Zürcher Hochschule
für Angewandte Wissenschaften

School of Engineering

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 1. June 2025
Betreuer:in Peter Heinrich

Abstract

Dies ist ein Platzhalter-Text. Dies ist ein Platzhalter-Text. Dies ist ein Platzhalter-Text. Dies ist ein Platzhalter-Text. Dies ist ein Platzhalter-Text. Dies ist ein Platzhalter-Text. Dies ist ein Platzhalter-Text. Dies ist ein Platzhalter-Text.

Table of Contents

1. Introduction

1.1. Preface

Electronic evidence plays a central role in today's criminal prosecution. Almost no criminal proceedings can do without the analysis of digital data - be it data from mobile phones, computers, IoT devices, cloud data and many other data sources. 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 does 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[1]. 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 from external. As soon as a mobile phone is reconnected to the internet it will updating apps and synchronising data, such as cloud services, messages and other application data and this of course will trigger write processes on the data storage. Also, there is the risk of a 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 would be clear, establishing a network connection or even simply powering on a device seems not to be 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 [2]correctly - In fact there are often problems during parsing in practice. Especially apps just known in Switzerland like “Twint” are not parsed automatically.
Without powering on the device, it is not possible to detect parsing errors. For example, when no WhatsApp messages are shown at all or to identify if any 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[3] 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 also causes write operations, but it is considered as 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 more and more 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 manual reviewing chat conversations trough a police investigator, it can happen that the content of chats clearly goes in one direction, but the images and videos taken are missing, this because they are no longer or have never been stored locally on the device. For example, an image that could be identified as an offence

[1] Faraday room or Faraday cage, which prevents communication from or into the room. A great and quite simple setup was made by the University of Gottingen [1].
[2] Parsing refers to the process of analysing structured data like file systems, databases, or app data, to extract data from the seized device.
[3] Brute force refers to methods that systematically attempt all possible passcode combinations.

or a prohibited media file containing violence or even child pornography may be missing from the device and because that it will not allow any 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. But how to gain this extra information without getting 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 it will normally give a consultation between the public prosecutor, the police officer who is in charge for 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 acquiring trough forensic hard- and software took already 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 the remote deletion of data and at the same time only allowing the necessary connections to the Internet. This MAS thesis aims to close this gap.

1.2. 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, 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 - 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).
Reference to use cases ???…

1.3. 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 work remains on the technical implementation of a device-aware firewall management system for controlled network access and forensic transparency.

2. Background

2.1. 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 the architecture and history: Walls that protected cities like the Great Wall of China or structural firewalls that prevented a fire in the kitchen to spread out on other parts of the building[2]. Similar, in networking, a firewall serves as a barrier to keep not wanted traffic outside and to allow other traffic to go through. Even if the "external side" is burning the firewall is intended to keep the internal network safe and unaffected.

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 or also called Next-Generation Firewalls (NGFW) integrated traditional firewall functions with deep packet inspection (DPI), intrusion prevention systems (IPS), and application 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. It is continuously learning from the network traffic. It aims to reduce the manual intervention [2], [3].

2.2. Overview of Current Firewall Solutions

This section provides an overview of actual available firewall solutions focused on Open Source, Enterprise and system integrated solutions.

2.2.1. 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 of a big advantage is also the opportunity to customize the product. Therefore, many projects offer commercial services like professional support and additional enterprise features. This helps the organisation behind to maintain the software and to fund the needed infrastructure. [8]

pfSense - Is 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 - Is 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 - Is 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 - Is a simple, stateful firewall solution aimed at users with basic network protection needs. It is easy to manage and configure, though it lacks advanced features like NAT and packet filtering.

2.2.2. Enterprise Firewalls

Enterprise firewalls are typically closed-source and unlike open-source alternatives, not freely accessible or modifiable. These solutions often come with a wide range of additional services, including professional support, subscription-based threat intelligence, and service-level agreements (SLAs), making them particularly attractive for organizations that require guaranteed uptime, vendor accountability, and integrated security management. Well-known brands are Palo Alto, Fortinet, Cisco, Check Point, Sophos.

