**ZHAW - MAS Informatik**

**MAS Thesis**

**Controlled communication of seized mobile devices in the IT Forensics Unit of Zurich Metropolitan Police**

A controlled network environment that completely blocks the data communication of seized mobile phones and only allows essential communication through whitelisting.

Alexander Wüst

Kronenwis 19

8864 Reichenburg

**Abgabe** 24. June 2025

**Betreuer:in** Peter Heinrich

**Abstract**

This work explores the challenge of enabling secure and controlled Internet access for seized mobile devices during forensic investigations. In many cases, access to cloud-based evidence, such as chat messages, backups, or user data, requires the device to go online. However, connecting a seized device to the Internet without strict controls introduces significant risks, including remote wipe commands, automatic data synchronisation, or background processes that may alter the evidentiary state. Ensuring access to valuable data while preserving forensic integrity is therefore a critical balance to achieve.

To address this challenge, a custom network control application was developed by combining OPNsense with the Django framework. The system enables flexible and targeted rule enforcement via DNS filtering, ISP name recognition, and enriched IP lists, allowing forensic examiners to selectively permit application-specific communication. Controlled test scenarios using both iOS and Android devices were conducted to evaluate whether predefined use cases could be executed successfully within the restricted network environment.

A central focus was placed on suppressing remote wipe and tracking functionality on both iOS and Android. For iOS, critical Find My-related DNS domains, such as fmipmobile.icloud.com, were identified and blocked, enabling access to iCloud services while preventing wipe commands. On Android, it was shown that talk.google.com and related subdomains handled wipe and ring commands, which could be selectively filtered without disrupting other Google services.

Particular attention was also given to the decentralised application Session Messenger, whose architecture bypasses traditional DNS and IP resolution.

The results demonstrate that it is technically feasible to enable access to specific cloud services, such as iCloud, WhatsApp, Telegram, and Binance, without exposing the device to broader infrastructure. DNS-based filtering proved effective in identifying and blocking wipe-related traffic while preserving access to application functionality.

Performance testing also revealed delays in sequential API-based rule creation, which could be significantly mitigated by using OPNsense Aliases, reducing rule application times from minutes to seconds.

**Table of Contents**

[1. Introduction 1](#_Toc201672083)

[1.1. Preface 1](#_Toc201672084)

[1.2. Goal of the Work 2](#_Toc201672085)

[1.3. Scope Limitations 2](#_Toc201672086)

[2. Background 3](#_Toc201672087)

[2.1. Evolution of Firewalls 3](#_Toc201672088)

[2.2. Overview of Current Firewall Solutions 3](#_Toc201672089)

[2.2.1. Open-Source Firewalls 3](#_Toc201672090)

[2.2.2. Enterprise Firewalls 4](#_Toc201672091)

[2.2.3. System-Level Firewalls 4](#_Toc201672092)

[2.2.4. Requirements for the Firewall Solution 4](#_Toc201672093)

[2.2.5. Compare Firewall Solutions Based on Requirements 5](#_Toc201672094)

[2.3. Hardware Evaluation 6](#_Toc201672095)

[2.3.1. Final Decision for Protectli V1410 6](#_Toc201672096)

[2.4. Remote Wipe 7](#_Toc201672097)

[2.4.1. Differences Between Wiping and Deletion 7](#_Toc201672098)

[2.4.2. Remote Wipe - Apple iOS 7](#_Toc201672099)

[2.4.3. Remote Wipe - Google Android 7](#_Toc201672100)

[3. Implementation 8](#_Toc201672101)

[3.1. Overview of the System Architecture 8](#_Toc201672102)

[3.2. Prototyping and Preliminary Testing 9](#_Toc201672103)

[3.2.1. Firewall Configuration During Prototyping 9](#_Toc201672104)

[3.2.2. API Integration and Log Retrieval 10](#_Toc201672105)

[3.2.3. MAC Address Handling and Device Identification 10](#_Toc201672106)

[3.2.4. DNS Rules Set to Access 10](#_Toc201672107)

[3.2.5. Impact of Apple Private Relay on Firewall Logfiles 11](#_Toc201672108)

[3.2.6. IP Enrichment and Company Identification 11](#_Toc201672109)

[3.2.7. Apple Find My Device Tests 12](#_Toc201672110)

[3.2.8. Verification of Basic Use Cases 13](#_Toc201672111)

[3.2.9. System Setup Recommendations After Preliminary Tests 14](#_Toc201672112)

[3.2.10. Conclusion of Section 3.2 15](#_Toc201672113)

[3.3. Backend Implementation 15](#_Toc201672114)

[3.3.1. Backend Data Schema 15](#_Toc201672115)

[3.3.2. api\_dhcp\_parser.py - Lease Synchronization 15](#_Toc201672116)

[3.3.3. api\_firewall\_sync.py - Device-Based Firewall Enforcement 15](#_Toc201672117)

[3.3.4. api\_firewall.py - Firewall API Abstraction 16](#_Toc201672118)

[3.3.5. api\_logs\_parser.py - Log Parsing and IP Enrichment 17](#_Toc201672119)

[3.3.6. ssh\_dns\_parser.py – Live DNS Parsing 17](#_Toc201672120)

[3.4. Frontend Implementation 17](#_Toc201672121)

[3.4.1. Technologies Used 18](#_Toc201672122)

[3.4.2. Backend Integration 18](#_Toc201672123)

[3.4.3. Functional Views 18](#_Toc201672124)

[3.4.4. Blocked View 19](#_Toc201672125)

[3.4.5. Passed View 20](#_Toc201672126)

[3.4.6. ISP View 20](#_Toc201672127)

[3.4.7. DNS View 21](#_Toc201672128)

[3.4.8. Manage Devices View 22](#_Toc201672129)

[3.4.9. Domain Lookup View 23](#_Toc201672130)

[3.4.10. Firewall Rules View 23](#_Toc201672131)

[3.4.11. Device Logs View 24](#_Toc201672132)

[3.5. Integration with OPNsense 24](#_Toc201672133)

[3.6. Rule Application Logic 25](#_Toc201672134)

[3.6.1. Rule Evaluation and Creation Logic 26](#_Toc201672135)

[3.6.2. Device Archival and Rules Removal 26](#_Toc201672136)

[3.6.3. Rule Cleanup and Removal 27](#_Toc201672137)

[3.7. Logging 27](#_Toc201672138)

[3.7.1. Purpose and Scope 27](#_Toc201672139)

[3.7.2. Database-Backed Structure 27](#_Toc201672140)

[3.7.3. Operational Visibility 28](#_Toc201672141)

[3.7.4. Planned Enhancements 28](#_Toc201672142)

[3.8. Challenges Encountered 28](#_Toc201672143)

[3.8.1. Granular App Control via IP and Firewall Rules 28](#_Toc201672144)

[3.8.2. IP Enrichment Bottlenecks 28](#_Toc201672145)

[3.8.3. Firewall Rule Propagation Delay 29](#_Toc201672146)

[3.8.4. MAC Address Randomization and IP Variability 29](#_Toc201672147)

[3.9. Summary 29](#_Toc201672148)

[4. Analysis and Results 30](#_Toc201672149)

[4.1. Prevent Remote Wipe on Mobile Devices 30](#_Toc201672150)

[4.1.1. iOS Remote Wipe Observations 30](#_Toc201672151)

[4.1.2. Android Remote Wipe Observations 32](#_Toc201672152)

[4.2. Practical Use Case Validation 33](#_Toc201672153)

[4.2.1. Test Environment Setup 33](#_Toc201672154)

[4.2.2. WhatsApp Messenger Access (iOS / Android) 34](#_Toc201672155)

[4.2.3. Telegram Messenger Access (iOS / Android) 35](#_Toc201672156)

[4.2.4. Session Messenger Access (iOS / Android) 36](#_Toc201672157)

[4.2.5. iCloud Photos Access – Media Retrieval (iOS) 37](#_Toc201672158)

[4.2.6. Binance - Account and Transactions Retrieval (iOS / Android) 38](#_Toc201672159)

[4.3. Firewall Rule Propagation Delay 39](#_Toc201672160)

[5. Discussion and Conclusion 40](#_Toc201672161)

[5.1. Summary of Key Findings 40](#_Toc201672162)

[5.1.1. Wipe Risk Mitigation 40](#_Toc201672163)

[5.1.2. iOS vs. Android Differences 40](#_Toc201672164)

[5.1.3. Session Messenger Insight 41](#_Toc201672165)

[5.1.4. Firewall Rule Handling 41](#_Toc201672166)

[5.1.5. Device Scope 41](#_Toc201672167)

[5.1.6. Third-Party MDM Services 42](#_Toc201672168)

[5.2. Practical Implications for Forensics 42](#_Toc201672169)

[5.3. Practical Recommendations 42](#_Toc201672170)

[5.4. Future Work 43](#_Toc201672171)

[References 45](#_Toc201672172)

[Appendix 47](#_Toc201672173)

[Declaration of Originality 48](#_Toc201672174)

# Introduction

## Preface

Electronic evidence has become a cornerstone of modern criminal investigations. In most cases, digital data, whether from mobile phones, computers, IoT devices, or cloud services, plays an important role in uncovering facts and supporting prosecution. Digital forensics plays a crucial role in this. IT forensic experts prepare the seized devices according to forensic standards to ensure that the data collected can be used in court.

It's important that the collected devices are completely isolated from data networks after seizing them to maintain data integrity. If this is not possible due to security settings, the Zurich Metropolitan Police use a Faraday room. This prevents the mobile phone from communicating with external networks such as cellular, WLAN, and Bluetooth, and protects the data on the device from being modified externally. As soon as a mobile phone is reconnected to the Internet it will start updating apps and synchronising data, such as cloud services, messages and other application data, which will, of course, trigger write processes on the data storage. There is also a risk of remote deletion of the device which needs to be avoided.

A fundamental principle of forensic work is the reproducibility of the results. Once write operations occur on the data storage, this reproducibility can no longer be guaranteed. So, it is clear that establishing a network connection, or even simply powering on a device, is not an option and needs to be avoided.

However, best practices at the Metropolitan Police Zurich have shown that powering on a device is necessary to verify whether the data acquired by the forensic hardware and software has been processed correctly. A defined protocol is followed to check for any apps or entries that may not have been parsed correctly. In fact, there are often problems during parsing in practice. Especially apps that are only known in Switzerland, like “Twint”, are not parsed automatically.

Without powering on the device, it is not possible to detect parsing errors. For example, powering on the device may reveal that no WhatsApp messages are shown or that media files such as photos are missing. In recent years, when physical access to a device was possible, like with a known or brute-forced passcode, it has proven useful to use this access to validate the acquisition and ensure no data was lost or misinterpreted during the process. The described procedure, although it also causes write operations, it is considered essential to ensure a good and complete data acquisition. Of course, this procedure must be conducted in a Faraday room or with airplane mode enabled.

It is increasingly common that not everything that is accessible on the device is saved on the device. Common examples are images and videos that are stored directly on provider servers (cloud). When manually reviewing chat conversations, it can happen that the content of chats clearly goes in one direction, but the images and videos may be missing because they were never stored locally. For example, an image that could be identified as an offence or a prohibited media file containing violence or even child pornography may be missing from the device, which prevents drawing clear conclusions. Or the communication seems to be clear, but the pictures are harmless? So, it is important to give during a search the best possible picture. This raises the question: how can this additional information be obtained without bringing the device online? The best way is to access the data directly with a cloud acquisition method. For this method the service needs to be supported, and a valid token needs to be extracted from the device. Sometimes even that method is not possible due to different reasons. Then there is typically a consultation between the public prosecutor, the police officer in charge of the case and the forensic examiner. One of the biggest problems is the possibility of a remote deletion when taking the device online. As the acquisition through forensic hardware and software has already taken place it will still be possible to have the original copy of the state of the device when it was seized.

There is no easy solution that makes it possible to take a device online in a controlled environment without risking remote data deletion. At the same time, only the necessary connections to the Internet should be allowed. This MAS thesis aims to close this gap.

## Goal of the Work

The primary objective of this thesis is to implement a wireless network with an integrated firewall, which by default blocks all Internet access for connected devices. A central focus of the project is the development of a custom web interface that interacts with a firewall solution via its API. Through this interface, the forensic examiner will be able to monitor and configure firewall settings and define individual access rules for connected devices.

The proposed solution targets an easy-to-use interface that allows the forensic examiner to make informed decisions about which network connections should be permitted and which should remain blocked. All connection attempts will be logged to ensure transparency and accountability. This is an essential requirement in forensic investigations. To meet these goals, the project includes the development of an application with the following core capabilities:

* Centralized management of device-based firewall rules
* Monitoring of network activity
* Logging of all connection attempts and allowed traffic
* Easy addition and removal of rules through a user-friendly interface

Therefore, suitable hardware will be evaluated and bought to ensure compatibility with the firewall solution and to meet performance and environmental requirements (e.g., rugged form factor, multiple network interfaces).

Later in section 4.2, the configured setup will be validated against defined use cases to assess practicability, usability, and verify whether the targeted functionality can be achieved without exposing the device to unnecessary services or risking remote wipe. This validation step will help confirm the real-world applicability and safety of the proposed firewall control mechanisms in forensic scenarios.

## Scope Limitations

This MAS thesis explicitly excludes the following topics from its scope:

* Implementation or evaluation of two-factor authentication methods using mobile networks (e.g., MobileID, SMS-based authentication)
* Analysis of write operations on mobile devices that may occur as a result of simply powering them on, even if considered best practice in forensic handling
* Integration of Deep Packet Inspection (DPI) or content-level traffic analysis within the firewall
* Examination of legal considerations or compliance issues, such as data protection laws, admissibility of evidence, or regulatory frameworks

The focus of this thesis remains on the technical implementation of a device-aware firewall management system that enables controlled network access and supports forensic transparency.

# Background

## Evolution of Firewalls

As computer networks began to connect to each other, the need to protect one network from the other and avoid external threats became increasingly important. The name “Firewall” came from physical barriers used in architecture and history: walls that protected cities, like the Great Wall of China, or structural firewalls that prevented a fire in the kitchen from spreading to other parts of the building [1]. Similarly, in networking, a firewall serves as a barrier to block unwanted traffic while allowing legitimate traffic to pass through. Even if the “external side” is burning, the firewall is intended to keep the internal network safe.

The development of firewalls has progressed through several generations. First-generation firewalls in the 1990s used basic packet filtering to allow or block traffic based on IP addresses and ports.

In the early 2000s, the second generation, stateful inspection firewalls emerged, introducing the ability to track the state of connections and inspect traffic in context, offering enhanced control. Then, application-layer firewalls enabled content-aware filtering and protocol-specific inspection, further increasing security.

Around 2008, the third generation, also called Next-Generation Firewalls (NGFWs), integrated traditional firewall functions with deep packet inspection (DPI), intrusion prevention systems (IPS), and application-level awareness.

The fourth firewall generation was launched around 2020. These firewalls use machine learning to detect zero-day threats in real time, going beyond traditional signature-based methods. Key features include zero-delay signature updates, automated security policy recommendations, and IoT device visibility based on behavioural analysis. These systems continuously learn from network traffic and aim to reduce manual intervention [2][3].

## Overview of Current Firewall Solutions

This section provides an overview of currently available firewall solutions, focusing on open-source, enterprise and system-integrated options.

