed, effectively mitigating CSRF attacks .

In general, it is known that a lack of security occurs on many home and business routers. These routers often run with default settings and without updated firmware and WAPjacking and WAPkitting leverage this. The attacks exploit the fact that routers, by default, trust requests made from within the local network, and many users don't change the default admin credentials.

3.2.2 Examples of local attacks

Malicious JavaScript code can perform Local Network Attacks by exploiting vulnerabilities in web browsers and the configuration of IoT devices. A way to do that is by discovering IoT devices on a Local Network by using HTML5 MediaError interface errors [18]. Then, in order to find the surrounding active devices in the sub net, it sends GET requests to every IP address in the local sub net on a specific port (often that port is port 81 since it's not that used). The timing of each response helps to understand which IP corresponds to active devices. Lastly, to identify the device, for each IP address the script sends requests using the HTML5 "<audio>" element for device-specific endpoints that accept GET requests and based on the error message (Media Error message that comes from the fact that endpoints do not host audio resources), the script determines to which known device the IP is associated. Identifying can be enough if discovery is what the adversary aims. The attack can be continued so the attacker ex filtrates more information for profiling. This part of the attack makes

use of DNS rebinding to gain full access to the HTTP endpoints of certain IoT devices on the local network. The script manipulates the DNS resolution process to ”rebind” the domain to a local IP address associated with the targeted IoT device. This enables the script to make HTTP requests to the local IP address without triggering cross-origin errors. As a result, the attacker can interact with a device’s web interface, extract sensitive information, and even send commands to control the device. This attack bypasses the same-origin policy, which is designed to prevent web pages from accessing resources on different domains, and allows the attacker to exploit vulnerabilities or weaknesses in the IoT device’s web interface.

WAPjacking

WAPjacking [16] is the process of altering a router’s current firmware’s configuration settings for the attacker’s advantage. For instance, a hacker may alter the router’s DNS server configuration to point to a rogue DNS server under their control. This enables the attacker to secretly facilitate phishing or pharming assaults by diverting reputable online queries to phony websites. This kind of attack may be carried out swiftly and stealthily, making it challenging for users to detect, because it modifies the router’s configuration instead of its firmware.

WAPkitting

WAPkitting [16] really compromises the router’s firmware. Using this technique, the router is infected with malicious firmware, which gives it control over every data that passes through it. In addition to stealing personal data and controlling internet traffic, the attacker can even utilize the router to build a botnet. Through total control over the router, the attacker can carry out intricate man-in-the-middle assaults, intercept and alter data, or send users to dangerous websites.

3.3 Protocols that enable local attacks

3.3.1 UPnP

Device discovery on a local network can be automated with the help of Universal Plug and Play (UPnP), which enables various devices connected to the same network to recognize and

establish connections with one another automatically. While UPnP is speedy and practical, it can also expose networks to hackers and intrusions. It uses network UDP multicasts without encryption or authentication and because of its practicality this feature is implemented on home routers to ease user's experience. Since UPnP lacks authentication, one device could request port mapping for another [33] [34].

3.3.2 mDNS

IoT devices and mobile applications use mDNS to communicate with each other. This protocol is used for device discovery, service advertisement, and gathering device information. By sending mDNS queries and listening to mDNS responses, devices and apps can discover other devices on the network, extract details like device models and MAC addresses, and gather information about the services offered by the devices. This information can be used for device fingerprinting, tracking, and profiling. However, the use of unique identifiers in mDNS hostnames can inadvertently expose sensitive information and facilitate cross-device tracking and fingerprinting. [20]

3.3.3 Telnet

Telnet is a protocol that provides a command-line interface for remote device management. However, it transmits data in plaintext, making it very vulnerable to eavesdropping and man-in-the-middle attacks. Attackers can capture Telnet traffic to steal credentials, monitor communications, or inject malicious commands. Even thought it's a simple protocol to use, Telnet's lack of encryption makes it a significant security risk, especially when used on local networks where it can be easily intercepted.

3.3.4 HTTP

HTTP is widely used for web communication and data exchange: it is the language that connects web browsers to web servers so that users can view webpages. However, when the HTTP traffic isńt encrypted (i.e., using HTTP instead of HTTPS), it's susceptible to interception and modification by attackers who may inject malwares into sites, steal personal data including passwords sending them over unsecured channels and corrupt web sessions. Insecure connection methods through http remain one of the vital things used as an attack

vector even on local networks.

3.3.5 DHCP

DHCP dynamically assigns IP addresses and other network configuration details to devices. During this process, it can disclose sensitive information such as device hostnames, MAC addresses, and details about the network structure. Attackers can exploit this information to map the network, identify potential vulnerabilities, and launch further attacks. Misconfigured DHCP servers can also be used to manipulate the network configuration, potentially redirecting traffic to malicious servers.

3.3.6 ARP

ARP is fundamental to the operation of local networks, as it maps IP addresses to MAC addresses. However, it can be exploited through ARP spoofing, a technique where an attacker sends false ARP messages over the network. This allows the attacker to associate their MAC address with the IP address of another device, effectively intercepting or redirecting the network traffic meant for that device. This man-in-the-middle attack can lead to the unauthorized collection of data and potential disruption of network services.

3.4 Attacks on Local Network from applications

Naoma Blanco [35] and the Wall Street Journal [36] made a shocking discovery when they discovered a spyware SDK that could broadcast messages to fingerprint devices attached to it over the whole Local Area Network (LAN). This SDK was created to gather and transfer a variety of data to the cloud. It was allegedly made available by a US defense contractor through apps that could be found on the US Play Store. Along with numerous device identifiers and geolocation data, this data also included device clipboard contents. Surprisingly, it was discovered that the same malicious SDK was included in encryption software that had been falsely promoted by a connected business that was also a root Certificate Authority. Moreover, there have been discoveries of other applications and SDKs like **innoSDK & AppDynamics** that harvested local network information potentially for advertising and tracking reasons.

Specifically:

innoSDK: It was discovered that ten programs did NetBIOS scans; three of them additionally used ARP to get MAC addresses before sending targeted NetBIOS requests to identify NetBIOS shares on the local network. In particular, the software ”Lucky Time - Win Rewards Every Day” used a third-party library called ”innosdk” that connected to a certain endpoint in order to broadcast a UDP datagram to every IP address in the 192.168.0.0/24 subnetwork. The SDK used an algorithmic approach to create network broadcasts, most likely in an attempt to avoid being identified as malware. This app was later taken down from the Google Play Store after analysis.

AppDynamics: Version 6.18.3 of the CNN app supported casting content to smart TVs with SSDP/UPnP. This was facilitated by Cisco’s AppDynamics SDK. This SDK may monitor network and user interface events, including local network operations, like access to UPnP/SSDP XML file descriptions. It was designed for app performance statistics. As UPnP device descriptors contain MAC addresses and exact device information, this opened the way to profiling user behavior and home networks. It was discovered that the SDK collected data on device IDs, Wi-Fi SSIDs, and a list of nearby devices, which raised privacy concerns. Later iterations of the CNN app eliminated the casting feature, but AppDynamics’ request tracking features remained, and gave rise to privacy concerns.

In the paper ”Voice Hacking: Using Smartphones to Spread Ransomware to Traditional PCs,”[37] a fully worked-out cyber attack methodology is described, depicting how voice commands from Android smartphones can be used to initiate a ransomware attack on a Windows PC, as long as it is on the same local network. The attack vector has its starting point when an unsuspecting user accidentally or unintentionally initiates a voice command, instructing the smartphone to open up a malicious webpage. This webpage is supposed to detect the Android version of the device and redirects it to another page, which contains exploits tailored for specific Android versions. If this is successful, it deploys a reverse TCP payload, creating a Meterpreter session that the attackers can now leverage to control the smartphone remotely.