2.2.3. 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.
iptables - Is a user-space utility program that allows administrators to configure the Linux kernel's Netfilter framework for packet filtering and NAT. It has been the traditional 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.[4], [5]
nftables - Is 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 easier-to-read syntax, as well as better performance and integration. Nftables is designed to minimize redundancy and support more maintainable and scalable rule sets.[6]
UFW (Uncomplicated Firewall) - 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.[7]

2.2.4. Requirements for the Firewall Solution

Early in the project phase, it was essential to determine the firewall platform for the project. The Primary goal was to build an extended interface for our specific need rather than developing a complete firewall solution from scratch. The selected system needed to provide a robust and flexible base. For evaluation purpose a minimal set of functional and technical requirements were defined:

Open-Source Licensing 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 in advanced stages of the project.
Self-Hosting on Local Infrastructure The solution must be fully deployable on the organization's own infrastructure without relying on any external cloud components or third-party services. This is essential for maintaining full control over the processed data, meeting forensic standards, and following with the information security and data protection policies (ISDS) of the City of Zurich.
API Support A modern well documented API. The project aims to integrate firewall control into a custom web interface, without the need to interact directly with the firewall's native user interface. API support is also essential for any automation workflows.
Device-Aware Rule Management 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, permitting one device to access a specific IP while restricting another.
Active Development The firewall must have an ongoing development with regular updates and security patches.
Feature-Rich Environment The solution should include a wide range of network security features out of the box. Minimize the need for third-party tools and to have the capability for later processing steps.

2.2.5. Compare Firewall Solutions Based on Requirements

As the 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 is focused solely on open-source, self-hosted solutions. Table 1 presents a comparison between the system-level and the open-source firewall platforms.

Table 1: Comparison of different firewall solutions

Firewall	Open Source	Self-Hosting	API Support	Device-Aware Rules	Active Development	Feature-Rich
iptables[9]	Yes	Yes	No	Limited	Yes	Moderate
nftables[6]	Yes	Yes	No	Limited	Yes	Yes
pfSense[10]	Yes	Yes	Limited	Yes	Yes	Yes
OPNSense[11]	Yes	Yes	Yes	Yes	Yes	Yes
VyOS[12]	Yes	Yes	Limited	Limited	Yes	Yes
ClearOS[13]	Yes	Yes	Limited	No	No	Decreasing with time as no active development is done

After evaluating the available open-source firewall projects and consulting their official documentation and community resources, **OPNsense** was selected as the source firewall for this project. OPNsense has an actively maintained platform with a huge user and developer community.
A great advantage of OPNsense is the officially supported and documented API, which already covers most when not all needed functions for the later project. The project offers regular updates, an open roadmap and detailed release notes.

Although the project would be possible on a base like nftables which is integrated directly into the Linux kernel and with the needed implementation time it would be possible as well. However, the expected additional development time was looked at as too time extensive, this is why a ready to use solution was selected.

2.3. Hardware Evaluation

The Hardware selection was not a primary focus at the outset of the project. After the decision was made to use OPNsense as the firewall platform, further research was made using online sources, including general searches (e.g., Google, ChatGPT) to identify suitable hardware configurations for a cost-effective setup.
The official OPNsense hardware appliance offerings [14] were reviewed but ultimately looked as out of scope due to their cost. These systems are primarily targeted at enterprise customers, with prices often exceeding the budget. The project aimed to stay within a budget of just a few hundred Swiss francs, ideally utilizing small-form-factor, low-cost hardware such as a Raspberry Pi 5.
While the Raspberry Pi 5 initially appeared to be a promising candidate due to its low costs, it was ultimately excluded. Although it meets the official recommended hardware requirements for OPNsense [15], including:
Processor: Minimum 1.5 GHz multi-core CPU
RAM: 8 GB
It failed to meet the additional project-specific requirements, which were essential for practical deployment in a forensic lab environment:
Industrial enclosure: The device must be housed in a durable case suitable for continuous operation in a professional setting.
Networking capabilities: The device must 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).
However rugged enclosures for Raspberry Pi exist, they are still part of a DIY solution and often require additional adapters or USB-to-Ethernet dongles, which compromise reliability and performance. When evaluating networking options specifically, it became clear that no off-the-shelf Raspberry Pi configuration could fulfil the requirements.
In the OPNsense forum [16] many community members shared links to low-cost firewall hardware available on platforms such as AliExpress. However, this option was not considered further due to concerns regarding warranty coverage, lack of technical support, and the potential for long delivery times - often several weeks. These factors would have introduced unnecessary delays and risk to the project timeline.

2.3.1. Final Decision for Protectli V1410

Finally, the decision was made to purchase hardware from Protectli, a US 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.
Selected 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, attention shifts from the software and hardware infrastructure to background information on the challenges of remote wipe in the context of seized devices, with a focus on both iOS and Android operating systems.

2.4. 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.» [17]

2.4.1. Differences Between Wiping and Deletion

Wiping means the process which ensures that the data stored on the device is irreversibly destroyed and to make the recovery impossible. This is typically achieved by overwriting the specific storage area or the entire storage with random data. On the other hand, a simple file deletion involves the system removing only the references to the data, while the actual data remains intact on the storage device. This data can often be recovered and made visible again using specialized tools and software, provided that the storage has not been overwritten due to the need for disk space.