### Open-Source Firewalls

An open-source firewall is a firewall solution whose source code is publicly available and freely accessible. The source code can be reviewed and modified. These firewalls are typically community-driven and cost-effective. One major advantage is the opportunity to customize the product. As a result, many projects offer commercial services, such as professional support and additional enterprise features. This helps the organizations behind the projects maintain the software and fund the necessary infrastructure (Table 1). [3]

Table 1: Overview of Open Source Firewalls

|  |  |
| --- | --- |
| **Firewall** | **Description** |
| pfSense[4] | A flexible, open-source firewall and router platform that offers NAT, packet filtering, and next-generation firewall features. It supports multiple interfaces, scales well, and provides a command-line interface for advanced configuration. |
| OPNSense[5] | A user-friendly, web-managed next-generation firewall with built-in intrusion detection (IDS), web filtering, and VPN support. It combines strong security features with an intuitive interface, making it suitable for many network environments. |
| VyOS[6] | A community-driven, fully open-source firewall that aims for high availability and uptime. It includes stateful inspection, NAT, and routing features, and is often used in hardware appliances for continuous performance. |
| ClearOS[7] | A simple, stateful firewall solution aimed at users with basic network protection needs. It is easy to manage and configure but lacks advanced features like NAT and packet filtering. |

### Enterprise Firewalls

Enterprise firewalls are typically closed-source and, unlike open-source alternatives, are not freely accessible or modifiable. These solutions often include a wide range of additional services, such as professional support, subscription-based threat intelligence, and service-level agreements (SLAs), making them particularly attractive to organizations that require guaranteed uptime, vendor accountability, and integrated security management.

Well-known enterprise firewall vendors include Palo Alto Networks, Fortinet, Cisco,Check Point, and Sophos.

### System-Level Firewalls

In addition to complete firewall platforms, Linux-based systems offer built-in packet filtering frameworks that provide fine-grained control over network traffic. These tools are often used for configuring firewalls at a lower level or for integration within larger systems. While they require more technical knowledge compared to full-featured firewall platforms, they offer flexibility and are widely used (Table 2).

Table 2: Overview of System Level Firewalls

|  |  |
| --- | --- |
| **Firewall** | **Description** |
| iptables | A user-space utility program that allows administrators to configure the Linux kernel’s Netfilter framework for packet filtering and NAT. It has traditionally been the default tool for firewall configuration on Linux systems. Although powerful, its syntax can be complex, especially in large-scale or dynamic environments. It is still widely supported and used in many legacy systems [8], [9]. |
| nftables | The modern replacement for iptables, introduced to simplify rule management and unify IPv4, IPv6, ARP, and other protocol filtering under a single framework. It offers a more consistent and readable syntax, improved performance, and better integration. nftables is designed to reduce redundancy and support more maintainable and scalable rule sets [10]. |
| ufw | Is a frontend for iptables designed to simplify firewall configuration for end users. It provides a user-friendly command-line interface and is often installed on Ubuntu based systems. While it lacks the depth of customization available in iptables or nftables, it is sufficient for basic firewall setups and often used in smaller deployments [11]. |

### Requirements for the Firewall Solution

Early in the project phase, it was essential to determine the appropriate firewall platform. The primary goal was to build an extended interface tailored to our specific needs, rather than developing a complete firewall solution from scratch. The selected system needed to provide a robust and flexible foundation. For evaluation purposes, a minimal set of functional and technical requirements was defined:

|  |  |
| --- | --- |
| **Requirements** | **Description** |
| Open-Source | The firewall must be distributed under an open-source license. This ensures the solution is free of licensing costs and avoids vendor lock-in. Furthermore, open-source access enables future customization, which may become necessary in advanced stages of the project. |
| Self-Hosting | The solution must be fully deployable on the organization’s own infrastructure without reliance on external cloud components or third-party services. This is essential for maintaining full control over processed data, meeting forensic standards, and complying with the Information Security and Data Protection policies (ISDS) of the City of Zurich. |
| API Support | A modern, well-documented API is required. The project aims to integrate firewall control into a custom web interface, eliminating the need to interact directly with the firewall’s native user interface. API support is also essential for automation workflows. |
| Device-Specific Rules | The firewall must support rules that can be defined per device, based on identifiers such as IP or MAC addresses. This capability is necessary for implementing granular access policies, for example, allowing one device to access a specific IP while restricting another. |
| Active Development | The firewall must be actively maintained, with ongoing development and regular updates, including security patches. |
| Feature-Rich | The solution should include a wide range of network security features out of the box, minimizing reliance on third-party tools and enabling support for potential future processing steps. |

### Compare Firewall Solutions Based on Requirements

Since enterprise firewall solutions (e.g., Palo Alto, Fortinet, Cisco, Check Point, and Sophos) are proprietary and do not meet the open-source licensing requirement, they were excluded from the comparison table below. These products are therefore considered out of scope for this project, which focuses solely on open-source, self-hosted solutions. Table 3 presents a comparison between system-level and the open-source firewall platforms.

Table 3: Comparison of different firewall solutions

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Firewall** | **Open Source** | **Self-Hosting** | **API Support** | **Device-Aware Rules** | **Active Development** | **Feature-Rich** |
| iptables[12] | Yes | Yes | No | Limited | Yes | Moderate |
| nftables[10] | Yes | Yes | No | Limited | Yes | Yes |
| pfSense[4] | Yes | Yes | Limited | Yes | Yes | Yes |
| OPNSense[5] | Yes | Yes | Yes | Yes | Yes | Yes |
| VyOS[6] | Yes | Yes | Limited | Limited | Yes | Yes |
| ClearOS[7] | Yes | Yes | Limited | No | No | Decreasing with time as no active development is done |

After evaluating the available open-source firewall projects and reviewing their official documentation and community resources, **OPNsense** was selected as the base firewall for this project. OPNsense is an actively maintained platform with a large user and developer community. A major advantage of OPNsense is its officially supported and well-documented API, which already covers most, if not all, of the functions required for this project. The project also offers regular updates, an open development roadmap, and detailed release notes. Although the project could have been implemented using a lower-level base like nftables, which is integrated directly into the Linux kernel, this approach would have required significantly more development effort. Due to the time constraints, a ready-to-use solution was selected.

## Hardware Evaluation

Hardware selection was not a primary focus at the outset of the project. After the decision to use OPNsense as the firewall platform, further research was conducted using online sources, including general web searches, to identify suitable, cost-effective hardware configurations.

The official OPNsense hardware appliance offerings [13] were reviewed but ultimately deemed out of scope due to their high cost. These systems are primarily targeted at enterprise customers, with pricing that often exceeded the available budget. The project aimed to remain within a budget of just a few hundred Swiss francs, ideally utilizing small-form-factor, low-cost hardware such as the Raspberry Pi 5.

While the Raspberry Pi 5 initially appeared promising due to its affordability, and it technically meets OPNsense’s official recommended hardware requirements [14] (Table 4), it was ultimately excluded. It failed to satisfy several project-specific requirements essential for deployment in a forensic laboratory environment.

The selected device had to be enclosed in a rugged case suitable for continuous operation in a professional setting. It also needed to support at least one WAN port and two or more LAN ports (e.g., one for Wi-Fi and one or more for Ethernet-connected devices). While rugged enclosures for the Raspberry Pi exist, they are typically part of DIY solutions and often require additional adapters or USB-to-Ethernet dongles, which compromise both reliability and performance.

During the evaluation of networking options, it became clear that no off-the-shelf Raspberry Pi configuration could meet these requirements. In the OPNsense forums [15], many community members shared links to low-cost firewall hardware available via platforms such as AliExpress. However, this option was not pursued further due to concerns regarding warranty coverage, lack of technical support, and long delivery times, often several weeks. These factors would have introduced unnecessary risk and delays to the project timeline.

Table 4: Recommended Hardware Specification for OPNsense

|  |  |
| --- | --- |
| **Component** | **Specification** |
| Processor | 1.5 GHz multi core cpu |
| RAM | 8 GB |
| Free Space | 120 GB SSD |

### Final Decision for Protectli V1410

Finally, the decision was made to purchase hardware from Protectli, a U.S.-based company with a location in Rossdorf, Germany. Protectli is specialized in hardware for open-source firewall appliances and is well-regarded for its compatibility with OPNsense. After evaluating several of the company’s models, the Protectli V1410 was selected, as it fulfilled all functional requirements for the project, including port availability, compact form factor, hardware durability, and full compatibility with the OPNsense software platform (Table 5).

Table 5: Protectli V1410 Overview

|  |  |
| --- | --- |
| **Device** | **Protectli V1410** |
| CPU | Intel® N5105 Quad-Core, 2.0 GHz (Turbo up to 2.9 GHz) |
| Networking | 4 × Intel® I226-V 2.5 GbE RJ-45 Ethernet ports |
| Memory | 8 GB LPDDR4 (on-board) |
| Storage | 32 GB onboard eMMC and 250 GB Kingston NVMe (NV2-250G) |
| Expansion | M.2 slots for optional Wi-Fi or LTE modules |
| Power Supply | 12 V with screw-in connector (included) |
| Other | Fanless, silent operation, coreboot-supported, compact form factor |
| Price | €284.55 (excluding VAT, as of February 17, 2025) |

This hardware provides sufficient performance and operational flexibility for use in a forensic laboratory environment. It fully meets the hardware requirements for OPNsense and includes additional storage capacity for extended logging or future use cases. The fanless design and industrial grade build further support long-term maintainability and stability under continuous operation. In the following section, the focus shifts from the software and hardware infrastructure to background information on the challenges of remote wipe in the context of seized devices, with a particular attention to iOS and Android operating systems.

## Remote Wipe

«*Remote wipe is a security feature that allows a network administrator or device owner to send a command that remotely deletes data from a computing device. It's primarily used to erase data on a device that has been lost or stolen, so the data won't be compromised if it falls into the wrong hands.*» [16]

### Differences Between Wiping and Deletion

Wiping refers to the process that ensures data stored on a device is irreversibly destroyed, making recovery impossible. This is typically achieved by overwriting specific storage areas, or the entire storage, with random data. In contrast, a simple file deletion only removes system references to the data, while the actual data remains intact on the storage device. This residual data can often be recovered using specialized tools and software, provided it has not been overwritten due to subsequent disk usage.

### Remote Wipe - Apple iOS

To understand how remote wiping functions on Apple iOS devices, it is important to first understand how the Apple ecosystem handles encryption. When an iOS device is initially set up, or after a factory reset, it generates a unique volume key, often referred to as the media key. This key is essential for encrypting all user data on the device. The media key is securely stored in a dedicated hardware component called the Secure Enclave, which is designed to protect sensitive information even in the event of physical compromise. All data on the device, including app data, system files, and user data, is encrypted using this key. Without it, access to stored data is not possible. When a remote wipe command is triggered, either via Mobile Device Management (MDM) or directly through the user’s iCloud account, the device will respond accordingly. In the case of MDM, the device first sends an acknowledgment response upon receiving the command. It then immediately executes the wipe. It is important to note that in MDM-managed devices, the remote wipe command can originate not only from Apple but also from authorized third-party vendors, such as Jamf, Microsoft, or MobileIron [17] [18]. The remote wipe operation does not erase all data directly. Instead, it removes the media key. Because the encrypted data cannot be decrypted without this key, the result is effectively the same as a full wipe, but much faster and more efficient. Even the device owner cannot access the data, as there is no known method to decrypt the content without the deleted key.

### Remote Wipe - Google Android

The Android operating system functions slightly differently from iOS. Beginning with Android 10, the system moved away from full-disk encryption (FDE) and adopted file-based encryption (FBE). After initial setup or a factory reset, the system generates a set of encryption keys: the Credential Encrypted (CE) key and the Device Encrypted (DE) key. These keys are stored in secure hardware components such as the Trusted Execution Environment (TEE) or a Secure Element. Android uses a layered key encryption model, in which all data on the device, including user data, app data, and most system files, is encrypted using individual keys. These unique file-level keys are themselves encrypted using either the CE or DE key. The CE key is only accessible after the user has successfully authenticated. Without this key, it is not possible to decrypt or access the stored data on the device [19]. A remote wipe command can be triggered using Google’s Find My Device service or via a Mobile Device Management (MDM) system, similar to Apple’s ecosystem. When the device receives a remote wipe command, it may first respond with an acknowledgment (in the case of MDM) before immediately initiating the wipe process. This procedure is primarily designed to destroy the CE and DE keys. The underlying data is not overwritten; however, without the encryption keys, the data becomes inaccessible and unrecoverable [21][22]. The wipe command can originate not only from Google but also from a variety of enterprise MDM providers if the device is enrolled in their management systems. These include Microsoft, VMware Workspace ONE, IBM MaaS360, MobileIron, Samsung Knox Manage, and many others [22].

# Implementation

This chapter describes the implementation of the system developed as part of this thesis. The goal was to create a practical solution that enables centralized network control. At the core of the system is a custom-built web application that communicates with an open-source firewall (OPNsense) via its API.

The implementation covers several key components: the design and development of the web interface, the backend logic that processes user input and interacts with the firewall, integration with the OPNsense API, logging and storage of all relevant data in a database, and the deployment of the system on suitable hardware.

## Overview of the System Architecture

The prototype developed in this thesis consists of several integrated components that together form a modular system. The main components of the system are listed below (Table 6). Figure 1 illustrates how the individual components interact with each other.

Table 6: Overview Core Components of the System Architecture

|  |  |
| --- | --- |
| **Component** | **Description** |
| Firewall | The open-source firewall OPNsense (version 25.1 – Ultimate Unicorn) serves as the foundation of the system. It enforces all network traffic rules and blocks all connections by default unless an explicit PASS rule is defined. Most interactions with the firewall are performed through its officially supported API, which facilitates dynamic rule management and log retrieval. DNS queries, however, are captured separately via SSH access using tcpdump, allowing real-time parsing and analysis of DNS traffic. The firewall system is deployed on a Protectli V1410. |
| Access Point: | A dedicated wireless access point, specifically, a TP-Link Omada EAP610GP-DESKTOP WLAN Access Point 1201, provides connectivity for mobile devices within the isolated test environment. |
| Web Framework | The user interface is implemented using the Django web framework (version 5.1.6). It allows forensic examiner to manage devices, view logs and configure firewall rules. The interface is hold minimalistic and focused on core functions. |
| Backend Logic | The core application logic is written in Python (version 3.13.1) and organized within the Django framework. It processes user input from the web interface, manages database operations and communicates with the OPNsense API. |
| Database | The system uses SQLite as a lightweight relational database to store device metadata, firewall rules and network logs. SQLite was chosen to simplify development and deployment. Django’s model-view architecture provides abstraction over the database layer, making future migration to a more robust database system (such as PostgreSQL or MySQL) straightforward if needed. |

A diagram of a router and a computer

AI-generated content may be incorrect.

Figure 1: Overview about the system setup

## Prototyping and Preliminary Testing

Before implementing the full system, several exploratory prototyping steps were undertaken using basic Python console scripts. As the project domain was entirely new to the author, this phase was essential for identifying potential challenges related to device behaviour, DNS resolution, and firewall integration. One of the key uncertainties at the outset was how connected devices would resolve hostnames, specifically, whether they would cache and reuse IP addresses or perform DNS resolution dynamically for each request. Additionally, it was unclear how large-scale services (e.g., Google, Apple) would behave in terms of load balancing, content distribution, and dynamic IP usage. These factors could impact both the effectiveness of firewall rules and the clarity of network logging.

To address these uncertainties, early prototypes focused on:

* Observing device behavior in a restricted network
* Creating simple scripts to parse firewall logs
* Testing hostname resolution patterns
* Understanding how network traffic behaves under different conditions

### Firewall Configuration During Prototyping

In the initial testing environment, the following firewall configuration strategy was applied in OPNsense:

**Internal Access Rule:** All traffic from the internal LAN\_net to the firewall itself was allowed. This ensured continued access to the web GUI and API during testing, regardless of other firewall restrictions.

**Global Deny Rule:** A final “block-all” rule was implemented to deny all traffic not explicitly allowed, both inbound and outbound. This ensured that newly connected devices would not have any WAN access unless a rule was programmatically defined.

### API Integration and Log Retrieval

A key objective of the prototype phase was to verify whether the OPNsense API could be used reliably to:

* Configure firewall rules
* Retrieve log entries, particularly for blocked connections

Initial API requests successfully returned a list of firewall rules that had been created programmatically via the API. However, rules configured manually through the web interface were not visible through the API. This limitation was considered acceptable, as the final system was designed to manage all firewall rules exclusively through automated API interactions. Early attempts to retrieve log entries were unsuccessful. However, after testing alternative endpoints and refining the request formatting, successful access to the firewall logs was achieved. This confirmed that sufficient data could be retrieved programmatically to support the intended functionality of the system.