With the smartphone now under the control of the attacker, a local network scan is launched

for Windows PCs vulnerable to the EternalBlue exploit; the smartphone now serves as a proxy to bypass network defenses and execute the EternalBlue exploit, whereby Windows machines get compromised remotely without the need for any kind of interaction. This further allows the proliferation of ransomware, or any other kind of malware, with the smartphone serving as the gateway. This attack vector reveals serious security vulnerabilities within the Android ecosystem, especially how easily outdated or unpatched Android devices can be exploited. A majority of Android devices are vulnerable, usually running versions with missing security patches; it poses a considerable danger in local networks where such devices may unknowingly be used as a gateway for cyber attacks against other network-connected systems. It further underlines the importance and need for regular updating and stringent network security measures to counter such threats.

Another paper from Sivaraman et al. [34] talks about an application designed to demonstrate an attack on smart-home networks by bypassing home router security via a smartphone app. This application was purposefully embedded with malware and disguised as a legitimate app on the Apple AppStore. The authors chose to create an iOS application because they state that Apple operates a more secure ecosystem than Android. The malware, once installed, performs two main functions:

- Scouting for IoT Devices: The malware first scans the home network for IoT devices. It uses protocols like SSDP (Simple Service Discovery Protocol) to identify devices such as light-bulbs, webcams, power-switches, etc., within the home. This scouting is important because it allows the malware to map out the IoT landscape within the home, which could not have been done from outside the network.

- Allowing External Access and Control: After the identification of the devices, the malware communicates with the home gateway—say, a router—to change its firewall settings. It uses UPnP—Universal Plug and Play—commands in order to create port mappings that allow external Internet traffic to be forwarded directly to the specified IoT devices within the home. This effectively exposes the devices to remote attacks.

The malicious app showed how easily a home network that seems secure could be compromised. It highlighted the need for stronger security measures in IoT devices and routers, as

well as the vulnerabilities in relying solely on perimeter security like that offered by standard home routers.

This research team used an iPhone survey app that would seem quite harmless, and they would hide the malware behind this app. It would only fire up its malicious functions if some criteria were met, so the chances of it being detected by the app review process were minimal. By taking advantage of this setup, the researchers were able to modify firewall configurations undetected and take control of IoT devices from a remote location.

The demonstration raised serious security implications in which even devices perceived to be secure behind NAT and firewalls could be compromised in sophisticated attacks to scout, manipulate, and control IoT devices inside a smart home. In this way, it is a very scalable approach to attack hundreds of thousands of homes, potentially leading to a large-scale event.

It is expressed that a security solutions that analyze network traffic to deduce illegitimate access should be investigated. The paper was written in 2016 and in 2020 with the release of iOS 14, Apple gave a solution like that as referred on chapter 1, thus Android, which was described as less secure ecosystem than iOS, hasn’t.

Chapter 4

Android Permission Model

4.1 Introduction

Regarding Android, its defense mechanism in order to protect the system's integrity, user's privacy and mitigate risks like the ones refereed on the previous chapters, is the Android Permission Model. In this chapter we are going to deep dive into the types of permissions, how they work and their evolution. We conclude on the differences between iOS and Android regarding protection on the local network.

4.2 Android Permissions

Android permissions are declarations that indicate the degree of access to data, system resources, or other applications' functionality that an application needs [6]. They are necessary to guarantee that apps run within a secure perimeter and to preserve user privacy. Developers ask for permissions in the manifest file of the program as shown on figure 4.1.

```
<manifest ... >
  <!-- Required only when requesting background location access on
       Android 10 (API level 29) and higher. -->
  <uses-permission android:name="android.permission.ACCESS_BACKGROUND_LOCATION" />
</manifest>
```

Figure 4.1: Declaring a background permission on the app's Manifest file, Source: Request Runtime Permissions [5]

There are three types of permissions: **Install time permissions**, **Runtime Permissions**

and **special permissions**.

4.2.1 Install time permissions

Install time permissions is the type of permission that is automatically granted to an app upon installation. If the user Accepts the permissions, the app is installed, else the app installation is canceled

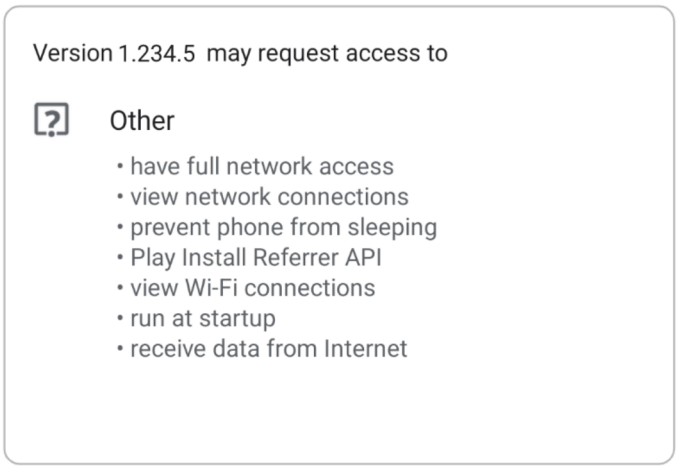

Figure 4.2: Install Time permissions as seen on Play store, Source: Permissions on Android [6]

There are two sub-categories of Install time permissions: **Normal Permissions** and **Signature Permissions**. Both types of permissions are install-time permissions because they either pose minimal risk (normal permissions) or are restricted to trusted apps (signature permissions).

Normal Permissions: There is little danger to the user's privacy or the operation of the device with these permissions. Although they are required for the fundamental operation of apps, they do not grant access to private data or device control. They are allowed during installation since they pose no danger and make the user experience easier.

As examples, consider:

- Internet access is required in order to use online functions.
- Vibrate: Required for alerts.

Signature Permissions: These are more advanced permissions that have the ability to modify system configurations or access private information. But only apps that have been signed with the same certificate as the system or another app that has the same certificate

may use them. Due of the developer's thorough screening, they can be trusted. They are also provided automatically during installation because they are intrinsically secure.

As examples, consider:

- Write System Settings: Permits the system to be changed.
- Install Packages: Enables the installation of updates or additional software.

4.2.2 Runtime Permissions

Runtime permissions are the ones that an application asks for during its operation as opposed to when it is installed. Since they can choose to grant or refuse particular rights only when they are truly required by the app, users are given greater control over the operation of their devices and personal data. This method improves security and privacy for users.

Examples of runtime permisions

- Access Location: Allows an app to access the device's location.
  - Example: A mapping app asks for location permission when it needs to provide directions.
- Camera: Allows an app to access the device's camera to take photos or videos.
  - Example: A social media app requests camera access when the user wants to take a picture.
- Contacts: Allows an app to access the device's contacts.
  - Example: A messaging app asks for contacts permission to show friends who are also using the app.
- Storage: Allows an app to read from or write to the device's external storage.
  - Example: A photo editing app requests storage permission to save edited photos.

4.2.3 Special Permissions

The type of **Special permissions** allows apps access to extra-sensitive functions or features that have the potential to have a big impact on the user's device or data. Because these

rights are so powerful, they require explicit user approval, which is typically obtained through the device's settings.

Example of special permission

- Draw Over Other Apps: Allows an app to display content over other apps. This can be used for features like chat heads like the Messenger one.
  - Use Case: A messaging app using chat heads to allow users to read and reply to messages from any screen.
  - Risk: Potential misuse for malicious overlays that could trick users into providing sensitive information.

4.3 Evolution of Android Permission Model

The Android permission paradigm has changed significantly over time to accommodate new security risks, improvements in technology, and user privacy concerns:

1. **First Model** (Up to Android 6.0). Install Time permissions. Users' ability to regulate what applications may access was limited because they had to accept all permissions before downloading an app. If the user did not accept any of the permissions required, it was impossible to install the app in whole without consent.

2. **Model of Runtime Permissions**(Android 6.0 and Up): With the release of Android 6.0 (Marshmallow), this architecture, which gave users the ability to approve or reject permissions at runtime, represented a paradigm shift. This method improved app privacy and gave users more control over their data.

3. **"Permission Auto-Reset," and "One Time Permissions"**: Android 11 added these functionalities which automatically removes permissions from apps that haven't been used for a long time. These functions protect user data from unauthorized access even further.