2.4.2. Remote Wipe - Apple iOS

To understand how the remote wipe of an Apple iOS device works it is important to understand how the apple system works. When an iOS device first set up or after each factory reset it generates a random volume key, “media key”. The key is stored locally on the device in a secure location called “Secure enclave”.
All data on the device including app data, system files and user data will be encrypted using the media key. Without the media key it is not possible to access the stored data on the device. If now a remote wipe command is triggered by the Mobile Device Management (MDM) or directly through the iCloud account. The device will as soon the command is received respond with an Acknowledgment (only MDM), and it will instantly perform the wipe command. [18]
It is important to know does with the MDM capability the wipe command can not only come from Apple himself it also can come from various companies if the device is a managed device. Just to list some of the companies: Jamf, Microsoft, MobileIron.[19]
The remote wipe command is designed to remove the media key only. The data himself is not wiped as this would be a time intensive process. However, by removing the media key, the stored data on the file system is not accessible anymore, as the decryption key is missing. This prevents access to the data even for the owner himself as there is no known possibility to decrypt the data anymore.

2.4.3. Apple iOS - Google Android

The Android operating system works slightly different than the iOS operating system. With Android 10 the system was no longer using full-disk encryption (FDE) the system changed to file-based encryption (FBE). After the first set up or after a factory reset, the system creates a set of encryption keys. The Credential Encrypted Key (CE) and the Device Encrypted Key (DE) which are stored in a secure hardware storage like Trusted Execution Environment (TEE) or Secure Element. The Android ecosystem uses a layered key system, where all data on the device, including user data, app data and most system files are encrypted with its own unique key. These unique keys using the CE or DE key for encryption. The CE key is only available after the user has authenticated. Without the CE key it is not possible to decrypt or access the stored data on the device.[20]

The remote wipe command can be triggered with Google's Find My Device Network or also like Apple over an MDM system. When the device receives this command, it may respond with an acknowledgment if done via MDM, after it will immediately initiate the remote wipe process. This process is primarily designed to destroy the CE and DE keys, the actual data is not overwritten. Without the encryption keys, the file data becomes inaccessible and unrecoverable[21], [22].

The wipe command can be issued not only by Google, but also by various enterprise providers if the device is enrolled in their MDM system. These include providers like Microsoft, VMware Workspace ONE, IBM MaaS360, MobileIron, Samsung Knox Manage and many others.[23]

3. Implementation

This chapter describes the implementation of the system developed in the context of this thesis. The goal was to create a practical solution that allows centralized control. 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, the integration with the OPNsense API, the logging and storage of all relevant data in a database, and the deployment of the system on suitable hardware.

3.1. 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:

Firewall: The open-source firewall OPNsense (OS 25.1 – Ultimate Unicorn) acts 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. All interactions with the firewall are done via its officially supported API, which enables dynamic rule management and log queries. Installed on a Protectly V1410.

WLAN Access Point: A dedicated wireless access point in this case a "TP-Link Omada EAP610GP-DESKTOP WLAN Access Point 1201" provides connectivity for mobile devices.

Web Interface / Web Framework: The user interface is implemented using the Django web framework (5.1.6). It allows forensic examiner to manage devices, view logs and manage firewall rules. The interface is quite minimalistic, focused on the core use cases.

Backend Logic: The core application logic is written Python (3.13.1) and organized into multiple files within the Django framework. It processes user input from the web interface, manage 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 choosen 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. Figure 1 illustrates how the individual components interact with each other.