### MAC Address Handling and Device Identification

During early testing, a real-world scenario involving an Apple iPhone and Apple Watch revealed a key challenge for the planned firewall rule management system. When the iPhone connected to the forensic WLAN for the first time, two distinct IP and MAC address pairs appeared within the same time frame. This initially raised concerns about MAC address randomization [23], a known privacy feature in modern mobile operating systems. Further investigation, however, showed that the second MAC/IP pair originated from a paired Apple Watch, which had automatically joined the network shortly after the iPhone. After manually deleting the DHCP leases in OPNsense, the iPhone reconnected and retained a consistent MAC address (02:89:49:2b:5c:f2), contrary to expectations that it would randomize its MAC address after each session due to iOS’s default privacy settings. Nevertheless, this behaviour highlighted the importance of designing the firewall management system to handle devices with multiple or changing MAC addresses. To address this, a mechanism for centralized device identification was proposed: each device would be assigned a unique “Asservaten-Nummer” (evidence ID), and all known MAC addresses associated with that device would be linked to this ID in the database.

### DNS Rules Set to Access

During initial firewall logging tests, it was observed that outbound traffic from devices, such as the test iPhone, appeared to target only internal IP addresses. For example, repeated connection attempts were logged from 192.168.5.155 (the test iPhone) to 192.168.5.1 (the OPNsense firewall). These requests were being blocked, and no external IP addresses appeared in the logs, initially suggesting minimal or idle network behaviour. Upon further investigation, it became clear that these blocked requests were the result of the iPhone attempting to resolve DNS queries using the default gateway IP address, in this case, the firewall. This is standard behaviour when no external DNS server is explicitly defined. By default, Apple devices automatically use the network-assigned DNS server. In many setups, the router is assigned as the DNS resolver, which is expected to forward queries to an upstream server or act as a local caching resolver. This observation revealed a critical architectural requirement: DNS resolution must be permitted for connected devices. Without it, hostname lookups will fail, rendering most modern applications unable to initiate outbound connections. To address this, the integrated Unbound DNS service in OPNsense was configured to listen on port 53. Extended logging was enabled to capture all DNS activity for analysis. Additionally, a port forwarding rule was implemented to redirect all DNS traffic from the local network, regardless of its intended destination, to the OPNsense DNS service. This redirection ensures that even if a seized device is manually configured to use an external DNS server, its DNS requests will be intercepted and resolved locally. As a result, no modifications are required on the seized devices themselves.

### Impact of Apple Private Relay on Firewall Logfiles

During further testing, it was observed that when accessing websites such as 20min.ch or google.ch using the Safari browser on an iPhone, the expected destination IP addresses did not appear in the firewall logs. Instead, the logs consistently showed traffic directed toward Apple-owned IP ranges. Subsequent analysis revealed that the test device had Apple iCloud Private Relay enabled [24]. This feature, available only to users with a paid iCloud+ subscription, functions as a privacy-preserving proxy system. When enabled, it masks the destination IP addresses of web requests made through Safari by routing traffic through a series of Apple-operated and third-party relay servers. As a result, the true destination of a web request is obscured, and only the intermediate Apple-controlled IP addresses are visible to the firewall. This behaviour significantly limits visibility into user activity when Private Relay is active.

### IP Enrichment and Company Identification

During testing, it quickly became evident that a raw list of IP addresses, often exceeding 100 entries per device, was insufficient for effective analysis. Simply presenting numeric IP addresses without any contextual information made it difficult to determine which blocked connections were legitimate and should be added to the allowed firewall rules list. Without additional data such as hostnames, ISP information, or service ownership, manual evaluation was impractical. As a result, an automatic IP enrichment process was required.

**Initial Attempt – Reverse DNS Lookup:** The first approach implemented was a reverse DNS lookup using Python’s built-in socket library. In some cases, this method returned helpful results, with hostnames clearly indicating the associated service or organization (e.g., whatsapp-chatd-edge-shv-01-zrh1.facebook.com, 80-67-82-201.deploy.static.akamaitechnologies.com). However, this method proved unreliable for several reasons:

* Many IP addresses did not resolve to any hostname
* Others resolved to generic or non-informative names
* Reverse DNS resolution lacks consistency across regions and providers

**IP Metadata API as an Improved Approach:** To overcome these limitations, the system was extended to query external IP metadata APIs to retrieve more structured and reliable information for each IP address. Several services were evaluated based on available features, rate limitations, and pricing models. As a result of this evaluation, ip-api.com was selected due to its balance of functionality and high free-tier usage. The service ipinfo.io was also considered, it offers high-quality data but has significant limitations on its free tier (Table 7).

Table 7: IP API Service overview

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Service** | **Free** | **Commercial use** | **Requests** | **Note** |
| ipregistry.co[25] | Limited | Yes | 100’000 | 100’000 requests in free account |
| ipapi.com[26] | Yes | Yes | 0.8h / 100m |  |
| ipwhois.io[27] | Yes | No | 13.8h / 10’000m |  |
| geo.ipify.org[28] | Limited | Yes | 500 | 500 requests in free account |
| ipgeolocation.io[29] | Yes | Yes | 41.6h / 30’000m |  |
| ipdata.co[30] | Yes | No | 62.5h / 45’000m |  |
| ip-api.com[31] | Yes | No | 2700h / 1’944’000m | 45 requests per minute |
| ipapi.co[26] | Yes | No | 41.7h / 30’000m |  |
| ipinfo.io[32] | Yes | Yes | Unlimited | Limited details |

The service ip-api.com was chosen based on the following criteria:

* Detailed ISP information, useful for grouping IP addresses by organization (ISP)
* A clear and high free tier (45 requests per minute)
* An affordable premium option ($13.30/month) with no rate limits, makint it suitable for future production use.

To respect the free-tier limitations during development, API requests were issued with a 1.4-second delay between each call. This strategy ensured stability and avoided exceeding the rate limit. Figure 2 illustrates a sample API response for the IP address 52.97.201.242.

Figure 2: API response for 52.97.201.242 from ip-api.com

{

"query": "52.97.201.242",

"status": "success",

"continent": "Europe",

"continentCode": "EU",

"country": "Switzerland",

"countryCode": "CH",

"region": "ZH",

"regionName": "Zurich",

"city": "Zurich",

"district": "",

"zip": "",

"lat": 47.3768,

"lon": 8.5416,

"timezone": "Europe/Zurich",

"offset": 7200,

"currency": "CHF",

"isp": "Microsoft Corporation",

"org": "Microsoft Corporation",

"as": "AS8075 Microsoft Corporation",

"asname": "MICROSOFT-CORP-MSN-AS-BLOCK",

"mobile": false,

"proxy": false,

"hosting": true

}

The implementation showed excellent results in practice. For most IP addresses, clear metadata could be retrieved, including fields such as isp, org, as, asname, and country. The isp field, in particular, proved to be a reliable indicator for associating IP addresses with specific companies.

### Apple Find My Device Tests

Further tests were conducted using the same iPhone previously connected to the project WLAN, with all network traffic blocked by default. The device was logged into the same iCloud account as a MacBook, which had unrestricted Internet access. Both devices had Bluetooth enabled, raising the question of whether the iPhone could receive Find My Device commands via Bluetooth, even while isolated from direct Internet access. Contrary to initial expectations, the iPhone did not behave like an AirTag. When the MacBook issued a “Play Sound” command via iCloud, the network-isolated iPhone neither updated its location nor played a sound. This suggests that Find My Device commands are queued and require a direct Internet connection to be received. No background peer-to-peer Bluetooth relay behaviour was observed in this context. This observation aligns with findings from Josh Hickman, author of the Binary Hick blog, who tested similar scenarios with iOS 15. He concluded that powered off or disconnected iPhones do not receive Find My commands unless they have direct Internet access [33]. In his post “iOS 15 powered-off tracking & remote bombs,” he noted that, unlike AirTags, iPhones do not appear to receive location updates or remote alerts via Bluetooth relays from nearby Apple devices, at least not under network-isolated test conditions. While current testing indicates that iPhones do not receive Find My commands via Bluetooth relays when network-isolated, this behaviour is subject to change. iPhones contain much of the same hardware found in AirTags, such as Bluetooth and Ultra-Wideband (UWB) chips, and Apple controls both the hardware and software ecosystem. Therefore, it is technically feasible for Apple to enable peer-to-peer tracking functionality on iPhones through a future software update. This could allow Find My commands or other remote interactions to be delivered via nearby Apple devices, even in the absence of a direct Internet connection.

### Verification of Basic Use Cases

To verify the real-world usability of the console-based prototype, a series of simple tests were conducted. The test device was again an Apple iPhone. It was prepared by disabling iCloud Private Relay, deactivating MAC address randomization, and closing all running applications. The device was then connected to the WLAN and left idle for an extended period of 30 minutes to observe its background network behaviour.

**Background Traffic Observation:** Despite no apps being actively used, the system recorded 7,468 blocked connection attempts, the majority of which originated from Apple system services. These requests targeted over 50 unique IP addresses, all within the 17.0.0.0/8 block. According to data from ip-api.com, these addresses were assigned to Apple Inc.This background traffic indicates that even when idle, iOS devices generate a substantial amount of communication, likely related to synchronization, analytics, or push notification mechanisms. A smaller subset of blocked IPs originated from well-known Content Delivery Networks (CDNs) such as Akamai, Fastly, Cloudflare, and Google LLC, suggesting background calls to embedded services or analytics platforms. Additionally, several blocked connections to Microsoft and Stadt Zürich servers were observed, consistent with the device being a business-issued phone. This test confirmed that a significant volume of traffic is generated by the operating system itself, independent of any user interaction.

**Testing Real Application Scenarios**

To validate the approach in a realistic use case, common messaging applications were tested under the firewall-restricted network.

**WhatsApp:** The WhatsApp application was launched after the device had connected and stabilized, as described earlier. Initially, the interface remained in a loading state, and no new messages appeared. The system detected a blocked IP address: 157.240.0.61, which resolved to Facebook, Inc. (the company behind WhatsApp). After manually adding a PASS rule for this IP address, the app immediately exited the loading state and began retrieving content.

Subsequent interaction with the app revealed three additional Facebook-owned IP addresses, which were also added to the allowlist. Once all four IPs were permitted, full functionality of WhatsApp was restored. The final firewall rules required for full WhatsApp functionality are summarised in Table 8.

Table 8: Manually Added Firewall Rules - Required for WhatsApp Functionality

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Action** | **Source IP** | **Destination IP** | **ISP** | **Location** |
| PASS | 192.168.5.166 | 157.240.0.61 | Facebook, Inc. | Frankfurt, Germany |
| PASS | 192.168.5.166 | 157.240.17.61 | Facebook, Inc. | Zurich, Switzerland |
| PASS | 192.168.5.166 | 157.240.17.60 | Facebook, Inc. | Zurich, Switzerland |
| PASS | 192.168.5.166 | 157.240.27.54 | Facebook, Inc. | Düsseldorf, Germany |

**Telegram:** A similar process was conducted for Telegram. After launching the app, an initial blocked IP, 149.154.167.91, was detected and added to the allowlist. This action enabled the loading of chat content. When attempting to view an image that was not stored locally, another blocked IP, 149.154.167.222, also associated with Telegram, was observed. After allowing this IP, the image downloaded successfully. The IP addresses required to restore full Telegram functionality are summarised in Table 9.

Table 9: Manually Added Firewall Rules - Required for Telegram Functionality

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Action** | **Source IP** | **Destination IP** | **ISP** | **Location** |
| PASS | 192.168.5.166 | 149.154.167.91 | Telegram Messenger | London, United Kingdom |
| PASS | 192.168.5.166 | 149.154.167.222 | Telegram Messenger | London, United Kingdom |

These tests demonstrate that the prototype is capable of supporting application-specific traffic control by dynamically detecting required IP addresses and selectively enabling them. This enables forensic examiners to bring devices online in a controlled, observable, and auditable manner, without granting unrestricted Internet access. Furthermore, the tests confirm that at least one core use case, enabling and monitoring messaging application behaviour, is fully functional with the current implementation.

### System Setup Recommendations After Preliminary Tests

During prototyping and preliminary testing, it became clear that certain baseline settings are necessary to ensure consistent and predictable behaviour when working with mobile devices in the project environment.

**Recommended Settings on the Evidence Device**

* Disable Proxy Services (e.g., Apple Private Relay): These can obscure real network activity and interfere with IP-level tracking.
* Disable MAC Address Randomization: Features that alter the MAC address (such as "Private Wi-Fi Address" on iOS) should be turned off to maintain consistent device identification across sessions.

**Network Configuration**

* The DHCP server should be configured to maintain IP leases for an extended period. This helps prevent reassignment of IP addresses to the same device and reduces the need for frequent adjustments in device management.

### Conclusion of Section 3.2

The successful validation of the proof-of-concept through the described prototyping and testing demonstrated that the core use cases, such as selective application-based access, IP logging, and rule enforcement, can be reliably supported within the intended network environment. These results confirm the technical feasibility of the project and establish a solid foundation for the subsequent implementation phase.

The following sections describe the full system implementation, based on the findings and design decisions established during the prototyping phase.

## Backend Implementation

The system’s backend is organized into three primary functional areas: DHCP, firewall, and logs. Each area is handled by a dedicated module. The underlying data model supports all system logic.

### Backend Data Schema

The system's data model consists of the following tables: DestinationMetadata, Device, DeviceAllowedISP, DeviceLease, DNSRecord, FirewallLog, FirewallRule and MetadataSeenByDevice. These entities and their relationships are illustrated in Figure 3.

For production deployment, PostgreSQL should be used as the primary database backend to ensure scalability, concurrent write support, and reliable transactional behavior. During development and prototyping, SQLite was used due to its simplicity and zero-configuration setup. Django abstracts most database-specific operations, allowing seamless transitions between SQLite and PostgreSQL with minimal effort. Using simple commands such as makemigrations and migrate, the database schema can be initialized or adapted automatically, enabling development with SQLite and deployment on PostgreSQL without code changes.

### api\_dhcp\_parser.py - Lease Synchronization

This module retrieves DHCP lease information from the API. It parses IP addresses, MAC addresses, lease times, hostnames, and other metadata, storing the result in the DeviceLease model. This provides the foundational mapping between devices and their dynamic IP assignments.

### api\_firewall\_sync.py - Device-Based Firewall Enforcement

This module serves as a coordination layer between the system's background logic and view-level components. It encapsulates the decision-making process needed to manage firewall rules dynamically, based on current device states and the list of allowed ISPs. The core responsibilities include:

* Determining the currently active IP address of a device using information from the **DeviceLease** model.
* Referencing the allowed ISPs from the DeviceAllowedISP model to evaluate whether communication with previously blocked destination IPs should now be permitted.
* Deciding whether existing firewall rules should be removed or new rules created.

Based on this logic, the module updates both the local FirewallRule model and the actual firewall configuration on the OPNsense system. Rule changes are executed using API functions provided by the api\_firewall.py module.

### api\_firewall.py - Firewall API Abstraction

Figure 3: Database Model - visualized with dbdiagram.io.

This module is responsible for all direct communication with the OPNsense firewall. It provides a clean abstraction over the firewall API and exposes stateless functions to support essential operations, including:

* add\_firewall\_rule - Adds new firewall rules
* delete\_rule\_by\_uuid - Deletes existing rules by their UUID
* check\_rule\_exists - Checks whether a specific rule is already present
* apply\_firewall\_changes - Applies pending rule changes to the firewall
* source\_ip\_adjustment - Updates the source IP of a rule

The coordination module api\_firewall\_sync.py delegates firewall-related actions to this API module. This separation of concerns allows the core logic to remain focused on rule evaluation, without having to manage the details of HTTP requests or API communication.

### api\_logs\_parser.py - Log Parsing and IP Enrichment

This module is responsible for parsing firewall logs and enriching destination IPs with metadata such as ISP, geolocation, and DNS records.