4.3.1 The example of Location permission evolution

Location permissions have come a long way since the early days of the Android platform, and today, they play a critical role in providing users with a tailored and personalized mobile

experience. Android has made significant changes to its location permission model to enhance user privacy, security, and overall user experience.

Initial Implementation (Android 1.0 - 5.0) In the early versions of Android, the Location permission was relatively straightforward:

- Apps requested access to coarse (network-based) or fine (GPS-based) location at install time.
- Users granted these permissions during app installation without the ability to manage them dynamically.

Introduction of Runtime Permissions (Android 6.0 - 8.1)

With Android 6.0 (Marshmallow), Google introduced a new permissions model:

- Permissions were divided into “normal” and “dangerous” categories.
- Location permission was classified as dangerous, requiring explicit user approval at runtime.
- Users could grant or deny location access when an app first requested it, and they could also revoke the permission later via settings.

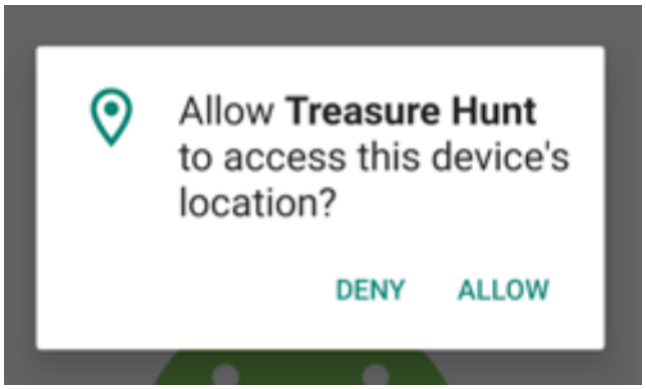

Figure 4.3: Initial Runtime Permission before Android 10 for Location, Source: Evolution of Location Permission in Android [7]

Background Location Access (Android 10)

Android 10 introduced further refinements to Location permissions:

- Users were given more granular control with the option to allow location access only while the app is in use (foreground location).
- Background location access required a separate permission, ensuring users were explicitly aware when apps accessed location data in the background.

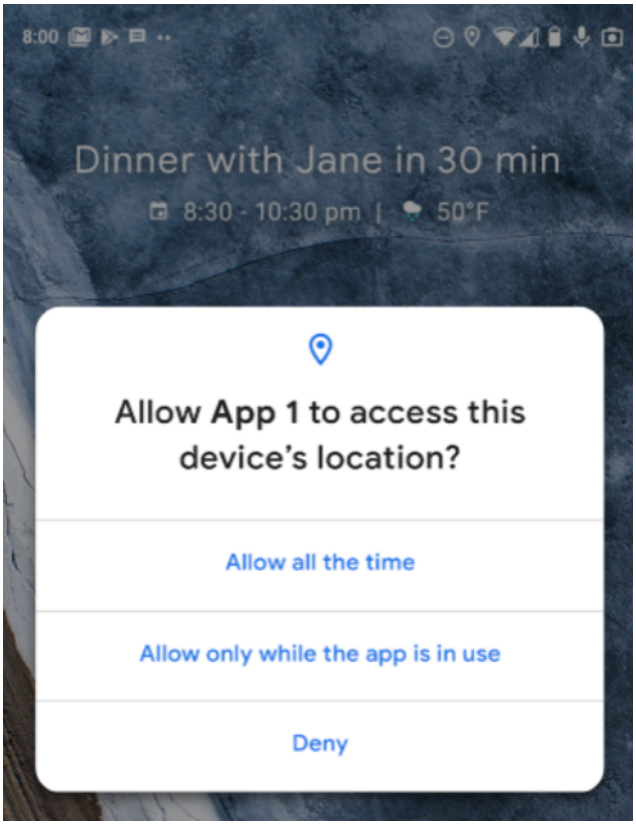

Figure 4.4: Tristate permission on Android 10 for LocationSource: Tristaste location permissions [8]

Enhanced Controls and Transparency (Android 11) In Android 11, Google continued to enhance user control and transparency:

- One-time permissions were introduced, allowing users to grant temporary access to location data for a single session.
- Apps needed to justify the need for background location access, often requiring users to go to settings to enable it.

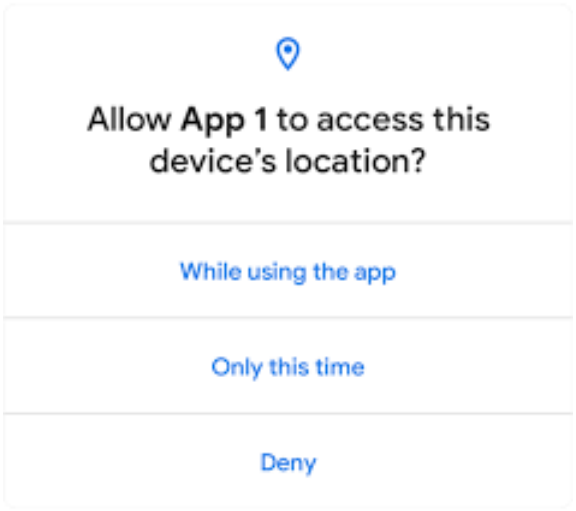

Figure 4.5: One time permission for accessing location that was introduced on Android 11, Source: Permissions updates in Android 11 [9]

Increased Privacy Measures (Android 12)

Android 12 further strengthened location privacy:

- Users could opt to provide approximate location instead of precise location, giving more control over their privacy.
- Additional indicators and notifications were added to inform users when an app was accessing location data.

The development of Android's Location permission is indicative of a larger movement toward increased user autonomy and openness. Android has dramatically improved user privacy while maintaining the ability to use important app functionality. This has been achieved by gradually providing more granular permissions and increasing transparency into the usage of location data.

4.4 Regarding accessing the network

Android manages internet access for applications through a set of network access permissions. This set of permissions of network access is normally managed by the user, trying to find a balance between functionality and security. Normal permissions that are allowed in this group include 'android.permission.INTERNET'[38], which allows apps to open network

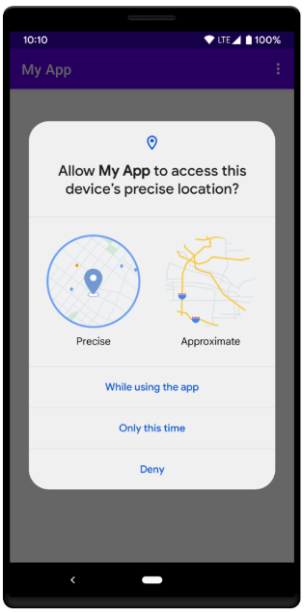

Figure 4.6: Location Accuracy permission on Android 12+, Source: Request Location Permissions [10]

sockets and connect to the internet. Because it remains categorized under ”normal” permissions, it is by default guaranteed at install time because it is of low risk to the privacy of the user.

For connectivity, another permission used is ACCESS_NETWORK_STATE [39]. It provides application access details related to network connections like Wi-Fi availability and mobile data availability, and what type of network it is. It allows access to details regarding Wi-Fi networking: its connection state and configured networks.
For example, CHANGE_NETWORK_STATE [40] and CHANGE_WIFI_STATE [41] are some permissions which allow apps to change the settings for network connectivity, that is, enabling or disabling settings for network connectivity or Wi-Fi-enabled settings.

These permissions are also granted against the ”normal” category and granted by default, supporting the applications to manage their network usage well and further minimizing the interruptions. In this manner, Android rationalizes these permissions, ensuring that apps really do have network capabilities for proper functioning in a secure way.

4.4.1 Permissions for local network access

As discussed on chapter 2, malicious actors try to exploit the local network to extract sensitive information and there have been occasions where malicious applications have been

caught extracting information. In order to mitigate those risks, Android and iOS make changes continuously in their permission models. To control access to nearby devices over WiFi, since Android 13, developers must request access to the NEARBY_WIFI_DEVICES [42] permission to read the SSID of the WiFi AP. From Android 9 to Android 13, SSID/BSSID access required either the ACCESS_FINE_LOCATION runtime permission [42].