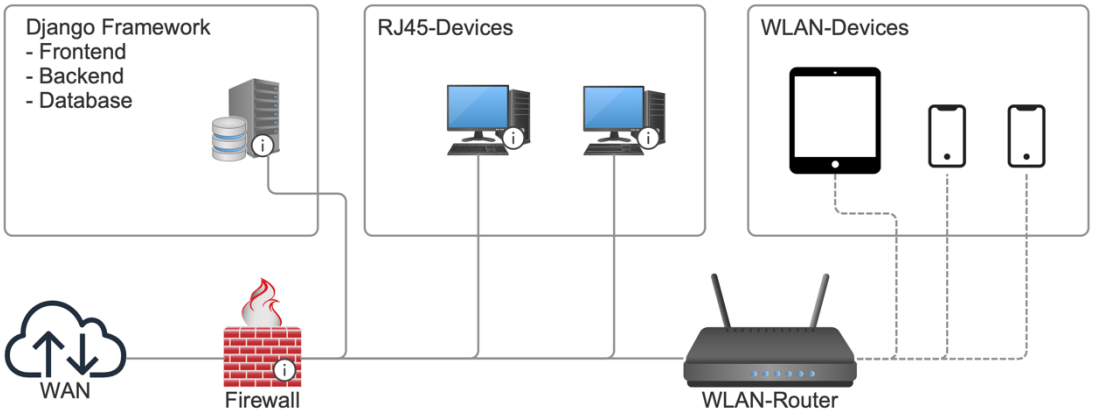

Figure 1: Overview about the system setup

3.2. Prototyping and Preliminary Testing

Before beginning the implementation of the full system, several exploratory prototyping steps were undertaken using simple Python console scripts. As the project domain was entirely new to the author, this phase was important for identifying potential challenges related to device behaviour, network resolution, and firewall integration.

One of the key uncertainties at the beginning was how connected devices would resolve hostnames - whether they would store and reuse IP addresses or resolve them 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, or dynamic IP usage, which could affect firewall rule effectiveness and logging clarity.

To address these uncertainties, early prototypes focused on:

- Creating simple scripts to parse firewall logs
- Observing device behaviour when placed in a restricted network
- Testing hostname resolution patterns
- Understanding how network traffic is structured under different conditions

3.2.1. Firewall Configuration During Prototyping

For the initial testing environment, the following configuration strategy was applied to the OPNsense firewall:

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, denying all traffic not explicitly allowed, both inbound and outbound. This ensured that newly connected devices would not have any internet access unless a rule was programmatically defined.

3.2.2. 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 attempts successfully returned a list of firewall rules that were previously created via the API. Rules configured manually through the web interface, however, were not visible through the API, this seemed to be acceptable as the final system will manage all rules programmatically.

Early API calls for log retrieval were unsuccessful, but after additional testing with alternative endpoints and improved request formatting, access to firewall logs was successfully achieved. This confirmed that sufficient data could be programmatically accessed to support the planned functionality of the final system.

3.2.3. MAC Address Handling and Device Identification

During early tests, a real-world scenario involving an Apple iPhone and Apple Watch highlighted an important challenge for the planned firewall rule management system: the dynamic and unpredictable behaviour of MAC address assignments.

When connecting an iPhone to the forensic WLAN network for the first time, it was observed that the device appeared with two distinct IP addresses and MAC addresses within the same time frame. Initially, this raised concerns about MAC address randomization[24], a known privacy feature in modern mobile operating systems. However, further investigation revealed that the second MAC address and IP pair belonged to 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 maintained a consistent MAC address (`02:89:49:2b:5c:f2`). Not as expected that it would change its MAC after each session due to iOS's default privacy settings. Nevertheless, this incident illustrated the need for the firewall management system to gracefully handle MAC address changes, particularly for devices that may exhibit multiple addresses over time.

To mitigate this, the concept of a centralized device identifier was noted for later implementation: each device is assigned a unique “Asservaten Nummer” (an evidence ID). All known MAC addresses associated with a device are then linked to this ID in the database.

3.2.4. 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 Ips. For example, repeated connection attempts 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 were visible in the logs, which initially suggested minimal or idle network behaviour.

Upon further investigation, it became clear that these blocked requests were a result of the iPhone's attempt to resolve DNS queries using the default gateway IP address (for example, the firewall). This is standard behaviour in most networked devices when no external DNS server is explicitly defined. Apple mobile phones, by default, automatically use the network's assigned DNS server. The system automatically assigns the router as the DNS resolver, expecting it to forward queries or to act as a local caching DNS server.
This highlighted an important architectural requirement:
DNS resolution must be permitted for connected devices; otherwise, hostname lookups will fail, and as a result, many applications will not initiate outbound connections to services.
To address this, the integrated "Unbound DNS" service in OPNsense was configured to listen on port 53. Extended logging was enabled to ensure visibility into all DNS requests. Additionally, a port forwarding rule was implemented to redirect all DNS requests from the local network, regardless of their original 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 requests will still be intercepted and resolved locally. As a result, no modifications are required on the seized devices themselves.

3.2.5. Impact of Apple Private Relay on Firewall Logfiles

During further tests, it was observed that when accessing websites such as 20min.ch or google.ch using the Safari browser on an iPhone, the expected destination IPs were not present in the firewall logs. Instead, the logs consistently showed traffic directed toward Apple-owned IP addresses.
Upon further analysis, it was determined that the test device had Apple iCloud Private Relay[25] enabled. This feature just available to users with a paid iCloud subscription, acts as a privacy-preserving proxy system. When active, it masks the destination IP addresses of web requests made through Safari, routing traffic through 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.