It continuously fetches and processes firewall logs from OPNsense, extracts relevant entries, enriches destination IP addresses using reverse DNS lookups and queries to ip-api.com, and stores structured log data into the FirewallLog model.

Additionally, it maintains an in-memory cache config.IP\_TABLEfor fast enrichment lookups and updates the DestinationMetadata and MetadataSeenByDevice tables to track visibility and relationships between metadata and devices.

### ssh\_dns\_parser.py – Live DNS Parsing

This module is responsible for parsing of DNS requests and responses directly from OPNsense’s Unbound DNS service using tcpdump over SSH. It stores structured DNS resolution events in the DNSRecord model. Since Unbound DNS does not expose an API for DNS query logging, a separate strategy was implemented. As a result, DNS parsing is handled in a dedicated, standalone module, ensuring that real-time resolution data can be captured and analysed without modifying the firewall system.

**Functionality overview:**

* Establishes a remote SSH connection to the OPNsense firewall and launches a persistent tcpdump session on interface igc1 to monitor DNS traffic:  
  tcpdump -i igc1 port 53 -n -l
* Parses incoming DNS queries and responses in real time.
* Temporarily buffers unresolved DNS queries in memory, using a short expiration time to manage memory usage.
* Matches incoming DNS responses to previously observed queries.
* Extracts resolved DNS information and stores it in the DNSRecord model.
* Updates the last\_seen\_at timestamp for known records when they are observed again.

The following Figure 4 illustrates a real-world DNS resolution captured by the dns\_ssh\_parser.py module for an Azure cloud domain:

Figure 4: Simplified representation of DNS request and response data as provided by the DNS server.

Request: 192.168.5.155.49352 > 192.168.5.1.53: 6838+ A? onedscolprdeus19.eastus.cloudapp.azure.com. (60)

Answer: 192.168.5.1.53 > 192.168.5.155.49352: 6838 1/0/0 A

52.168.117.175 (76)

Stored: 192.168.5.155:49352 > 52.168.117.175

onedscolprdeus19.eastus.cloudapp.azure.com.

## Frontend Implementation

The frontend design process began with a simple hand-drawn sketch to visualize the intended layout and interaction flow. It quickly became evident that the interface should be divided into several logical sections: a header, a main content area (used for displaying logs and metadata), a side panel for grouped ISP views, and later, a footer for general information. The header and footer were separated from individual view templates and are included across all pages to ensure a consistent layout.

The frontend is implemented using Django’s templating system, combining standard HTML and CSS with Django views and models. An overview of the dashboard layout is shown in Figure 5.

The header contains two primary categories:

* **Views:** Includes data-driven interfaces such as log overviews (blocked or allowed traffic) and grouped ISP-based views.
* **Tools:** Provides access to interactive utilities for managing evidence devices, viewing DHCP leases, performing IP lookups, and manually adjusting firewall rules.

A device selection dropdown is placed on the left side of the header. This dropdown allows the user to filter all views and actions based on the selected device. Importantly, the system does not offer a global view of all network traffic across devices. All traffic analysis and log inspection are scoped to a specific device, meaning that interaction with traffic data is only possible after selecting an individual device from the interface.

### Technologies Used

The frontend is built entirely with Django’s server-side rendering approach. It uses:

* Django Templates for HTML generation
* Django Admin for base database view and management
* Custom Django views for interactive, user-driven logic and API integration

During early testing in the Python console, the use of visual feedback symbols such as ✅, ⚠️, and ❌ proved highly effective. These icons offered immediate and intuitive feedback about the system status, particularly when debugging in the Django shell. Their use was carried over to the web interface, where they continue to enhance usability by showing system states more clearly than plain text or numeric codes.

See Figure 5 for an overview of this setup, including the Blocked Logs View, which illustrates how blocked connections are captured and displayed.

### Backend Integration

All frontend views are implemented as Django function-based views and are tightly integrated with the backend logic described in Section 3.3:

* The Log Viewer retrieves enriched data from the FirewallLog and DestinationMetadata models. These are updated every few seconds with newly parsed log entries. IP enrichment, such as organization, geolocation, and DNS hostname, is performed in a background thread to ensure metadata becomes available shortly after the log entry is recorded.
* Actions in the “Organisations (ISP)” side panel trigger functions in api\_firewall\_sync.py, which in turn calls api\_firewall.py to apply, read, update or delete firewall rules via the OPNsense API.
* Device metadata and lease information are dynamically linked through the DeviceLease model, enabling real-time device identification and rule enforcement.

This integration ensures consistency between the frontend and backend, enabling forensic examiners to work with near real-time network data.

### Functional Views

The frontend interface is composed of several specialized views that provide targeted functionality for forensic analysis and firewall management.

### Blocked View

This view displays all blocked network connections associated with the currently selected device. Each log entry is presented in a single- line format and includes the following fields:

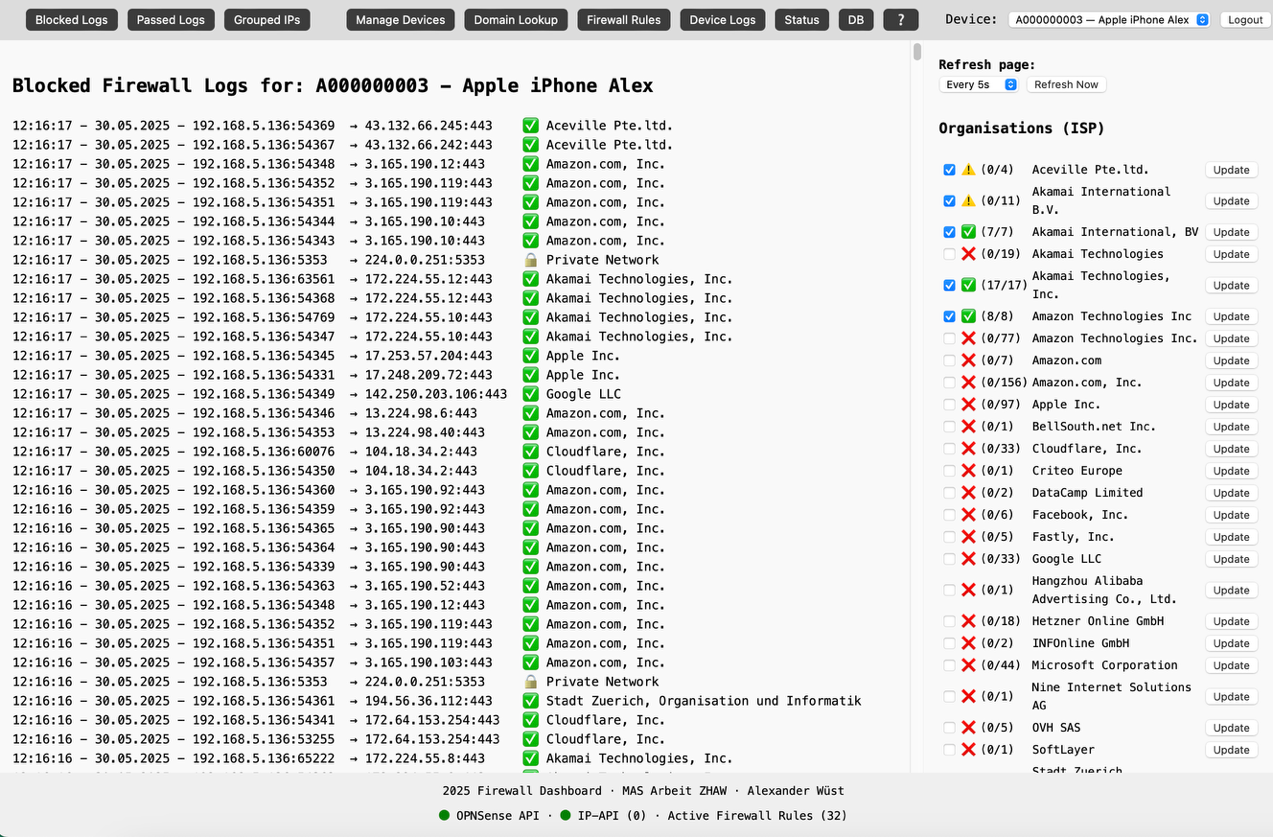
* Timestamp and Date of the blocked event
* Source IP Address of the device
* Destination IP Address that was blocked
* Enrichment Details: If metadata is available, this field displays the associated organization or isp (e.g., Amazon.com, Inc.). Otherwise, a placeholder such as "Unknown, lookup pending" is shown while enrichment is still in progress.

Figure 5: Frontend interface – Blocked Logs View.

This view enables the examiner to identify which outbound connections were attempted and whether they relate to known services. On the left-hand side panel, an Update dropdown allows users to configure automatic refresh intervals for the page, ensuring that log data remains up to date without requiring manual intervention. Alternatively, a manual refresh button is available to immediately reload the view on demand. Just below the update controls, users can select or deselect checkboxes next to each listed ISP in the sidebar to define which ISPs should be allowed for the currently selected device. These checkbox selections are not enforced immediately but are directly saved to the DeviceAllowedISP model. When the “Update” button next to a specific ISP is pressed, the backend is triggered to evaluate the corresponding changes and update the firewall configuration accordingly by adding or removing the relevant rules. Additionally, a "Update All ISP Rules!" button is available at the bottom of the sidebar. This triggers a full evaluation of all selected ISPs and their related IP addresses. This process can take noticeable time to complete, particularly when the system must evaluate and apply changes to hundreds of destination IP addresses through the OPNsense API. To improve clarity regarding rule enforcement status, visual indicators (icons) have been added next to each ISP entry. The number next to each ISP entry (e.g., 3/12) represents the currently applied firewall rules in relation to the total number of known destination IPs associated with that ISP. For example, 3/12 indicates that only 3 out of 12 relevant IP addresses are currently permitted by the firewall. This allows forensic examiners to quickly assess whether additional rules still need to be enforced or revoked, based on the state of the associated checkboxes. This functionality was added following direct feedback from an initial system demonstration conducted in the forensic laboratory and reflects the need for clear, actionable insights during live device analysis.

### Passed View

The Passed View is functionally based on the same structure as the Blocked Logs View but instead displays all allowed traffic entries. The primary purpose is to provide an overview of permitted network activity, offering insights into which external services or destinations the device is actively communicating with. Each entry includes timestamp, source and destination IP addresses, and the metadata enrichment field ISP. This view allows forensic examiners to verify that only explicitly allowed traffic is passing through the firewall and to monitor for any unintended or unexpected communication patterns.

### ISP View

The ISP View (see Figure 6) differs significantly from the previously described log views. It is designed to present a live snapshot of recent IP communications, categorized by their enriched metadata, particularly the ISP field.

**Upper Section – Unfiltered IPs (Blue)**

The upper (blue) section displays all IP addresses observed within the last N seconds. The duration of this time window can be selected via a dropdown menu, allowing the examiner to define the observation interval. A “Refresh” button reloads the view manually, while a “Delete Seen Metadata” button clears the current list to start with a clean view.

Each IP entry includes:

* First Seen and Last Seen timestamps, reflecting the time window during which communication (or connection attempts) occurred.
* Basic metadata, enabling live observation of device behaviour.

This view is particularly useful for real-time monitoring. Forensic examiners can allow a device to run for a short period, then initiate app usage to observe the resulting network connections as they appear in near real time.

**Lower Section - Grouped by ISP (Orange)**

The lower (orange) section organizes destination IP addresses by ISP and provides a sortable table with enriched metadata for each IP entry. The following columns are included:

* ORG: The organization name associated with the IP.
* ISP: The Internet Service Provider associated with the IP address. This field is broader and more general than the ORGfield, which may reflect a specific service or business unit.
* DNS Request: The hostname used to resolve the IP, shown with a tooltip for additional resolution context.
* Location: Geolocation information (e.g., country, region).
* First Seen: Timestamp of the initial observation.
* Last Seen: Timestamp of the most recent communication.
* DNS Reverse: The reverse DNS name (if available).

Each row features a control button on the left side that enables direct management of firewall rules for the given IP. The button's icon reflects the rule state:

* **+** indicates that no rule is currently applied (clicking adds a PASS rule).
* **–** indicates that a rule is already active (clicking removes the rule).

This mechanism provides intuitive, hands-on control over individual IP addresses, allowing forensic examiners to incrementally build or refine rule sets as network behaviour is observed.

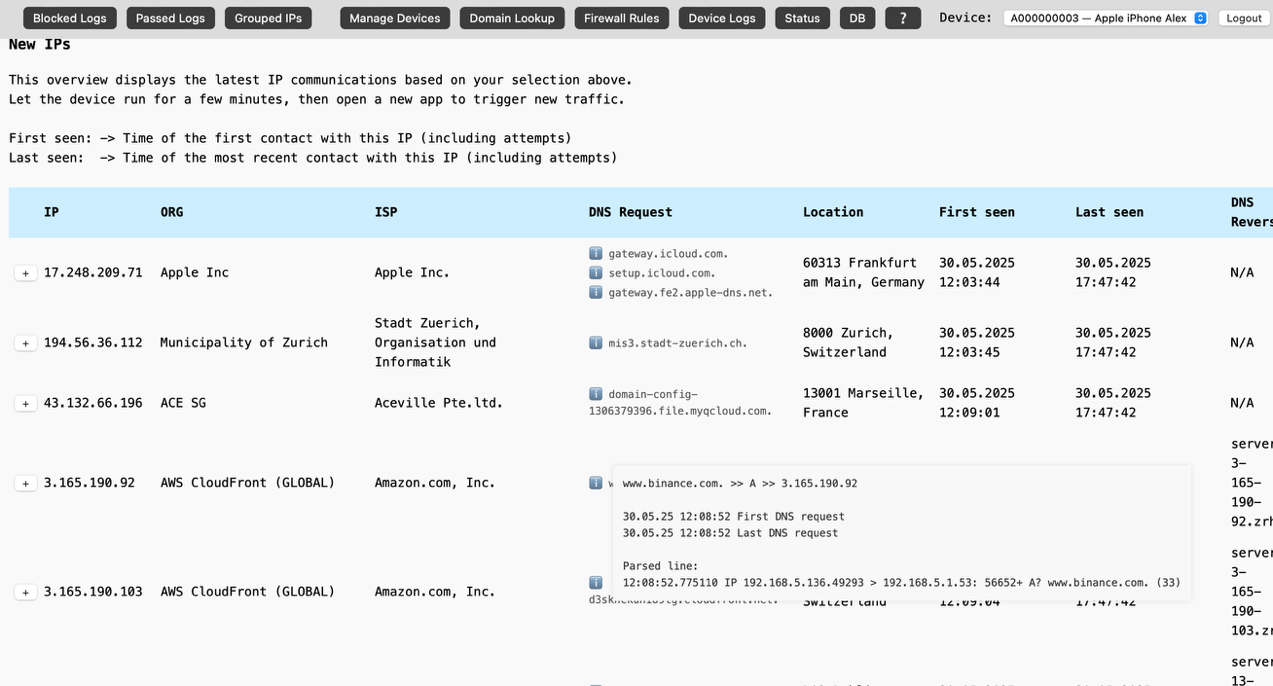


Figure 6: Frontend interface – Grouped IP's View

### DNS View

The DNS View provides a structured interface for analysing traffic based on the DNS queries by a specific device. It is grouped by the resolved destination IP address and relies on the ssh\_dns\_parser.py module, which monitors DNS traffic in real time via tcpdump over SSH. Both DNS requests and responses are stored in the DNSRecord model.

Each group in the view represents a destination IP and lists all associated DNS queries that resolved to it. For every DNS record, the following information is displayed:

* Timestamp of the query
* Query type (e.g., A, AAAA, CNAME)
* Domain name Domain name requested
* Source IP of the device making the request
* Resolved IP (i.e., destination IP returned in the DNS response).

A real-time search field allows the user to filter records by keyword, and a checkbox-based selection system enables the creation of firewall rules based on selected DNS entries. The “Mark all” / “Unmark all” function applies only to currently visible entries, allowing focused interactions even within large datasets.