However, from the study conducted by Girish et al. [20], these Android permission model changes do not prevent mobile apps from scanning the local IP address space using mDNS and SSDP/UPnP, for which Android even provides native support through the Nsd- Manager (Network Service Discovery Manager) interface [43].

Local network traffic on iOS requires developers to obtain the com.apple.developer.networking.multicast entitlement, whose use requires Apple's express approval, and include the NSLocalNetworkUsageDescription in the app manifest in order to communicate with other hosts in the local network (either through unicast or multicast connections); this permission also requires the explicit consent of the user [20]. Moreover, in order to enhance control over the local network access and privacy, from iOS 14 onwards, users receive notifications whenever an application wants to access the local network giving the opportunity to the users to Block or Allow that network traffic [1].

In conclusion, this chapter outlines the evolution of the Android permission model, highlighting its progression in enhancing data protection and mitigating risks. It is evident that while there has been considerable development in controlling access to location data, the permissions related to internet, and as an extension local network, access have not evolved similarly. Compared to the more user-centric updates seen in iOS, Android's approach to network-related permissions appears somewhat less developed. This observation underscores a critical area where Android could further refine its permission model to better align with user expectations and contemporary security needs.

Chapter 5

Application Development for managing Local Network Access on Android

5.1 Introduction

The Internet of Things has been a great advancement regarding our ability to control and interact with devices in our daily life. Chapter 4 gives us an overview on how the Android permission has advanced over the years to enhance protection and avoid information leakage regarding networks. The example of the Location Permission is given to visualize this development. However, it's visible that there are more things that need to be done regarding the Network Permission since bypasses occur as explained on Chapter 3 even with the current updates. Given the presence of such delicate and sensitive data in the private network, the need for further protection is visible. Currently, users have limited control over their home networks, making it difficult to counteract such malicious activities. It would be worthwhile investing in security solutions that analyze network traffic to deduce illegitimate access [34]. This gap the application LocalVPN tries to close, designed to empower users with the ability to monitor and manage their home network usage more effectively.

5.2 Requirement Analysis

In order to monitor local network traffic and notify users in real time when third-party programs try to access the local network, LocalVPN was created. Giving users control over their network security, it allows them to accept or reject these attempts at access.

Functional Requirements

1. Network Traffic Monitoring:
    - Continuously monitor all outgoing network traffic within the local network.
    - Detect attempts by any installed third-party application to access the local network.
2. User Notification System:
    - Generate real-time notifications for the user when a third-party application attempts to access the local network for the first time.
    - Each notification must provide information about the accessing application, including the application name and the type of access attempted.
3. Decision Making Mechanism:
    - Include ”Allow” and ”Block” options within each notification, enabling users to make immediate decisions regarding network access for the third-party application.
    - Implement a default action for scenarios where the user does not respond to the notification within a specified time frame.
4. Any app that attempts to explore your local network for the first time and wants to interact with devices on it has to request permission via the notification that the LocalVPN app will send.
5. The application needs to have the right permissions in the Manifest file.
6. Any application that is excluded from accessing the Local Network needs to still be able to function as intended when it tries to access the public network

5.3 Essential Concepts Before Development

5.3.1 Android Studio

The implementation of the LocalVPN application was done using Android Studio [44]. Android Studio is an integrated development environment (IDE) provided by Google, for developing Android applications. It is a powerful tool that offers many possibilities from code development to simulation of application in a real environment. Through Android Studio the developer can design the user interface, program its functionality, perform the necessary checks and debug when problems are detected in the operation of the application.

Testing on Virtual Device

Testing on virtual devices helps developers validate applications quickly and effectively across a variety of hardware and software environments. App quality and development speed are enhanced by its support for automated and thorough testing scenarios.

During the testing of the application two virtual devices where used:

- Pixel 2 API 30
- Pixel 4 API 34

Testing on Physical Device

Testing an application on a physical device is important in order to experience the app in a real-world scenario, interact with actual device hardware and figure out if the application works as intended. Moreover, this way the developer understands better the application's performance and usability from the user's perspective. To configure a physical Android device as a device that can be used on Android Studio to test an application there needs to be changes on the physical devices settings to enable Developer Options and USB debugging.

The physical Device used was: **Redmi Note 8T**

5.3.2 Android's VPN Service

Android provides APIs for developers to create virtual private network (VPN) solutions. It provides developers with the ability to programmatically configure and manage network traffic, thus ensuring that data exchanged over open networks are secure. VpnService is a base class for applications to extend and build their own VPN solutions. In general, it creates a virtual network interface, configures addresses and routing rules, and returns a file descriptor to the application. Through the VpnService class, an application can create a network interface that can encrypt, encapsulate, and send traffic through such a network interface without having to gain root access on the device. In the former case, this capability allows a user to access networks otherwise unavailable to others, and in other cases, one can go ahead to encrypt traffic, stop data leakage, and work on enhancing privacy and security. Instead of exposing your application directly to a VPN server, your VPN service allows your application to locally intercept and handle data on the device before forwarding to a remote VPN server, or it can also just locally monitor and take control over network traffic. [45]

Android shows the following UI components for VPN connections:

- Before a VPN app can become active for the first time, the system displays a connection request dialog. The dialog prompts the person using the device to confirm that they trust the VPN and accept the request.
- The status bar includes a VPN (key) icon to indicate an active connection.
- The VPN settings screen (Settings > Network & Internet > VPN) shows the VPN apps where a person accepted connection requests. There's a button to configure system options or forget the VPN.
- The Quick Settings tray shows an information panel when a connection is active. Tapping the label displays a dialog with more information and a link to Settings.

5.4 Implementation

The only way to build a no-root firewall on Android is to use the Android VPN service. Thus as a foundation for the application we used an already existing project from Github [46] that served as the code of building a VPN service which monitors TCP and UDP traffic and monitors network traffic on Android devices. We adapted and expanded the project's capabilities to address the specific needs of monitoring and blocking only Local Network traffic. Adaptations meant extending the algorithm in order to grab the traffic meant for private IP addresses and recognising each third party application that tries to access the Local Network and informs accordingly. Moreover, to enhance user awareness and control, we added a notification system so that the user is warned each time an unauthorized application attempts to access the Local network. The user then has the option to ”Allow” or ”Block” that application from accessing the local network.

Android's VPN Service usage

To capture traffic, Android's VPN Service class is used. For the development of our application, the only traffic that is rerouted through the virtual TUN interface is the traffic that targets the Local (or Private) network which means outgoing packets that have as a destination IP address an address that belong to one of these ranges:

- Class A: 10.0.0.0 to 10.255.255.255
- Class B: 172.16.0.0 to 172.31.255.255
- Class C: 192.168.0.0 to 192.168.255.255

Those IP ranges were specified while building the VPN Service.

Notification system

Each time a network packet with destination address belonging to the above ranged is captured, a time sensitive notification [47] is sent to the user containing the name of the unauthorized application that tries to access the Local Network and two action buttons [48] that give the option to block or allow access.

We used app package name (e.g.,com.xiaomi.smarthome) as the unique identifier to get the application name shown in the notification and to make sure that we don't get overlapped apps from different sources.

In case of the **Allow** button pressed, the application name is getting into the list with the "allowed applications" so that all traffic that belongs to that application is allowed to access the Local Network.

In case of the **Block** button pressed, the application name is getting into the list with the "blocked applications" so that all traffic that belongs to that application is dropped.

5.5 LocalVPN's usage and User Interface

When launching the application, the user sees Figure 5.1.As referred above, before LocalVPN app can become active for the first time, the system displays a request dialog as shown on image5.2.