Icons next to each resolved IP indicate whether firewall rules are currently applied for that destination. The visual feedback helps the examiner decide which DNS-based connections should be allowed or blocked, with direct integration into the rule management system via form submission.

This view enhances forensic transparency by offering full visibility into the DNS behaviour of a device, including passive lookups initiated by applications and background services.

### Manage Devices View

The Manage Devices view plays a central role in configuring which devices are actively monitored by the system (see Figure 7). It provides essential functionality for adding, archiving, and restoring devices, as well as assigning or hiding DHCP leases. A device must have at least one active lease assigned in order to enable log analysis and firewall rule enforcement within the frontend views.

**Adding and Approving Devices**

New devices are registered by submitting the following metadata:

* Device ID: A unique identifier assigned to the device
* Description: A brief label (e.g., Samsung Galaxy S22)
* Examiner: The forensic analyst responsible for the device
* DNS Server: The resolver used for DNS-based rule evaluation and enforcement

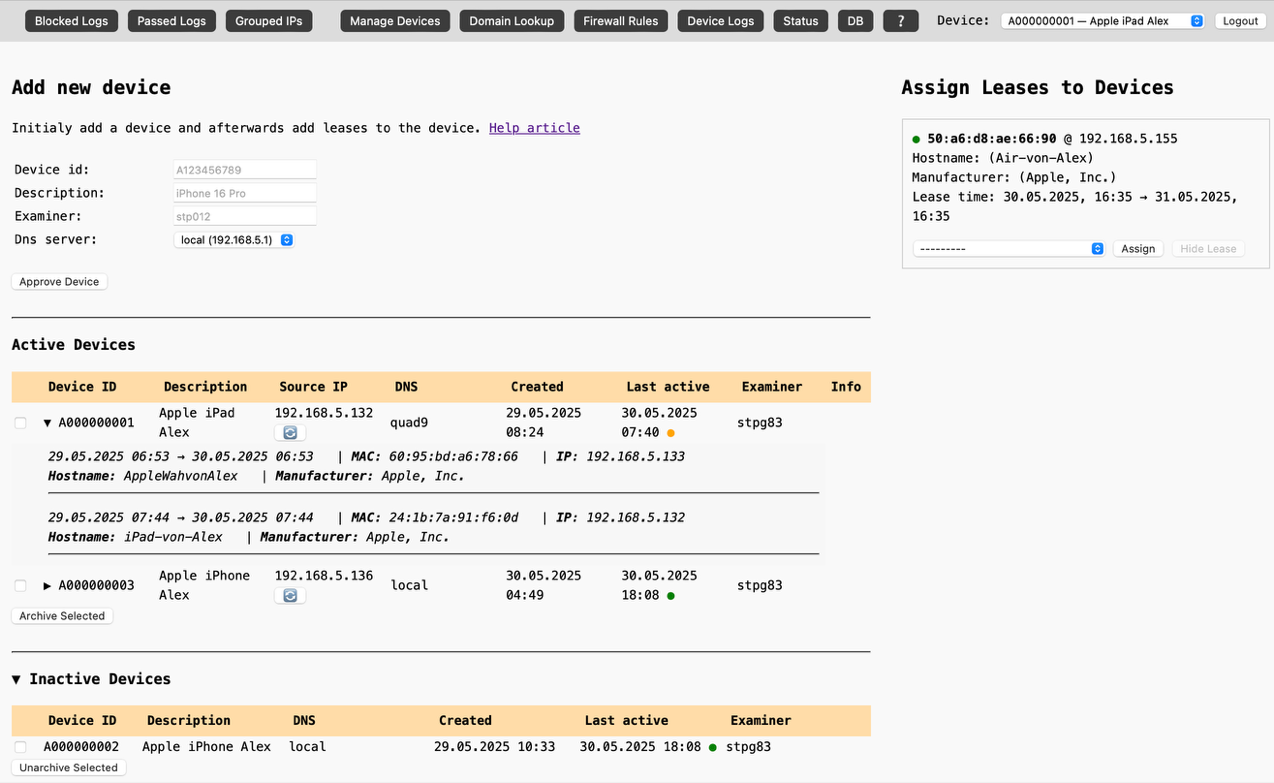
Devices not marked as archived are listed under Active Devices. Currently assigned network leases can be viewed and managed by expanding the dropdown menu associated with each device entry. For reliable MAC address tracking, it is recommended to disable the Private WLAN Address feature on iOS devices. This ensures the use of a consistent hardware MAC address across network sessions. A contextual help section is linked directly on the page to provide guidance on this setting.

Figure 7: Frontend interface – Device Management View

**Device Lists**

The interface categorizes devices into two groups:

* **Active Devices**: Displayed in the dropdown menu for lease assignment and used to filter log and traffic views.
* **Inactive Devices**: Archived devices that no longer appear in live monitoring views.

Each device entry displays relevant metadata, including Device ID, Description, DNS server, Creation timestamp, Last active timestamp, Examiner, and Info. Devices can be archived or restored (unarchived) as needed. When a device is archived, all associated firewall rules are automatically removed. This helps maintain a lean and manageable firewall rule set and ensures continued responsiveness of the OPNsense API by limiting active rule overhead.

**Assign Leases to Devices**

The right-hand panel lists all currently unassigned DHCP leases. Each lease entry contains the following attributes:

* IP address
* MAC address
* Hostname (if available)
* Manufacturer (if available)
* Lease time range (start and end timestamps)

Each lease can be assigned to only one device, linking it in the backend to the corresponding entry in the Device model. By linking each lease to a device, the system ensures that all related network activity is properly displayed under the correct device in the interface. Leases that are no longer relevant, such as expired or inactive entries, can be hidden from the list using the Hide Lease button.

### Domain Lookup View

The Domain Lookup view offers an interactive interface for resolving domain names into their associated IP addresses, along with the corresponding ISP metadata field. This functionality supports manual investigation by enabling forensic examiners to identify the underlying infrastructure and service providers responsible for a given domain. For example, resolving the domain binance.com returns a set of IP addresses and their enrichment details, as illustrated in Table 10. This information can assist in determining whether related network connections should be permitted or blocked through the firewall.

Table 10: DNS lookup for binance.com via DNS Lookup View

|  |  |
| --- | --- |
| **Source IP** | **ISP** |
| 54.64.24.218 | Amazon.com, Inc |
| 3.112.157.195 | Amazon Technologies Inc. |
| 13.114.29.123 | Amazon Technologies Inc |

### Firewall Rules View

The Firewall Rules view displays all active PASS rules associated with the currently selected device. As the system operates under a default-deny policy, only explicitly permitted connections appear here. These rules may have been generated automatically, based on ISP-based logic, or manually added by the examiner.

Each entry includes the following attributes:

* Action (always "PASS")
* Protocol, Source IP, Destination IP, and Port
* Associated ISP (if available)
* Timestamp indicating when the rule was added
* Flags identifying whether the rule was created manually or as a result of DNS-based resolution
* Delete button for removing eligible rules

Forensic examiners can selectively remove rules via this interface. However, rules derived from DNS-based logic are protected from deletion to preserve continued domain resolution. Since external DNS servers are no longer permitted within the system configuration, the creation of DNS-based firewall rules is currently inactive. However, the underlying logic has not been removed from the implementation, preserving the option to re-enable DNS-based rule generation should future requirements require external DNS resolution.

### Device Logs View

The Device Logs view provides a comprehensive, timestamped record of all observed network activity for the currently selected device. Each log entry includes the following fields:

* Timestamp
* Action (e.g., BLOCK or PASS)
* Interface
* Source IP
* Destination IP
* Protocol
* Enriched ISP and Country information (if available)

To enhance readability, entries are color-coded: red for blocked traffic and green for allowed traffic, enabling immediate visual distinction between permitted and denied connections. A planned feature will introduce device-specific log export functionality, allowing forensic examiners to generate complete traffic records for individual devices. This export will include a timeline-based visualization, correlating firewall rule changes (additions and removals) with associated network activity. This feature is intended to improve traceability and support thorough forensic analysis.

## Integration with OPNsense

All communication with the OPNsense firewall is handled via its REST API, enabling remote creation, removal, and management of firewall rules, as well as access to log data and DHCP lease information. The backend communicates with the firewall over HTTPS using Python’s requests library. To ensure modularity and separation of concerns, all API interactions are encapsulated within the api\_firewall.py and api\_dhcp\_parser.py modules.

**Authentication and Security:** Secure access to the firewall is achieved through two complementary mechanisms:

* **API credentials:** The API\_KEY and API\_SECRET are stored in a local .env file and accessed via Django's environment configuration system.
* **Client Certificate:** A certificate file certificate\_crt.pem is used authenticate the client and establish a secure connection with the firewall.

Each API request includes both HTTP basic authentication credentials and the client certificate. Using environment variables and Django's settings module ensures sensitive information is not hardcoded and remains easily configurable.

**Integrated OPNsense API Endpoints:** The following OPNsense API endpoints are actively used by the system:

* **Add Rule:** /api/firewall/filter/addRule
* **Delete Rule:** /api/firewall/filter/delRule/{uuid}
* **Search Rules:** /api/firewall/filter/searchRule
* **Apply Changes:** /api/firewall/filter/apply
* **Retrieve DHCP Leases:** /api/dhcpv4/leases/searchLease
* **Fetch Logs:** /api/diagnostics/firewall/log/search

All of these endpoints are accessed through reusable helper functions defined in the api\_firewall.py module, promoting consistency and maintainability in API interactions.

**Optimized Request Flow and Performance:** To reduce overhead, the module maintains a shared requests.Session() instance with preconfigured authentication headers. This approach eliminates redundant session setup for each individual API request, resulting in more efficient communication with the OPNsense firewall. For diagnostic purposes, verbose logging can be enabled with configurable verbosity levels. While full debugging output may become excessive under high-traffic conditions, the logging system has been modularized to support targeted debugging. For instance, DNS-related logs can be enabled independently of other subsystems, allowing focused inspection without overwhelming the console output.

**Error Handling and State Consistency**

All API requests are enclosed in requests.RequestException blocks to gracefully handle network errors and avoid runtime crashes. Discrepancies between the firewall's actual rule set and the internal database representation are automatically reconciled. When inconsistencies are detected, the internal system state is updated to reflect the most current known configuration from OPNsense. A key mechanism in this process is the verify\_opnsense flag, which is triggered for any rule changes that may require direct validation. If a mismatch between the database and firewall state is detected, the affected rule is removed, following the principle that removing a potentially incorrect rule is safer than allowing unintended access. To maintain ongoing consistency, all existing rules are periodically fetched from the firewall and compared to the local database using their unique UUIDs. If a rule exists on the firewall but is no longer tracked in the database, it is automatically removed. This hybrid verification strategy combines a full synchronization sweep with targeted rule-level validation. It ensures a high level of integrity across both systems while minimizing unnecessary API calls, resulting in a robust and performant rule management workflow.

## Rule Application Logic

The system implements dynamic and policy-driven firewall rule management through a coordinated backend process. Rules are not static, they are created, updated, or removed based on device state. This logic is centralized in a coordination function that interacts with both the Django database and the OPNsense firewall.

**Two Entry Points for Rule Creation**

Firewall pass rules can be introduced through two primary mechanisms:

* **Policy-based ISP Grouping:** When a forensic examiner authorizes one or more ISPs for a specific device, the system automatically identifies all previously blocked IP addresses associated with those ISPs. It then generates and applies corresponding pass rules to permit traffic to those destinations. This ensures continuity of access for approved services while maintaining prior restrictions.
* **Manual Rule Addition in the Frontend:** Within the Grouped IPs View, users have the option to manually add or remove pass rules for individual IP addresses. This provides granular control, allowing exceptions to the general ISP-based policy, for example, to authorize or deny a specific address regardless of its associated ISP.

**Source IP Handling and Automatic Rule Reassignment**

In dynamic DHCP environments, devices often receive new IP addresses over time. To maintain consistent enforcement of firewall policies, the system automatically adjusts existing rules when a device’s source IP changes. For instance, if a device transitions from 192.168.5.155 to 192.168.5.170, the backend coordination logic performs the following actions:

* Detects the updated lease assignment through the DHCP lease parser
* Reassigns all active pass rules previously associated with the old IP to the newly assigned IP
* Updates both the firewall configuration and the corresponding database entries to reflect the change

This automatic realignment ensures that rules remain effective even if the IP address changes. No need for manual cleanup or reconfiguration.

### Rule Evaluation and Creation Logic

The system determines which IP addresses should be permitted based on a structured evaluation process:

* 1. The currently active IP address of the device is identified via the DeviceLease model.
  2. The list of user-approved ISPs associated with the device is retrieved.
  3. All previously blocked destination IP addresses linked to those ISPs are collected.
  4. For each of these destination IPs the system checks whether a corresponding rule already exists in the database. If no such rule exists, a new entry is created in the FirewallRule model, and an API call is triggered to add the rule to the OPNsense firewall.

To preserve data integrity, a firewall rule is only recorded in the database if its creation on the firewall was successful.

### Device Archival and Rules Removal

When a device is archived:

* All active firewall pass rules associated with the device are removed from OPNsense.
* These rules are marked as terminated in the database by assigning an end\_date.
* The device is removed from the header dropdown menu to prevent further interaction.

Upon unarchiving a device:

* No firewall rules are automatically re-applied.
* The device reappears in the header dropdown, making it selectable for future actions.

This approach ensures that inactive devices do not accumulate stale firewall entries, helping to reduce rule processing overhead and improving system performance.

### Rule Cleanup and Removal

Following the addition of new pass rules, the system performs a consistency check on all currently active (non-manual) firewall rules associated with the device. It identifies any rules targeting destination IP addresses that are no longer associated with an approved ISP.

Such rules are queued for removal and processed in a batch operation:

* The rules are first removed from the OPNsense firewall using the delete\_multiple\_rules() function, which allows for efficient bulk deletion.
* Once deleted, the rules are marked as ended in the database by updating their end\_date field.
* Finally, apply\_firewall\_changes() is invoked to commit the changes on the firewall, ensuring the updated rule set takes effect.

This automated cleanup mechanism maintains alignment between user-defined ISP policies and the actual firewall configuration.

## Logging

The system does not implement a separate or dedicated logging layer. Instead, all log information is structured and stored directly within the Django-managed database. This design enables consistent, query able access to historical network activity without the need for additional infrastructure. The system focuses solely on IP-level metadata. It does not perform deep packet inspection or analyse traffic content. Its primary function is to document which destination IP addresses were contacted by a monitored device, and whether those attempts were permitted or blocked by the firewall.

### Purpose and Scope

The logging functionality supports a controlled network exposure model for connected devices. Instead of permitting full Internet access, the system blocks all traffic by default and selectively allows communication defined by the forensic examiner. By capturing the resulting connection attempts and enforcement actions, the system enables forensic examiner to answer key questions, such as:

* When was a specific firewall rule created or removed?
* Which IP addresses did a device attempt to contact, and were those connections allowed?
* Which ISP or organization was associated with a given IP address?

### Database-Backed Structure

All logs and related metadata are persistently stored in the database:

* FirewallLog: Stores all observed connection attempts (both allowed and blocked), enriched with IP-level metadata.
* DestinationMetadata: Contains enrichment results for destination IPs, including ISP, geolocation, DNS, and organizational ownership.
* FirewallRule: Records applied firewall rules, along with timestamps for creation and removal.
* MetadataSeenByDevice: Tracks which devices have communicated with specific destination IPs, supporting grouping and time-based queries.

This structure enables full historical reconstruction of a device’s network activity and the enforcement logic behind it.

### Operational Visibility

Frontend views such as Blocked, Passed, ISP and DNS rely entirely on this stored data. Color-coded entries indicate blocked (red) and allowed (green) connections in the Logs view. The examiner can see which IPs were recently contacted, whether those IPs were previously seen, and which rules were in effect at the time.

Because logs are stored in the database rather than relying solely on OPNsense’s volatile log storage, the system can offer long-term retention, complex queries, and device-specific views that extend the capabilities of the firewall itself.