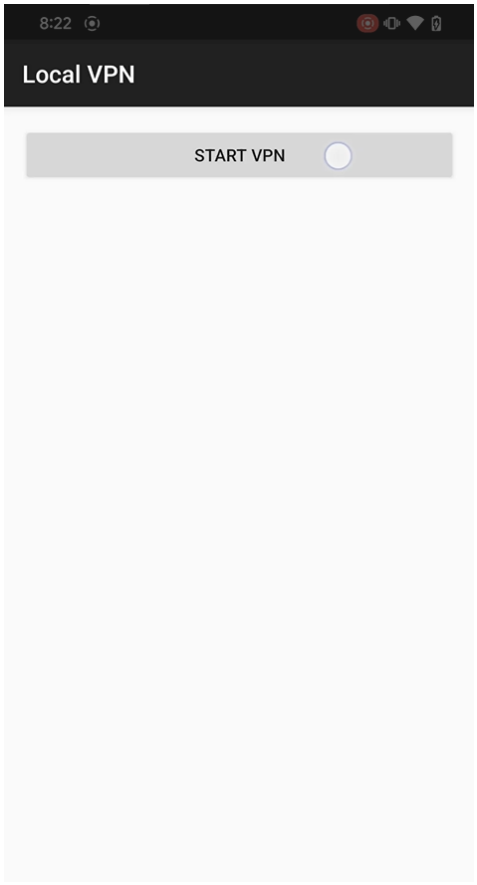

Figure 5.1: Starting the application

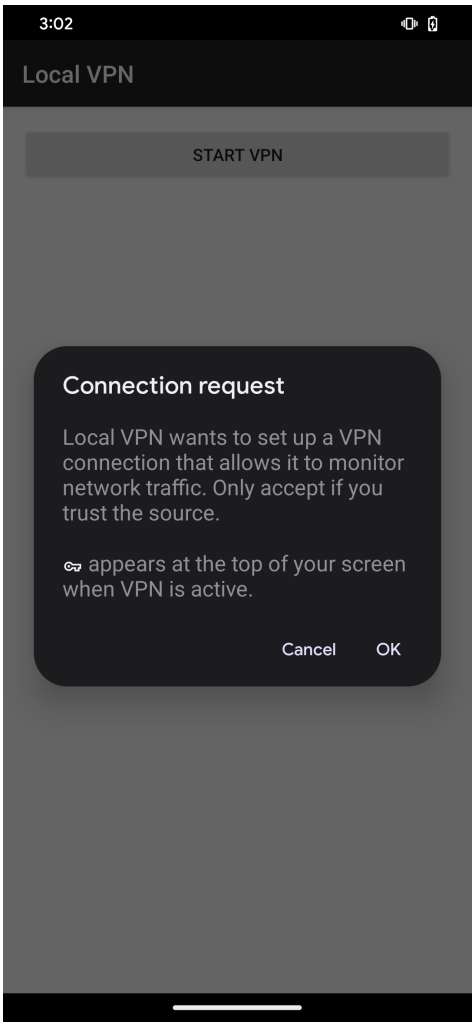

Figure 5.2: When LocalVPN needs to start for the very first time

After activation a key is appearing on the notification bar letting the user know that a VPN is running as shown on 5.3

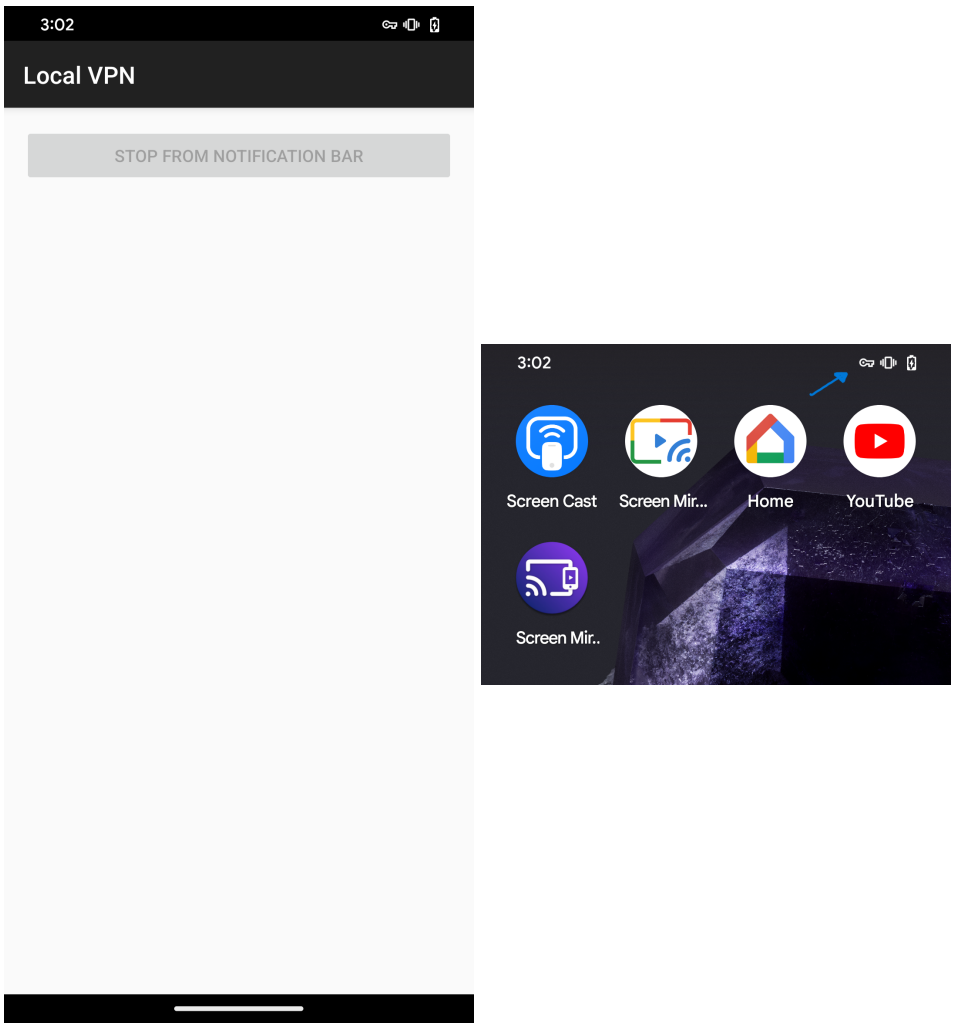

Figure 5.3: Key icon appearing on notification bar after VPN activation

After enabling the VPN application, the user can navigate in the settings page to see the status of it or disable it as show in Figure 5.4

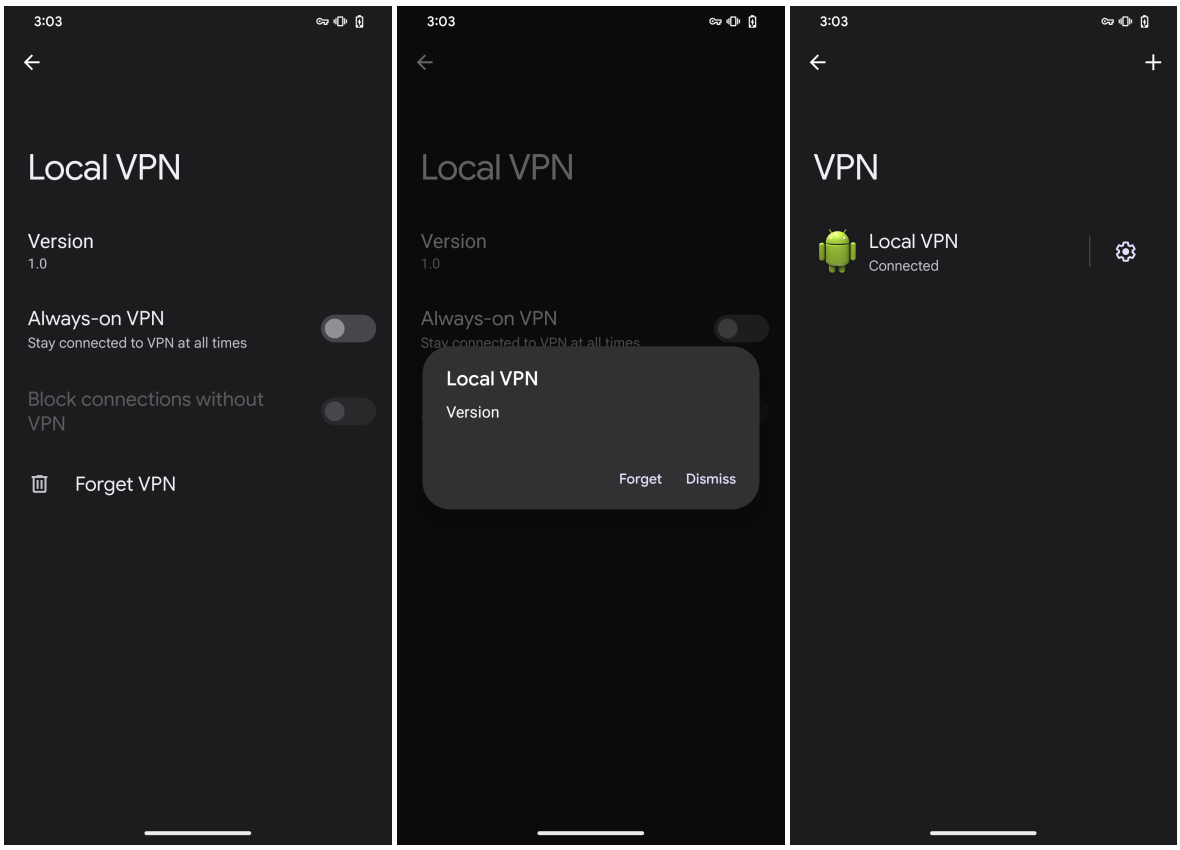

Figure 5.4: Settings Page For VPN on Android

To test the functionality of the application we tried to access the home router via Chrome. So, we have Chrome, a 3rd party application trying to access the Local network as shown on Figure 5.5

Right after hitting the Enter button, we get a notification informing us that Chrome tries to access the Local Network with the options to Block or Allow this action as shown on Figure 5.6

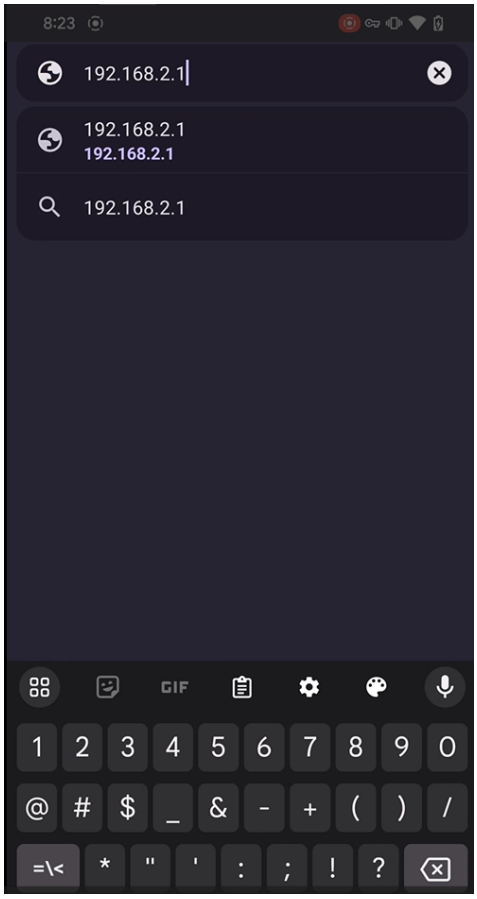

Figure 5.5: The router's IP address on Chrome search

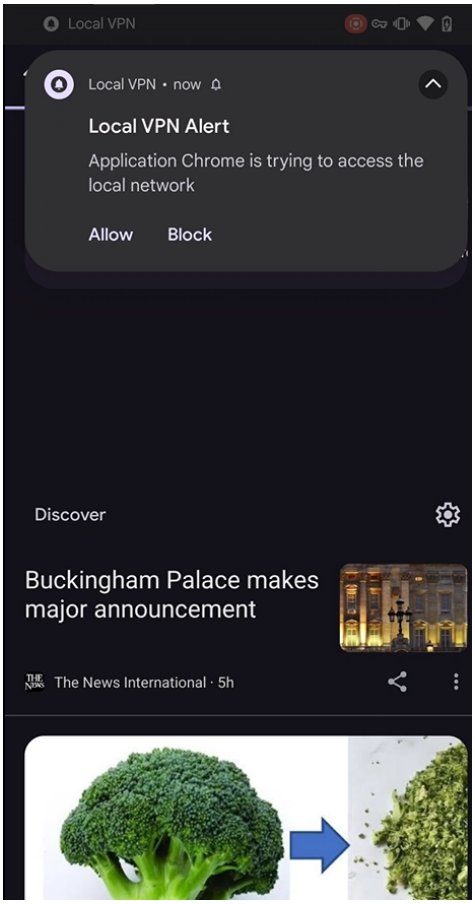

Figure 5.6: Notification when Chrome tries to access the home's router

If we press the Allow button, the action goes through and the log in page of the router is displayed as shown on Figure 5.7 If we press the Block button the connection is dropped and we cannot access the log in page of the router as shown on Figure 5.8

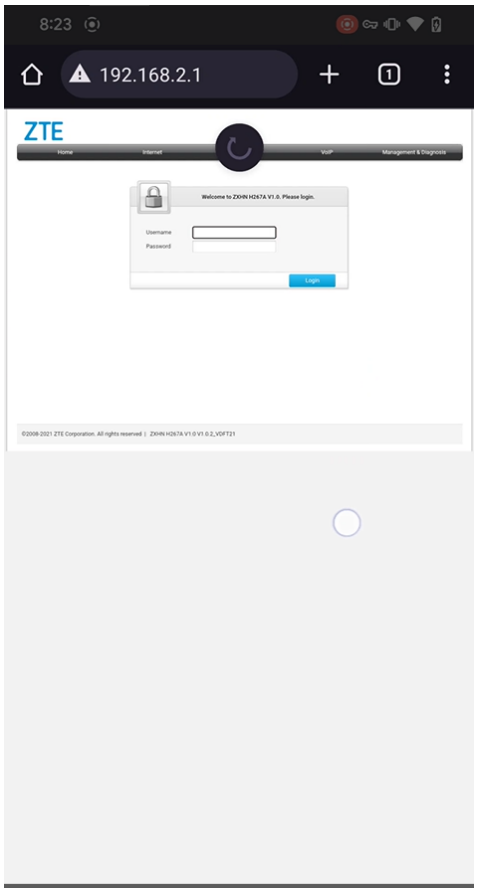

Figure 5.7: On Allow the router's page appears as intended

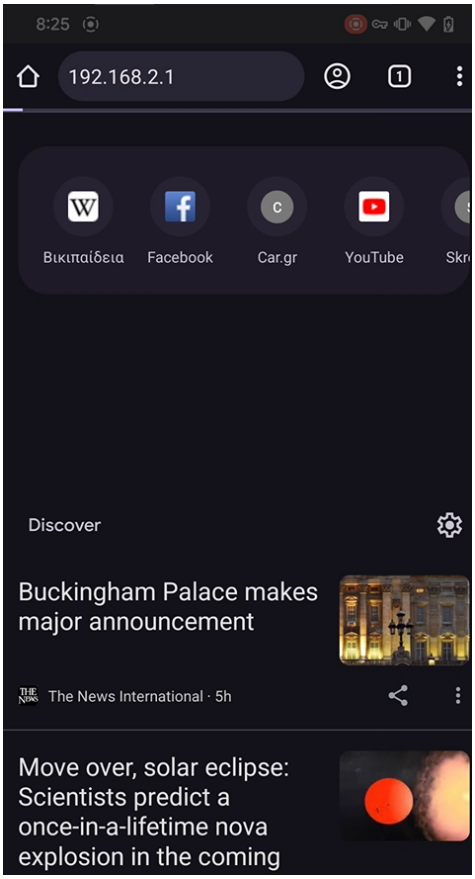

Figure 5.8: On Block, there is no action happening

However, when the block button is pressed, the rest of the network traffic coming from Chrome towards the Internet, is not blocked and the application is functioning as intended. For example searching for YouTube 5.9 or accessing Wikipedia 5.10