### Planned Enhancements

Future improvements to the logging system are planned to support advanced forensic analysis and reporting needs:

* Export of rule histories per device, including exact timestamps for rule application and removal
* Export of connection logs, both blocked and allowed, for auditing and evidence documentation
* Firewall configuration snapshots, allowing examiners to correlate rule states with observed behaviour at specific points in time

These enhancements aim to improve traceability, support reproducibility of findings, and enable comprehensive post-incident review.

## Challenges Encountered

During development and testing, several technical challenges were encountered that influenced the system’s final design and implementation. These challenges were primarily rooted in limitations of IP-based control, mobile device behaviour, and external API dependencies. The following section summarises the most relevant problems and the corresponding solutions or workarounds.

### Granular App Control via IP and Firewall Rules

Some applications, such as WhatsApp, Telegram, or Binance, rely on large networks of rotating IP addresses, many of which are partly hosted by content delivery networks (CDNs) like Akamai, Amazon or Cloudflare. Initially, the intention was to manually approve only a minimal set of IPs per app. However, due to the lack of visibility into which IPs were required for specific functionality, this approach proved infeasible.

To mitigate this, the system adopted a more general strategy by allowing entire ISP groups. For instance, once Akamai was marked as allowed for a device, all related IPs were permitted without requiring detailed per-IP analysis. This provided sufficient control without the need for deep packet inspection or reverse-engineering of app behaviour.

### IP Enrichment Bottlenecks

The system uses ip-api.com to enrich destination IPs with metadata such as isp, org, and location(lat/lon). However, the API's public endpoint is limited to 45 requests per minute, which led to delays in processing logs and updating the frontend with enriched data. Real-time IP enrichment created a bottleneck during log parsing and delayed frontend updates when many new IPs appeared in a short timeframe.

**Enrichment Asynchronously:** Enrichment was decoupled from the main parser by moving the logic into a standalone ip\_enrichment.py module. This avoided blocking log ingestion during slow API responses.

**Memory-first Caching**: A runtime memory cache config.IP\_TABLE was introduced to store enrichment results temporarily. If an IP was enriched within the last 96 hours, it would be skipped without querying the database or the API.

**Database Fallback**: If an IP was not found in memory, the system checks the database. Only if no valid result (or an outdated one) is found will the external API be queried.

**IP Filtering**: Non-routable or special-use IP ranges (such as private networks, loopback, multicast, and reserved blocks) are now explicitly skipped before the enrichment request is done. This avoids unnecessary lookups for addresses that are not public traceable.

**Rate Limiting**: The enrichment process enforces a fixed delay of 1.4 seconds between API calls. Testing confirmed that this interval stays safely below the 45-requests-per-minute threshold, preventing the API from rate-limiting the system. These combined strategies significantly reduced the number of enrichment requests and improved the system's responsiveness. Once the in-memory cache is populated, most enrichment queries do not require database or network access, making the system efficient even under higher log volume.

### Firewall Rule Propagation Delay

When applying a large number of rules via the OPNsense API, for example hundreds of destination IPs after toggling an ISP group, the system experienced notable delays. Rule creation could take several minutes, during which the frontend appeared unresponsive, and the firewall was not yet enforcing the changes.

**Session reuse and backend optimizations**: helped reduce request overhead, but OPNsense does not support true rule batching via API. As a result, rule application remains a sequential process and is time-consuming in large-scale updates. An alternative method using Aliases, as demonstrated in section 4.3, significantly improves performance by consolidating hundreds of IPs into a single rule.

### MAC Address Randomization and IP Variability

Modern devices, particularly iOS and Android, frequently change both MAC and IP addresses for privacy reasons. This caused problems when trying to maintain a consistent mapping between a physical device and its current network identity. To resolve this, the data model was expanded with the DeviceLease model which includes ip\_address and mac\_address with timestamps and device associations. This made it possible to track devices over time, even when their network identifiers changed.

## Summary

This chapter presented the detailed implementation of the prototype system developed as part of this thesis. Building upon concepts validated through initial prototyping, the solution was refined into a modular Django-based application that interfaces with the OPNsense firewall via its official API. Key components of the system include a dynamic backend for rule enforcement, IP enrichment and log parsing, a persistent database-backed architecture for state management, and a responsive frontend designed to provide forensic examiners with near real-time visibility and control over device-specific network activity. Several challenges were addressed during development, including variability in device behaviour (e.g., MAC and IP address changes), adherence to external API rate limits, and maintaining performance during the application of large rule sets. Despite these constraints, the system successfully demonstrates that fine-grained, IP-based firewall control of mobile devices is technically feasible, without relying on deep packet inspection or invasive monitoring techniques.

# Analysis and Results

This section takes a closer look at the behaviour of iOS and Android devices in relation to remote wipe prevention. The ring command is used as a proxy for the actual erase function, as both appear to rely on nearly identical infrastructure. The question is whether it is technically feasible to allow core device functions, such as iCloud access or messaging, while blocking wipe-related traffic. It is still unclear how tightly these services are integrated, and whether a targeted filtering approach can reliably prevent undesired commands. To validate this, each previously defined use case (WhatsApp, Telegram, iCloud Photos, Session, Binance) will be revisited under slightly more "exciting" conditions. For every test, a remote wipe command will be deployed in parallel to simulate a high-risk forensic scenario. The goal is to determine whether the solution can still deliver app-level access without triggering data loss. During this section, the pressure felt noticeably higher, each rule added to the test devices carried more weight than in earlier phases.

## Prevent Remote Wipe on Mobile Devices

Before validating individual use cases, it is essential to briefly address the risk of remote wipes. Both iOS and Android support mechanisms that can trigger a full device erase over the Internet. In forensic scenarios, connecting a device to the Internet may unintentionally permit the delivery of remote wipe commands. To mitigate this, all tests in this chapter were designed with strict firewall controls, allowing only selected IP traffic. This setup aims to permit specific functionalities like app access or cloud sync while blocking the broader infrastructure that might carry a wipe signal.

### iOS Remote Wipe Observations

DNS and IP-level testing revealed that a significant portion of Apple’s service infrastructure operates within the 17.0.0.0/8 IP address block, which is owned entirely by Apple Inc. This range contains services fundamental to the functioning of iOS devices, including:

* iCloud access and synchronisation
* Device backup and restore
* Find My iPhone
* Activation Lock and device management

Within this broader allocation, the 17.248.209.0/24 subnet emerged as particularly critical. Numerous essential services, including those related to iCloud access and remote wipe functionality, consistently resolved to addresses in this range. This suggests that certain subnets within Apple’s infrastructure may serve as centralized endpoints for sensitive device commands, including wipe or ring signals. Understanding the role of this subnet is essential for targeted filtering strategies. Allowing partial access to Apple infrastructure (e.g., to enable iCloud Photos) without also enabling communication with wipe-related endpoints requires granular control.

For instance, DNS queries that resolved to the 17.248.209.0/24 subnet included:

* fmipmobile.fe2.apple-dns.net.
* mobilebackup.fe2.apple-dns.net.
* fmfmobile.fe2.apple-dns.net.
* fmf.fe2.apple-dns.net.
* p138-contacts.icloud.com.
* p41-contacts.icloud.com.
* contacts.fe2.apple-dns.net.
* p42-fmf.icloud.com.
* fmip.fe2.apple-dns.net.
* p42-fmfmobile.icloud.com.
* setup.fe2.apple-dns.net.
* quota.fe2.apple-dns.net.
* p42-mobilebackup.icloud.com.
* acsegateway.fe2.apple-dns.net.
* mask-api.fe2.apple-dns.net.
* gateway.fe2.apple-dns.net.
* p42-acsegateway.icloud.com.

Initial testing revealed that without allowing the 17.248.209.0/24 subnet, core iCloud services, including Find My iPhone functionalities such as remote ring and wipe, remain inaccessible. This subnet need to be explicitly permitted for these features to function. However, further tests showed that even when the subnet is open, selectively blocking certain DNS entries can prevent delivery of ring or wipe commands. Specifically, DNS queries to endpoints such as

* p42-fmipmobile.icloud.com
* p42-fmf.icloud.com
* fmf.fe2.apple-dns.net
* fmfmobile.fe2.apple-dns.net
* fmip.fe2.apple-dns.net
* p124-fmip.icloud.com

appear to be directly related to Find My iPhone functionality. When these domains are blocked, no wipe or ring commands were received by the device, despite successful connectivity to other iCloud services. It was also observed that these DNS requests are not immediately issued by the device. Instead, they seem to be triggered only after resolving certain gateway domains, such as gatewayws.icloud.com, indicating the use of a layered or geographically-aware routing mechanism. IPs tied to this process, such as 17.248.209.73, were geolocated to infrastructure in Frankfurt am Main, Germany, supporting the hypothesis of region-specific routing by Apple. These findings underscore the necessity of ongoing validation, especially across different iOS versions, as the specific DNS patterns and behavior may vary. Initial experimentation using non-evidentiary test devices is recommended to identify DNS queries associated with wipe-related services under varying conditions. In summary, it appears feasible to enable access to most iCloud services while suppressing the wipe/ring command. This strategy could significantly mitigate the risk of unintentional remote data loss, even under conditions of broad IP-level whitelisting at the firewall interface.

### Android Remote Wipe Observations

Following the iOS observations, the Android test device was examined under similar conditions to understand how remote wipe and ring commands are delivered, and whether selective network filtering could suppress them. Unlike Apple’s infrastructure, where a distinct and critical subnet could be identified (e.g., 17.248.209.0/24), Android did not reveal a similarly clear-cut pattern. Initial DNS and IP analysis of the Android device showed no uniquely identifiable domains or address blocks related to wipe/ring delivery. In contrast to Apple’s structured infrastructure, Android appeared to rely more diffusely on general-purpose Google services. All outbound connections associated with Google LLC were whitelisted to assess whether remote wipe or ring commands would arrive. Even under these permissive conditions, the commands were not received, prompting deeper investigation. Sometimes it looked like a CDN was responsible, and later the same test seemed to involve a different company, resulting in misleading conclusions. A surprising finding was that remote ring signals on Android were subject to substantial delays, with delivery times ranging from 15 to 30 minutes. Through trial and error, and gradual whitelisting of DNS traffic, these specific domains emerged as critical:

* talk.google.com
* mtalk.google.com
* alt1-mtalk.google.com
* alt2-mtalk.google.com
* alt3-mtalk.google.com
* alt4-mtalk.google.com
* alt5-mtalk.google.com
* alt6-mtalk.google.com
* alt7-mtalk.google.com
* alt8-mtalk.google.com
* alt9-mtalk.google.com

These domains, used for Google’s messaging infrastructure (including Firebase Cloud Messaging), appeared to be directly responsible for handling remote management commands. It was also noted that the corresponding IP addresses consistently ended in .188, suggesting potential internal load-balancing or service designation mechanisms. Once these domains were blocked on the DNS level, all other IPs associated with Google LLC could be safely permitted without triggering a wipe or ring event. The device remained accessible and functional even after 24 hours of continuous observation. Interestingly, Samsung Find My Mobile feature appeared to rely on the same backend infrastructure, with delivery of remote commands also routed through Google’s network. No additional DNS requests or alternative communication paths were observed. These findings suggest that, similar to iOS, Android’s wipe and ring functionality can be selectively suppressed without fully blocking general cloud connectivity. By isolating and filtering talk.google.com, investigators may reduce the risk of data loss. It should be noted that the behavior of talk.google.com may vary across Android versions and OEM customizations (e.g., Xiaomi). Future Android updates could modify how remote wipe commands are delivered, highlighting the need for continuous validation to ensure continued protection.

However, these observations only enhance protection against wipe or ring commands issued directly by iOS or Android. They do not cover remote actions triggered via third-party MDM solutions, as discussed in 2.4.2 and 2.4.3

## Practical Use Case Validation

This section presents practical demonstrations illustrating how the implemented system operates under real-world conditions. The goal is to validate the system's ability to selectively allow network communication for mobile devices. All use cases were conducted using standardized device configurations and within a consistent test environment. Each test scenario focuses on the interaction between a device and a specific application.

The structure of each use case includes:

* A clearly defined objective
* A detailed procedure outlining the steps taken
* The resulting system behaviour and observations

The following sections describe the test environment and device preparation. This is followed by several representative use cases, which were defined in consultation with forensic colleagues. The aim of these discussions was to identify the most common and relevant reasons for connecting a seized device to the Internet during an investigation. Based on these insights, a variety of use cases were selected to reflect both practical needs and technical challenges encountered in forensic work.

Ultimately, the decision was made to include the following scenarios: Communication apps Telegram and WhatsApp, the lesser-known app Session, media access via iCloud, and crypto wallet access through Binance. These cases were chosen to demonstrate a range of typical forensic concerns, including encrypted messaging, lazy loading, cloud synchronization, and potential financial data retrieval.

### Test Environment Setup

This section outlines the shared environment and device configuration used throughout the following use case scenarios. All tests were conducted in a controlled WLAN network configured with a default-deny firewall policy, requiring explicit manual approval of outbound connections on a per-IP basis.

**Network and System Configuration**

* **Firewall:** OPNsense (version Ultimate Unicorn 25.1) configured to block all outbound traffic by default. Custom firewall rules were applied dynamically via API to permit specific connections when necessary.
* **Access Point:** TP-Link EAP610, directly connected to the OPNsense firewall via Ethernet. No other devices or traffic were present on this WLAN, ensuring a clean and controlled test environment.
* **Web Application:** A locally hosted Django Framework provided an interface for interacting with firewall logs, rule management, and IP enrichment results.
* **Rule Handling:** Firewall rules were created, verified, and removed via the OPNsense API. All changes were logged to support rule behaviour analysis.
* **IP Enrichment:** IP metadata (e.g., ISP, organization, and geographic location) was retrieved using ip-api.com.
* **DNS Resolution:** All DNS queries from test devices were redirected to the local Unbound DNS service on the OPNsense firewall, ensuring visibility into domain lookups and allowing DNS-based rule control.

**Test Devices: iOS and Android Configuration**

To gain insight into the behaviour of both major mobile operating systems, iOS and Android, it was decided to conduct use case testing using one device from each platform. Given the potential risk of a remote wipe (whether triggered intentionally or unintentionally), new iPhones and Samsung devices were specifically set up for this purpose. This precaution ensures that critical data is not lost during testing and allows safe exploration of the wipe functionality.

Up to this point, limited testing had been performed on iOS, using the Find My iPhone feature's ring command as a placeholder for the actual erase function. However, it remains unclear whether both commands interact with the system in the same way. This distinction must be thoroughly evaluated before conducting tests on real seized devices. At the time of writing, no comparable tests had yet been conducted on Android. This marks the starting point for structured testing on both platforms. The goal is to perform each defined use case on both devices and document the observed behaviour accordingly. Table 11 lists the specifications of the two test devices, including model, operating system version, and relevant configuration details.

Table 11: Test Devices Overview

|  |  |  |
| --- | --- | --- |
| **Device** | iOS Test Phone | Android Test Phone |
| **Model** | iPhone 14 Pro | Samsung Galaxy S21 Ultra |
| **OS Version** | iOS 18.4 | Android 13, Security Patch Level Nov 2023 |
| **Private Relay** | Deactivated | - |
| **MAC Random.** | Deactivated | Deactivated |
| **Notes** | - | No newer software version available |

### WhatsApp Messenger Access (iOS / Android)

**Objective:**

This test evaluates whether WhatsApp can operate within a network environment configured to explicitly permit only selected IP connections.

Specifically, the test aimed to verify the following objectives:

* Access to the chat overview after launching the app, verifying that core connectivity to the WhatsApp infrastructure is functional.
* Retrieval of uncached chat messages from WhatsApp servers, demonstrating live synchronization beyond local storage.
* Download and display of previously received or sent media that is no longer stored on the device, confirming selective media access.

**Steps:**

* The test device (iOS and Android) was connected to a WLAN with a default-deny firewall policy.
* Launched WhatsApp and observed that no content loaded initially.
* Monitored Blocked view and identified 21 entries for Facebook, Inc.
* Switched to DNS view and allowed all entries matching \*.whatsapp.net.