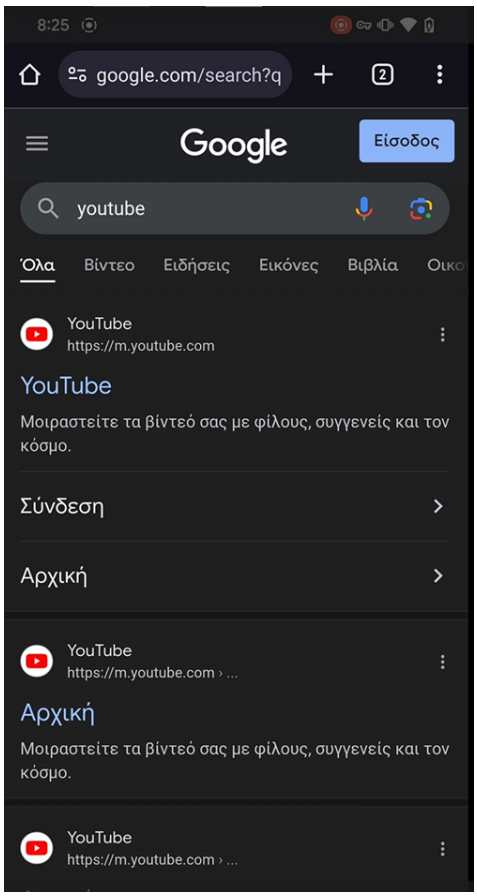

Figure 5.9: Searching for YouTube in Chrome after Block button is pressed

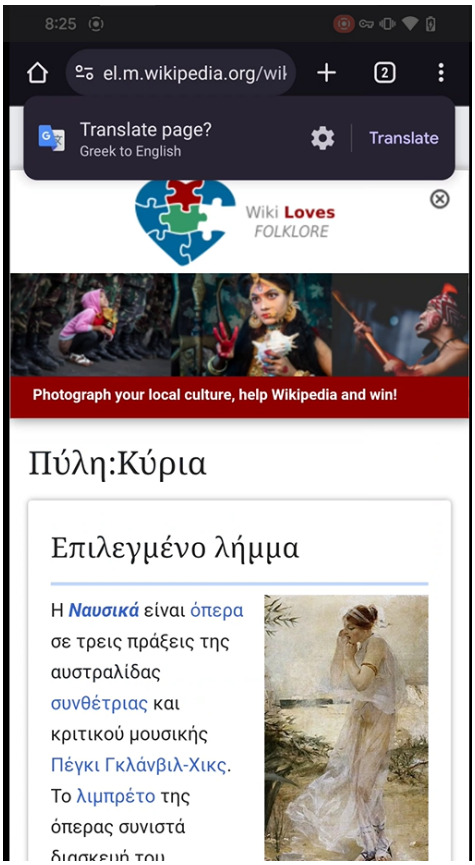

Figure 5.10: Accessing Wikipedia via Chrome after Block button is pressed

5.6 Further Findings

Another application that tried to access the local network without a particular reason was the G4 Connect [49].

The application was observed to try to access the local network even before Logging in with an account, just by opening the application. Specifically, the app looks and finds the IP address of the router.

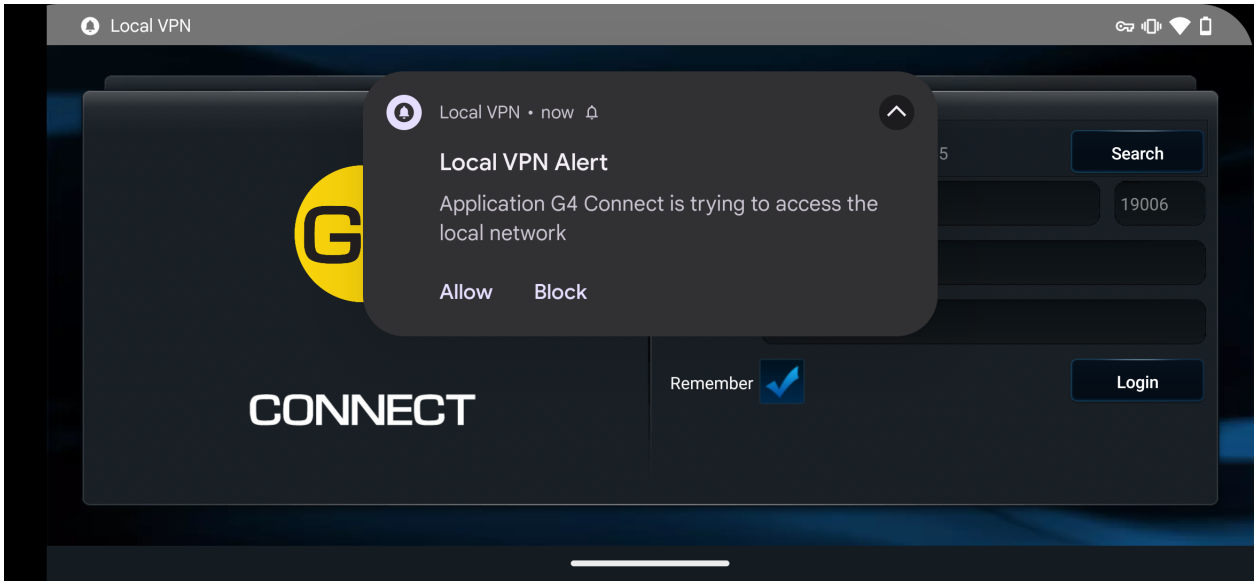

Figure 5.11: G4 Connect application tries to access the local network

Chapter 6

Conclusion

6.1 Summary

This thesis work explores the increasingly important requirement for higher security in Android devices' local networks due to the increasing prevalence of two things: smart homes and IoT technologies. It observes that there are big differences in security models for Android concerning access on local net connectivity to those applied by iOS. A practical solution to this problem can be found through the use of the LocalVPN app which enables users to observe and handle local network traffic in real time, gives the opportunity to monitor it. Here, the significance of enhancing the security framework of Android is highlighted to enhance more the safeguarding of sensitive network environments.

6.2 Future Extensions

The functionality of the LocalVPN application improves by adding a database that makes it simple for users to monitor and control which applications are allowed or blocked from connecting to the local network. The goal of this update is to give users a more transparent and user-friendly experience while giving them more control over how well they monitor and manage network access. The program is also switching from Java to Kotlin. This change happens to take use of Kotlin's enhanced safety measures and current features, which could boost the app's performance and maintainability. In addition, extensive user research will be carried out to guarantee user's usability and discover apps that try to access the Local Network.

6.3 Future Work

Threat Identification: By detecting which applications try to access the local network, the LocalVPN can help identify potentially malicious or suspicious software and help lead to the deletion of malicious applications. This aids in early threat detection and enhances overall network security.

Regulatory Compliance: The monitoring application also helps companies in certain industries characterized by heavy regulation to ensure that their business and operations stay within the laws and standards regarding access and transmission of data, without getting into expensive legal woes.

6.4 Limitations

Currently, LocalVPN saves the allowed and blocked applications in linked lists. In a future update, these lists is migrated to a database. This ensures persistent settings even in the event of application closure, hence providing an assured and robust user experience due to the help in maintaining the same access controls across sessions.

Bibliography

[1] Apple. If an app would like to connect to devices on your local network, 2023. https://support.apple.com/en-us/102229.

[2] Dns rebinding. https://spyboy.blog/2023/01/28/dns-rebinding/.

[3] Cross-site scripting (xss) writeup. https://medium.com/@muzamail-ashraf/cross-site-scripting-xss-3d9d7bd5d62f.

[4] Csrf (cross-site request forgery) explained. https://patchthenet.com/blog/csrf-cross-site-request-forgery-explained.

[5] Android Developers. Request runtime permissions. https://developer.android.com/training/permissions/requesting.

[6] Android Developers. Permissions on android, 2024. https://developer.android.com/guide/topics/permissions/overview.

[7] Medium. Evolution of location permission in android. https://medium.com/ymedialabs-innovation/evolution-of-location-permission-in-android-659448bd4e6d.

[8] Android Developers. Tristate permissions. https://source.android.com/docs/core/permissions/tristate-perms.

[9] Android Developers. One time persmission. https://developer.android.com/training/permissions/requesting#one-time.

[10] Android Developers. Request location permission. https://developer.android.com/develop/sensors-and-location/location/permissions.

[11] René Mayrhofer, Jeffrey Vander Stoep, Chad Brubaker, and Nick Kralevich. The android platform security model. *ACM Transactions on Privacy and Security (TOPS)*, 24(3):1–35, 2021.

[12] Nikita Buchka. Switcher: Android joins the 'attack-the-router' club, 2016. `https://securelist.com/switcher-android-joins-the-attack-the-router-club/76969/`.

[13] VT Lam, Spiros Antonatos, Periklis Akritidis, and Kostas G Anagnostakis. Puppetnets: Misusing web browsers as a distributed attack infrastructure. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 221–234, 2006.

[14] Jeremiah Grossman and TC Niedzialkowski. Hacking intranet websites from the outside "javascript malware just got a lot more dangerous", 2006. `https://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Grossman.pdf`.

[15] Sid Stamm, Zulfikar Ramzan, and Markus Jakobsson. Drive-by pharming. In *Information and Communications Security: 9th International Conference, ICICS 2007, Zhengzhou, China, December 12-15, 2007. Proceedings 9*, pages 495–506. Springer, 2007.

[16] Alex Tsow, Markus Jakobsson, Liu Yang, and Susanne Wetzel. Warkitting: the drive-by subversion of wireless home routers. *Journal of Digital Forensic Practice*, 1(3):179–192, 2006.

[17] Alex Tsow. Phishing with consumer electronics-malicious home routers. *MTW*, 190, 2006.

[18] Gunes Acar, Danny Yuxing Huang, Frank Li, Arvind Narayanan, and Nick Feamster. Web-based attacks to discover and control local iot devices. In *Proceedings of the 2018 Workshop on IoT Security and Privacy*, pages 29–35, 2018.

[19] Carlos E Rubio-Medrano, Pradeep Kumar Duraisamy Soundrapandian, Matthew Hill, Luis Claramunt, Jaejong Baek, Geetha S, and Gail-Joon Ahn. Dypoldroid: Protecting against permission-abuse attacks in android. *Information Systems Frontiers*, 25(2):529–548, 2023.

[20] Aniketh Girish, Tianrui Hu, Vijay Prakash, Daniel J Dubois, Srdjan Matic, Danny Yuxing Huang, Serge Egelman, Joel Reardon, Juan Tapiador, David Choffnes, et al. In the room where it happens: Characterizing local communication and threats in smart homes. In *Proceedings of the 2023 ACM on Internet Measurement Conference*, pages 437–456, 2023.

[21] EC Council. Iot security: Safeguarding critical networks against digital assaults, 2023. Accessed on 23/03/2024 from `https://www.eccouncil.org/cybersecurity-exchange/network-security/guide-to-iot-security-protecting-critical-networks/`.

[22] OpenVPN. Iot vulnerabilities for cybersecurity. `https://openvpn.net/blog/iot-device-vulnerability/`. Accessed on 10/05/2024.

[23] Vijay Sivaraman, Hassan Habibi Gharakheili, Clinton Fernandes, Narelle Clark, and Tanya Karliychuk. Smart iot devices in the home: Security and privacy implications. *IEEE Technology and Society Magazine*, 37(2):71–79, 2018.

[24] Bruce Schneier Stephen Farrell, Farzaneh Badiei and Steven M. Bellovin. Reflections on ten years past the snowden revelations. `https://www.tenable.com/plugins/nessus/12217`, 2023. Accessed on 15/5/2024.

[25] Mark Stanislav and Tod Beardsley. Hacking iot: A case study on baby monitor exposures and vulnerabilities. *Rapid7 Report*, 2015.

[26] Darlene Storm. more wireless baby monitors hacked: Hackers remotely spied on babies and parents. *Available via ComputerWorld, Apr*, 2015.

[27] Soteris Demetriou, Nan Zhang, Yeonjoon Lee, XiaoFeng Wang, Carl A Gunter, Xiaoyong Zhou, and Michael Grace. Hanguard: Sdn-driven protection of smart home wifi devices from malicious mobile apps. In *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, pages 122–133, 2017.

[28] Sukhvir Notra, Muhammad Siddiqi, Hassan Habibi Gharakheili, Vijay Sivaraman, and Roksana Boreli. An experimental study of security and privacy risks with emerging household appliances. In *2014 IEEE conference on communications and network security*, pages 79–84. IEEE, 2014.

[29] Proofpoint. Proofpoint uncovers internet of things (iot) cyberattack. https://www.proofpoint.com/us/proofpoint-uncovers-internet-things-iot-cyberattack, 2014. Accessed on 10/05/2024.

[30] Shashank Gupta and Brij Bhooshan Gupta. Cross-site scripting (xss) attacks and defense mechanisms: classification and state-of-the-art. *International Journal of System Assurance Engineering and Management*, 8:512–530, 2017.

[31] Mohammadreza Hazhirpasand, Arash Ale Ebrahim, and Oscar Nierstrasz. Stopping dns rebinding attacks in the browser. pages 596–603, 01 2021.

[32] OWASP. Cross site scripting prevention cheat sheet. https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html#xss-prevention-rules-summary.

[33] John Bergbom. Attacking the internal network from the public internet using a browser as a proxy. *Forcepoint, Mar*, 19, 2019.

[34] Vijay Sivaraman, Dominic Chan, Dylan Earl, and Roksana Boreli. Smart-phones attacking smart-homes. In *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, pages 195–200, 2016.

[35] Joel Reardon. The curious case of coulus coelib. the appcensus blog., 2022. https://blog.appcensus.io/2022/04/06/the-curious-case-of-coulus-coelib/.

[36] Wall Street Journal. Google bans apps with hidden data-harvesting software., 2022. https://www.wsj.com/articles/apps-with-hidden-data-harvestingsoftware- are-banned-by-google-11649261181.

[37] Bryson R Payne, Leonardo I Mazuran, and Tamirat Abegaz. Voice hacking: Using smartphones to spread ransomware to traditional pcs. *Journal of Cybersecurity Education, Research and Practice*, 2018(1):2, 2018.

[38] Android Developers. Internet. https://developer.android.com/reference/android/Manifest.permission#INTERNET. Accessed on 10/05/2024.

[39] Android Developers. Access network state permission. https://developer.android.com/reference/android/Manifest.permission#ACCESS_NETWORK_STATE. Accessed on 10/05/2024.

[40] Android Developers. Change network state. https://developer.android.com/reference/android/Manifest.permission#CHANGE_NETWORK_STATE. Accessed on 10/05/2024.

[41] Android Developers. Change wifi state. https://developer.android.com/reference/android/Manifest.permission#CHANGE_WIFI_STATE. Accessed on 10/05/2024.

[42] Android Developers. Proofpoint uncovers internet of things (iot) cyberattack. https://developer.android.com/develop/connectivity/wifi/wifi-permissions. Accessed on 10/05/2024.

[43] Android Developers. Nsdmanager. https://developer.android.com/reference/android/net/nsd/NsdManager, 2022. Accessed on 15/05/2024.

[44] Android Developers. Meet android studio. https://developer.android.com/studio/intro. Accessed on 20/03/2024.

[45] Android Developers. Android vpn service, 2024. https://developer.android.com/develop/connectivity/vpn.

[46] zhengchun Matt Patera Mohamed Naufal, Chuck Valenza. Localvpn. https://github.com/hexene/LocalVPN/tree/master.

[47] Android Developers. Display time-sensitive notifications. https://developer.android.com/develop/ui/views/notifications/time-sensitive, 2024. Accessed on 15/5/2024.

[48] Android Developers. Add action buttons. https://developer.android.com/develop/ui/views/notifications/build-notification#Actions, 2023. Accessed on 15/12/2023.

[49] Play Store. G4 connect. https://play.google.com/store/search?q=G4%20connect&c=apps. Accessed on 10/06/2024.