**Result:**

Once the initial set of connections was permitted, basic chat functionality, such as viewing recent messages, became available. However, to retrieve older, uncached messages and media, additional DNS entries had to be resolved and allowed shortly after the initial launch.

Six domains were required to enable full WhatsApp functionality:

* e13.whatsapp.net
* e2.whatsapp.net
* g-fallback.whatsapp.net
* g.whatsapp.net
* mmg-fallback.whatsapp.net
* mmg.whatsapp.net

The test confirmed that WhatsApp operations could be selectively enabled without permitting unrestricted Internet access.

### Telegram Messenger Access (iOS / Android)

**Objective:**

This test evaluates whether Telegram can operate within a network environment configured to explicitly permit only selected IP connections.

Specifically, the test aimed to verify the following objectives:

* Access to the chat overview after launching the app, verifying that core connectivity to the Telegram infrastructure is functional.
* Retrieval of uncached chat messages from WhatsApp servers, demonstrating live synchronization beyond local storage.
* Download and display of previously received or sent media that is no longer stored on the device, confirming selective media access.

**Steps:**

* The test device (iOS and Android) was connected to a WLAN with a default-deny firewall policy.
* Launched Telegram and observed that no content loaded initially.
* Monitored Blocked view and identified three entries for Telegram Messenger Amsterdam Network and four entries for Telegram Messenger Network
* Approved the two ISPs directly in the Blocked view,

**Result:**

Once the initial set of connections was permitted, basic chat functionality, such as viewing recent messages, became available. However, to retrieve older, uncached messages and media, it was necessary to update the ISP group again, resulting in a total of nine active rules by the end of the session. The final ruleset is shown in Figure 8.

During testing, no DNS requests related to Telegram were observed in the DNS view. A subsequent inspection of the DNSRecord database model confirmed that no Telegram-related entries had been recorded. This indicates that the Telegram application likely uses hardcoded IP addresses and does not rely on DNS resolution at runtime. This reinforces the assumption that they are statically embedded within the application. However, the addresses were successfully enriched using the ip-api.com service, which provided sufficient metadata for classification. Following the initial IP approvals, additional connections to previously unseen IP addresses were observed. Whether these addresses are also embedded within the app or retrieved through alternative resolution methods, such as DNS-over-HTTPS (DoH[[1]](#footnote-1)), was not investigated as part of this test. The test confirmed that Telegram operations could be selectively enabled without permitting unrestricted Internet access.

ISP: Telegram Messenger Amsterdam Network (4)

IP ORG ISP DNS Location

149.154.167.223 N/A Telegram Messenger Amsterdam Network - EC1N London, UK

149.154.167.41 N/A Telegram Messenger Amsterdam Network - EC1N London, UK

149.154.167.92 N/A Telegram Messenger Amsterdam Network - EC1N London, UK

149.154.165.120 N/A Telegram Messenger Amsterdam Network - EC1N London, UK

ISP: Telegram Messenger Network (5)

IP ORG ISP DNS Location

91.108.56.167 N/A Telegram Messenger Network - 1012 Amsterdam, NL

149.154.175.100 N/A Telegram Messenger Network - 1012 Amsterdam, NL

149.154.175.57 N/A Telegram Messenger Network - 1012 Amsterdam, NL

149.154.175.53 N/A Telegram Messenger Network - 1012 Amsterdam, NL

149.154.175.60 N/A Telegram Messenger Network - 1012 Amsterdam, The Netherlands

sdsdsd

Figure 8: Applied Firewall Rules to Restore Telegram Functionality

### Session Messenger Access (iOS / Android)

**Objective:**  
This test evaluates whether Session Messenger can operate within a network environment configured to explicitly permit only selected IP connections.

Specifically, the test aimed to verify the following objectives:

* Access to the chat overview after launching the app, confirming that connectivity to the core Session infrastructure is established.
* Retrieval of uncached chat messages from Session servers, demonstrating live synchronization beyond local storage.
* Download and display of previously received or sent media that is no longer stored on the device, confirming selective media access.

**Steps:**

* The test device (iOS and Android) was connected to a WLAN with a default-deny firewall policy.
* Launched Session and observed that no content loaded initially.
* While monitoring the Blocked view, no direct entries or identifiable patterns related to Session were observed.
* Used the DNS Lookup view with getsession.org which showed seven Cloudflare IP addresses, all within the 104.21.0.0/16 range. These were added to the firewall rules, but the application was still unable to establish a connection. Subsequent observations indicated that Session does not rely solely on centralized infrastructure or conventional DNS resolution.
* Further investigation into the Session architecture revealed its reliance on the Oxen Network, a decentralized, onion-routed system that does not operate through fixed IPs or standard domain mappings. As a result, Numerous connection attempts originated from a diverse set of unknown ISPs, consistent with the behavior of a distributed privacy-focused network.
* Eventually, it was observed that Session attempts TCP connections over ports 22020 to 22023. Adding a general firewall rule to allow outbound traffic over these ports enabled successful communication and restored full application functionality of the app.

Figure 9 shows the firewall Passed view after adding a rule permitting outbound traffic on TCP ports 22020–22023, clearly illustrating that Session Messenger immediately initiates several communications across multiple IP addresses. This confirms that the application depends on dynamic peer-to-peer connections established over these ports.

192.168.5.140:49816 → 162.19.67.191:22021 ✅ OVH SAS

192.168.5.140:49100 → 164.68.98.57:22021 ✅ Contabo GmbH

192.168.5.140:33272 → 195.201.134.115:22021 ✅ Hetzner Online GmbH

192.168.5.140:50328 → 164.68.115.246:22021 ✅ Contabo GmbH

192.168.5.140:40030 → 91.231.182.106:22021 ✅ South Park Networks LLC

192.168.5.140:60704 → 159.65.192.33:22021 ✅ DigitalOcean, LLC

Figure 9: Pass view after adding firewall rule

**Result:**Session Messenger leverages decentralized routing through the Oxen network[34], bypassing traditional centralized infrastructure. This architectural choice renders standard IP-based or DNS-based allowlisting ineffective, as Session does not rely on fixed domains or predictable IP ranges. Instead, it establishes connections across a dynamic mesh of community-run nodes using non-standard ports 22020–22023[35], [36]. These ports are essential for enabling Session’s functionality. Once these specific ports were allowed through the firewall, Session operated normally. All key features including chat messaging, media transmission (e.g., videos), and access to previously exchanged files became functional without the need for any specific domain or IP-based allowances.

This highlights a fundamental difference in network architecture between Session and more traditional messaging platforms (e.g., WhatsApp, Telegram), which rely on centralized servers resolvable via DNS and hosted in static IP ranges. The decentralized design of Session aligns with its privacy-first philosophy but poses unique challenges for firewall-based control and monitoring.

These findings indicate that additional system functionality is required to support Session without relying on conventional firewall rule management. However, it remains technically feasible to enable Session Messenger by permitting the required ports.

### iCloud Photos Access – Media Retrieval (iOS)

**Objective:**

The objective of this test is to evaluate whether iOS Photos / iCloud Media can operate within the described environment, where only explicitly allowed IP connections are permitted.

Specifically, the test aimed to verify the following objectives:

* Whether full-resolution images that are not cached locally on the device can be successfully retrieved from iCloud storage.
* Which IP addresses or ISPs are contacted during the iCloud media download process, to support rule definition and monitoring.
* Whether iCloud connectivity can be selectively permitted to allow photo access while still blocking critical infrastructure that may enable a remote wipe command from Apple’s servers

**Steps:**

* The iOS test device was connected to a WLAN with a default-deny firewall policy.
* Launched the Photos app, no non-local content was displayed.
* DNS blocks in place for Find My Device (see Section 4.1.1):

p42-fmipmobile.icloud.com, p42-fmf.icloud.com, fmf.fe2.apple-dns.net, fmfmobile.fe2.apple-dns.net, fmip.fe2.apple-dns.net, p124-fmip.icloud.com.

* Observed 78 ISP entries from Apple Inc. in the firewall’s Blocked view.
* Approved the relevant ISP directly from the Blocked view.
* The Photos app was checked again, but non-local videos were still not loading.
* A second round of ISP rule approval was performed, resulting in a total of 100 active rules

**Result:**

After the second round of ISP rule approvals, the iOS device successfully retrieved photos and videos that were not cached locally. This indicated that the underlying iCloud media services had become fully operational. In addition to media access, the device was able to load iCloud account settings, including options to initiate a full device backup, access storage information, and view account details. This level of functionality was unexpected, given the assumption that stricter filtering, especially through targeted DNS blocks of known Find My infrastructure, would also disrupt broader iCloud services. However, the findings indicate that core iCloud functionalities, including photo and video retrieval, account access, and backup management, can operate independently of services related to device tracking or remote wipe. The test confirmed that iCloud operations can be selectively enabled without granting unrestricted Internet access, even when access to the broader 17.248.209.0/24 subnet is permitted.

### Binance - Account and Transactions Retrieval (iOS / Android)

**Objective:**

The objective of this test is to evaluate whether Binance can operate within the described environment, where only explicitly allowed IP connections are permitted.

Specifically, the test aimed to verify the following objectives:

* Retrieval and display of the user's account balance
* Retrieval and display of recent transaction history

**Steps:**

* The test device (iOS and Android) was connected to a WLAN with a default-deny firewall policy.
* Launched the Binance app, no content was initially displayed.
* Monitored the DNS view and identified two key domains: \*.binance.com. and \*.bnbstatic.com.
* Approved these entries, which resulted in the creation of 75 firewall rules. It took three iterations of rule application to fully restore Binance app functionality. In other test runs, up to six iterations and over 170 rules were required.

**Result:**

After applying the necessary rules, the Binance app became fully functional. This included access to real-time balance, cryptocurrency prices, and transaction history.

The test demonstrated that Binance functionality can be selectively enabled using a controlled whitelist approach, without granting general Internet access. However, the timing of rule application proved to be a critical factor. If rule updates were delayed, the app continued to generate additional DNS lookups and IP requests, thereby unnecessarily expanding the required rule set. This behaviour highlights the importance of timely rule propagation during firewall-restricted testing.

## Firewall Rule Propagation Delay

When applying a large number of rules via the OPNsense API, such as toggling an ISP alias group containing hundreds of destination IP addresses, the system exhibited significant latency. During this period, the web interface appeared unresponsive, and the new rules were not yet enforced by the firewall.

Empirical observations revealed significant performance degradation in OPNsense when applying firewall rules via its API. Under minimal load, initial rule additions, when few or no active rules exist, took approximately 0.2 seconds per rule. As the number of active rules increased, latency also increased significantly. With approximately 300–500 existing rules, latency increased to around 1.5 seconds per additional rule. In large rule sets exceeding 700 entries, the average delay reached more than 2 seconds per rule. A stress test involving the creation and commit of 600 rules resulted in a total execution time of approximately 15 minutes. While not reflective of typical usage, this demonstrates the scalability limitations in scenarios involving bulk policy changes or multi-device operations. Rule removal proved faster, ranging from 0.2 to 1.2 seconds per entry, but still scaled with the total number of active rules. See Figure 10 for illustration.

Figure 10: Average Time per Firewall Rule Operation in OPNsense

Performance was improved slightly by reusing persistent API sessions and minimising redundant calls. However, the OPNsense API processes rule changes sequentially and lacks native support for batch operations, resulting in significant delays when handling large rule sets. An alternative approach, discussed in the OPNsense community, involves generating a complete firewall configuration file and re-uploading it in one step. However, this approach is likely to fail if the syntax is not strictly followed, as the configuration file includes all system settings. Any adjustment must be precise and well-formed. This method was not evaluated within the scope of this project. A major improvement was discovered through the use of OPNsense Aliases. An Alias allows grouping many IPs under a single identifier referenced by one rule. In testing, adding an alias with 1,000 IPs took only ~4 seconds, compared to over 20 minutes when adding individual rules. Aliases are always rebuilt as a whole, so they are less efficient for one-off changes. However, for typical bulk operations, in scenarios where five or more entries are added simultaneously, they offer a clear performance advantage. With the Django database as the single source of truth, alias regeneration offers a rapid and consistent update mechanism.

# Discussion and Conclusion

This section reflects on the key findings of the conducted test series, outlines the limitations of the current approach, and discusses the practical implications for forensic operations. It also provides targeted recommendations and identifies areas for future research. The results demonstrate that network-based control of seized mobile devices is both feasible and valuable, allowing forensic examiners to safely access cloud data while mitigating risks such as remote wipe or command execution. The following sections summarise these insights

## Summary of Key Findings

This section summarises the main outcomes of the experimental evaluation, focusing on how DNS and firewall-based controls can be used to selectively allow or block network communication on mobile devices. The findings cover the effectiveness of wipe risk mitigation strategies, differences in platform behavior between iOS and Android, challenges posed by decentralized applications like Session, and the performance characteristics of different firewall rule handling strategies.

### Wipe Risk Mitigation

The tests demonstrated that it is technically feasible to enable selective access to services such as iCloud, WhatsApp, and Binance, while simultaneously blocking device wipe and ring commands through a combination of DNS filtering and targeted IP rule enforcement. This approach allows user activity, such as viewing iCloud photos or accessing app data, while mitigating the risk of remote actions that could result in data loss or undermine evidential integrity.

### iOS vs. Android Differences

iOS demonstrates consistent DNS and IP behaviour, though the specific roles of individual services remain partially unclear. Multiple DNS queries, such as those containing fmipmobile, fmf, fmfmobile, and fmip, all relate to Apple’s Find My network, but it is not yet clear which query corresponds to which function (e.g., wipe, ring, location). However, these services consistently resolve to IP addresses within the 17.248.209.0/24 subnet, which also hosts iCloud services. This overlap is significant, as it allows selective blocking of control commands (like wipe or ring) while maintaining access to cloud content, which may be essential in forensic investigations.

In contrast, Android’s behaviour is more generalized. Wipe commands and related control functions are routed through services under the \*talk.google.com domains. While it is not obvious at first glance, all relevant DNS queries resolved to IPs x.x.x.188, and no other observed Google services used IPs with this ending. This provides a reliable indicator for identifying Android’s remote command infrastructure. However, unlike iOS, the Android wipe process occasionally exhibited significant delays.

One challenge in DNS-based control arises from privacy features such as Apple Private Relay and iCloud+, which can obscure DNS visibility by encrypting queries or routing them through Apple-controlled endpoints. These services effectively bypass local DNS filtering, making it difficult to monitor or block domain-based activity. Although these features were disabled during testing, future systems must account for their potential impact. Therefore, it is essential that such privacy features be explicitly disabled.

### Session Messenger Insight

Session’s decentralized architecture, built on the Oxen network and reminiscent of Tor in its routing logic, is fundamentally incompatible with traditional IP- or DNS-based firewall whitelisting. Instead, it requires port-level filtering, specifically allowing outbound TCP connections on ports 22020–22023, to function properly. In this case, the required port range does not appear to be widely used by other common services, reducing the risk of unintended access when these ports are opened across the network. However, this approach departs from typical DNS-based filtering strategies. Unlike DNS-based filtering, where rules can be tightly scoped to identifiable services, port-level exceptions introduce uncertainty, as it is unclear what other traffic may pass through those same ports. This makes it riskier or poses a greater risk, as it increases the risk of unwanted changes on the mobile device.

For applications like Session, the forensic examiner should have the ability to define per-device port-based exceptions or build in support for specific use cases to directly apply the necessary ruleset.

Importantly, different use cases require different approaches. It is essential to thoroughly plan and analyse the application’s network behaviour before applying any firewall rules. In cases involving non-standard or decentralized services, providing dedicated functionality in the frontend to guide rule creation, based on previously observed connection patterns, would significantly improve usability and security.

### Firewall Rule Handling

Efficient rule management is essential in forensic contexts requiring dynamic, case-specific network control. As shown in Section 4.3, sequential rule creation via the OPNsense API introduces significant delays when managing large rule sets. As an illustration, applying 600 individual rules took approximately 15 minutes, making this approach impractical for real-time operations.

A key improvement would be the use of OPNsense Aliases, which group many IPs under a single rule. This was tested but not implemented yet. Applying an alias with 1,000 IPs took only about 4 seconds, compared to a time well over 20 minutes with individual rules. This performance gain is due to the fact that alias updates do not require reapplying or rebuilding the full rule table; the alias content is simply reloaded in the background, without altering the rule structure. While aliases are less suitable for frequent, single-entry updates, they are well-suited for bulk operations, especially when paired with a structured backend like Django. Overall, alias-based rule handling is recommended for real-world scalability, offering both speed and control in forensic environments.

### Device Scope

The tests in this thesis were conducted on a single iOS device and a single Android device, each intended as representative examples of their respective operating systems. While the results provide valuable insight into network behaviour and control mechanisms, it is important to acknowledge that device behaviour may vary significantly based on several factors. Android’s open ecosystem allows manufacturers to implement custom firmware and networking stacks. As a result, behaviour may differ across vendors, such as Xiaomi, Huawei, Samsung, or Google Pixel, especially in how they handle push notifications, background services, or system-level wipe commands. Furthermore, new software releases, on both platforms, can lead to changes highlighting the need for continuous testing to ensure continued effectiveness of the firewall configurations, such as DNS blocklists. Therefore, the findings presented should be interpreted as baseline observations, with the understanding that real-world deployments may require device-specific validation to accommodate vendor, configuration, or OS-level differences.

### Third-Party MDM Services

This thesis focuses primarily on native operating system behaviour, including built-in mechanisms such as remote wipe, device tracking, and cloud access. However, third-party Mobile Device Management (MDM) platforms introduce additional layers of control and communication mechanisms which are beyond the scope of this analysis. These platforms may use their own server infrastructure, APIs, and communication protocols, which can differ significantly from those used by the native OS. For example, MDM-triggered wipe commands might be delivered through proprietary push channels or cloud services that operate independently of those used by Apple or Google.

## Practical Implications for Forensics

One of the fundamental principles of digital forensics is to preserve the original state of a device from the moment of seizure. Traditionally, this has meant avoiding any alteration of data, often enforced through the use of hardware write blockers, which prevent modifications to storage media during analysis. For many years, this was regarded as a strict and non-negotiable standard in forensic practice. However, the landscape has significantly evolved. Modern devices increasingly rely on onboard storage and secure elements, such as TPM (Trusted Platform Module) chips, which render traditional acquisition methods, like direct disk imaging, impractical or even impossible. Consequently, booting a device to access its data, once considered a high-risk compromise of evidence integrity, has become a necessary and accepted step in many forensic workflows. In some cases, the question is no longer how to avoid altering the device, but rather whether any data can be acquired at all.

In mobile forensics, this challenge has long been recognised. Mobile devices typically use non-removable, soldered memory, requiring them to be booted for data acquisition. This practice is widely accepted due to the physical design limitations of smartphones. Moreover, in many cases, controlled modifications, such as bypassing security features or enabling low-level access, are necessary to extract any data at all. When using advanced acquisition tools like Cellebrite Premium or Axiom GrayKey, a complete forensic image of the device is generated during this initial phase. This image serves as the preserved, verifiable reference point. Any subsequent interaction with the live device, whether for cloud access, app reauthentication, or data inspection, can always be compared against this baseline to detect modifications, deletions, or timestamp changes.

Despite the need to adapt to technical realities, the core forensic principle of minimizing alterations remains vital. This thesis demonstrates that, using the developed frontend, it is possible to safely bring a device online in a tightly filtered and controlled network environment. This enables access to additional cloud-based without exposing the device to unrestricted Internet access or triggering remote wipe or alert commands. The key outcome is that targeted online acquisition can be performed quickly, safely, and reproducibly, without extensive manual configuration or compromising forensic soundness. This approach balances evidence preservation with the practical requirements of modern digital forensics.

## Practical Recommendations

Based on the findings of this thesis, the following recommendations are proposed for implementing safe and effective network control in forensic environments:

**Prioritize DNS-Based Filtering Before ISP Allowlisting**

DNS-based filtering offers more granular control and clearer visibility during the initial stages of analysis. Unlike broad ISP-level allowlisting, DNS rules enable more precise targeting of required services. Domain names provide clearer insight into application behaviour, and DNS logs help identify which services the device is attempting to reach. In many cases, it is unnecessary to permit access to the entire ISP infrastructure. For example, if only WhatsApp functionality is needed, allowing all traffic to the Facebook Inc. infrastructure may expose the device to other services like Facebook or Instagram. This violates the forensic principle of minimizing exposure and avoiding unnecessary changes. DNS-based filtering enables service-specific access, reducing the risk of triggering unintended app behaviour and maintaining stricter evidence preservation.

**Block Wipe-Related DNS Entries by Default**

Certain domains, such as fmipmobile.icloud.com for iOS and talk.google.com for Android, are closely associated with remote wipe, ring, or device tracking functionalities. To prevent unintended activation of these commands, such domains should be proactively blocked in the network. This measure helps ensure that seized devices remain unaltered by remote commands, preserving the integrity of potential evidence during initial examination and acquisition.

However, it is critical to recognize the limitations of relying solely on static DNS-based blocking. These domain names can change over time, become deprecated, or become obscured by load balancers or Content Delivery Networks (CDNs). Furthermore, the increasing adoption of encrypted DNS protocols like DNS-over-HTTPS (DoH) or DNS-over-TLS (DoT) can make traditional DNS monitoring and blocking ineffective. In addition, allowing other Apple or Google services to function may inadvertently reenable communication pathways to critical control domains.

**Validate on Test Devices Before Applying to Evidence**

It is strongly recommended that all rulesets be validated on test devices prior to deployment on evidentiary devices. Network behaviour can vary significantly between device models, OS versions, and configurations (e.g., MDM-enrolled devices). Controlled validation helps prevent data loss. Furthermore, controlled testing supports the goal of minimizing alterations to the target device. When an effective and minimal ruleset is identified, one that allows the required application or function to operate while blocking unnecessary or dangerous network activity, it reduces the risk of affecting unrelated services. This process of trial-and-error on non-evidentiary or identically configured test devices is crucial to determining the most precise and least invasive approach for the actual target device.

## Future Work

This project has demonstrated a viable and secure approach to network-based evidence acquisition and control. However, several areas remain for further development, optimisation, and broader evaluation:

**Enhanced Alias Rule Handling:** Further development should focus on improving the Alias-based firewall rule handling by implementing batch processing for larger rule sets. This would significantly reduce the total number of active firewall rules and enhance responsiveness of the API. A proposed enhancement includes a hybrid method: allow temporary application of individual rules for immediate effect, which are later consolidated into the corresponding alias. This approach combines the speed of single-rule handling with the long-term efficiency of alias grouping.

**Frontend Logic for DNS Whitelisting:** Adding functionality in the frontend to manually approve or deny unknown DNS queries, particularly those associated with high-risk domains such as Apple’s, would increase operator control. For example, unknown Apple-related DNS names could be blocked by default and only resolved or allowed after manual review and approval. This provides a safeguard against triggering unintended behaviours, including location tracking or remote wiping.

**Dynamic Rule Expansion Based on Session Context:** Implement the option for automatically updating rulesets. If a specific ISP or DNS entry is approved, the system could automatically extend access to related entries for the same device. For instance, if Google LLC is allowed during a session, newly identified IP addresses associated with Google for that device could be dynamically included. Similarly, if fstream.binance.com is allowed, the system could permit all new resolved IP addresses to this domain for that device.

**MDM Behaviour Under Network Restrictions:** Further investigation is needed into the behaviour of third-party Mobile Device Management (MDM) systems under restricted network conditions. Platforms such as Microsoft Intune may deliver wipe or control commands via distinct mechanisms. Understanding how these services react in isolated environments is essential for safe handling of MDM-enrolled devices.

**Broader Device and OS Testing:** The current tests were limited to a single iOS and Android device. Future work should include a broader test set, covering various vendors and OS versions, such as:

* Google Pixel with Android 14+
* Google Pixel running Graphene
* Xiaomi and Huawei devices running manufacturer or custom ROMs
* iPadOS

This would ensure broader applicability of findings and help identify platform-specific behaviours that could impact rule design or wipe risk mitigation.

In conclusion, this thesis has demonstrated that rule-based network control offers a practical and technically sound method for enabling secure online access to seized mobile devices in forensic contexts. By leveraging granular DNS filtering, IP allowlisting, and dynamic rule handling, the system provides investigators with the ability to selectively permit essential application functionality while mitigating risks such as remote wipe or unauthorised communication. The approach balances the need for evidentiary preservation with the operational demands of modern forensic workflows and adapts to the technical realities of increasingly secure and cloud-reliant mobile platforms. While further refinement and broader validation are necessary, the prototype developed and tested establishes a strong foundation for future tools designed to manage mobile device exposure in a forensically sound manner.

# References

[1] K. Ingham and S. Forrest, ‘Network Firewalls’, University of New Mexico, 2002. [Online]. Available: http://iar.cs.unm.edu/~forrest/publications/firewalls-05.pdf

[2] Palo Alto Networks, ‘The History of Firewalls | Who Invented the Firewall?’ Accessed: Aug. 05, 2025. [Online]. Available: https://www.paloaltonetworks.com/cyberpedia/history-of-firewalls

[3] C. Dilmegani, ‘Top 7+ Open Source Firewall Options in 2025: Features & Types’, AIMultiple Research. Accessed: Jun. 02, 2025. [Online]. Available: https://research.aimultiple.com/open-source-firewall/

[4] ‘pfSense® - World’s Most Trusted Open Source Firewall’. Accessed: May 14, 2025. [Online]. Available: https://www.pfsense.org/

[5] OPNSense, ‘OPNsense Documentation’, OPNsense. Accessed: Feb. 12, 2025. [Online]. Available: https://docs.opnsense.org

[6] ‘VyOS – Open source router and firewall platform’, VyOS. Accessed: May 14, 2025. [Online]. Available: https://vyos.io/

[7] ‘ClearOS’. Accessed: May 14, 2025. [Online]. Available: https://clearos.com/

[8] A. Dubey, ‘Comprehensive Guide to Linux Firewalls: iptables, nftables, ufw, and firewalld’, Medium. Accessed: Aug. 05, 2025. [Online]. Available: https://medium.com/@amandubey\_6607/comprehensive-guide-to-linux-firewalls-iptables-nftables-ufw-and-firewalld-9e86e0a49979

[9] ‘iptables(8) - Linux manual page’. Accessed: May 14, 2025. [Online]. Available: https://man7.org/linux/man-pages/man8/iptables.8.html

[10] ‘nftables wiki’. Accessed: May 14, 2025. [Online]. Available: https://wiki.nftables.org/wiki-nftables/index.php/Main\_Page

[11] ‘UncomplicatedFirewall - Ubuntu Wiki’. Accessed: May 30, 2025. [Online]. Available: https://wiki.ubuntu.com/UncomplicatedFirewall

[12] ‘iptables(8) - Linux manual page’. Accessed: May 14, 2025. [Online]. Available: https://man7.org/linux/man-pages/man8/iptables.8.html

[13] ‘Hardware – OPNsense® Shop’. Accessed: May 14, 2025. [Online]. Available: https://shop.opnsense.com/product-categorie/hardware-appliances/

[14] ‘Hardware sizing & setup — OPNsense documentation’. Accessed: May 14, 2025. [Online]. Available: https://docs.opnsense.org/manual/hardware.html

[15] ‘Hardware and Performance’, OPNsense Forum. Accessed: May 14, 2025. [Online]. Available: https://forum.opnsense.org/index.php?board=21.0

[16] ‘What is a Remote Wipe? | Definition from TechTarget’, Search Mobile Computing. Accessed: May 14, 2025. [Online]. Available: https://www.techtarget.com/searchmobilecomputing/definition/remote-wipe

[17] Apple Inc, ‘Apple Platform Security’, p. 302, Dec. 2024.

[18] ‘List of mobile device management software’, *Wikipedia*. May 10, 2025. Accessed: May 14, 2025. [Online]. Available: https://en.wikipedia.org/w/index.php?title=List\_of\_mobile\_device\_management\_software&oldid=1289694990

[19] ‘File-based encryption’, Android Open Source Project. Accessed: May 15, 2025. [Online]. Available: https://source.android.com/docs/security/features/encryption/file-based

[20] ‘Find, secure or erase a lost Android device - Android Help’. Accessed: May 15, 2025. [Online]. Available: https://support.google.com/android/answer/6160491?hl=en-GB

[21] ‘Wipe corporate data from a device - Google Workspace Admin Help’. Accessed: May 15, 2025. [Online]. Available: https://support.google.com/a/answer/173390?hl=en

[22] ‘Android Business Device Solutions Directory - Android Enterprise - EMMs’, Android Enterprise. Accessed: May 15, 2025. [Online]. Available: https://androidenterprisepartners.withgoogle.com/emm/

[23] ‘Privacy features when connecting to wireless networks’, Apple Support. Accessed: May 15, 2025. [Online]. Available: https://support.apple.com/en-gb/guide/security/secb9cb3140c/web

[24] Apple Inc, ‘iCloud Private Relay Overview’, p. 11, Dec. 2021.

[25] Elaunira SARL, ‘Affordable IP Geolocation and Threat Intelligence Pricing - Ipregistry’. Accessed: May 15, 2025. [Online]. Available: https://ipregistry.co

[26] Kloudend, Inc., ‘ipapi - IP Address Lookup and Geolocation API | No SignUp’. Accessed: May 15, 2025. [Online]. Available: https://ipapi.co

[27] ipwhois.io, ‘IP Geolocation API - Pricing’. Accessed: May 15, 2025. [Online]. Available: https://ipwhois.io

[28] ipify.org, ‘IP Geolocation API - Try Our IP Location API Free Of Charge’. Accessed: May 15, 2025. [Online]. Available: https://geo.ipify.org

[29] JFreaks Software Solutions, ‘IP Geolocation API Pricing’, IP Geolocation API Pricing. Accessed: May 15, 2025. [Online]. Available: https://ipgeolocation.io

[30] ipdata.co, ‘IP Geolocation API with Threat Intelligence’. Accessed: May 15, 2025. [Online]. Available: https://ipdata.co

[31] ip-api.com, ‘IP-API.com - Geolocation API’. Accessed: May 15, 2025. [Online]. Available: https://ip-api.com

[32] IPinfo, ‘Trusted IP Data Provider, from IPv6 to IPv4’. Accessed: May 15, 2025. [Online]. Available: https://ipinfo.io

[33] J. Hickman, ‘iOS 15 Powered-Off Tracking & Remote Bombs’, The Binary Hick. Accessed: May 14, 2025. [Online]. Available: https://thebinaryhick.blog/2021/10/27/ios-15-powered-off-tracking-remote-bombs/

[34] ‘Introduction to Oxen | Oxen Docs’. Accessed: Jun. 12, 2025. [Online]. Available: https://docs.oxen.io/oxen-docs

[35] ‘Express service node setup guide | Oxen Docs’. Accessed: Jun. 12, 2025. [Online]. Available: https://docs.oxen.io/oxen-docs/using-the-oxen-blockchain/oxen-service-node-guides/setting-up-an-oxen-service-node

[36] iana.org, ‘Service Name and Transport Protocol Port Number Registry’, Service Name and Transport Protocol Port Number Registry. Accessed: Jun. 12, 2025. [Online]. Available: https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml?search=22022

# Appendix

Hier sind die in der Arbeit referenzierten Anhänge aufzuführen.

# Declaration of Originality

Bitte Wortlaut aus «Merkblatt Erstellung Abschlussarbeit in CAS, DAS und MAS» übernehmen.

1. DNS-over-HTTPS (DoH) encrypts DNS queries over HTTPS (port 443), unlike traditional DNS (port 53), which transmits queries in plaintext and can be intercepted or modified. [↑](#footnote-ref-1)