3.2.6. IP Enrichment and Company Identification

During testing, it quickly became evident that a raw list of IP addresses alone 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 was ok and should be added to the allowed firewall rules list. Without additional data such as hostnames, ISP information, or service ownership - manual evaluation was not a deal. As a result, an automatic ip enrichment with additional information was needed.

Initial Attempt - Reverse DNS Lookup

The first approach implemented was a reverse DNS lookup, using the built-in socket library in Python. In some cases, this method generated helpful results, with hostnames clearly indicating the associated service or company (e.g., **`whatsapp-chatd-edge-shv-01-zrh1.facebook.com`**, **`80-67-82-201.deploy.static.akamaitechnologies.com`**). However, this method proved unreliable:

- Many IP addresses did not resolve to any hostname at all
- Others resolved to generic or uninformative names
- Reverse DNS lacks consistency across regions and providers

Improved Approach: IP Metadata API

To overcome these limitations, the system was extended to query external IP metadata APIs to collect more reliable and structured information per IP address. Several services were evaluated see Table 2, based on functionality, rate limits and pricing the service **`ip-api.com`** was selected for developing. The service ipinfo.io has also a great offer but with a really limited query answer on the free tier.

Table 2: IP API Service overview

Service	Free	Commercial use	Requests	Note
ipregistry.co[26]	Limited	Yes	100’000	100’000 requests in free account
ipapi.com[27]	Yes	Yes	0.8h / 100m	
ipwhois.io[28]	Yes	No	13.8h / 10’000m	
geo.ipify.org[29]	Limited	Yes	500	500 requests in free account
ipgeolocation.io[30]	Yes	Yes	41.6h / 30’000m	
ipdata.co[31]	Yes	No	62.5h / 45’000m	
ip-api.com[32]	Yes	No	2700h / 1’944’000m	45 requests per minute
ipapi.co[27]	Yes	No	41.7h / 30’000m	
ipinfo.io[33]	Yes	Yes	Unlimited	Limited details

ip-api.com was chosen due to:

- Detailed ISP data (useful for grouping by company)
- A clear and usable free tier (45 requests per minute)
- And an affordable premium option ($13.30/month) with no rate limits, suitable for future production use.

To respect the free tier limits during development, API requests were sent with a 1.4-second delay between them. This strategy ensured stability and prevented running into the rate limit. The following Figure 2 shows the response from the API for **`52.97.201.242`**.

Figure 2: Query with 52.97.201.242 to 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 **`isp`**, **`org`**, **`as`**, **`asname`** and **`country`**. The ISP field in particular proved to be reliable for associating IPs with companies.

3.2.7. Apple Find My Device Tests

Further tests were made using the same iPhone as earlier connected to the project WLAN with all traffic blocked by default. The phone 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 when isolated from direct internet access.
Contrary to initial expectations, the iPhone did not behave like an AirTag. While the MacBook issued a "play sound" command via iCloud, the iPhone which was network-isolated, did not update the own location or play 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 was observed in this context.
This was also observed by Josh Hickman, author of the Binary Hick blog, who tested similar conditions with iOS 15 and concluded that powered-off or disconnected iPhones do not receive Find My commands unless they have direct internet access.[34] His post “iOS 15 powered-off tracking & remote bombs” highlights that, unlike AirTags, iPhones do not appear to receive location updates or alerts via Bluetooth relays from nearby Apple devices, at least not in network-isolated test scenarios.

3.2.8. Verification of Basic Use Cases

To verify the real-world usability of the console prototype, a few easy tests were made. The test device was again the Apple iPhone. The device was prepared by disabling iCloud Private Relay, deactivating MAC address randomization, and closing all running applications. It was then connected to the WLAN and left idle for an extended period (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 were directed to over 50 unique IP addresses, all within the `17.0.0.0/8` block. The addresses were assigned to "Apple Inc" (isp field from `ip-api.com`). This background traffic indicates that even in an idle state, iOS devices initiate a wide range of communication attempts, likely for sync, analytics, or push notification purposes.
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. In addition, 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 amount of traffic is generated by the system itself, independent of user interaction.

Testing Real Application Scenarios

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

WhatsApp

The WhatsApp application was opened after the device had connected and stabilized, as described in the previous section. Initially, the interface remained in a loading state, and new messages did not appear. The system detected a new blocked IP address, `157.240.0.61`, which was resolved to `Facebook, Inc.` (the company behind of WhatsApp). After manually adding a PASS rule for this IP, the app immediately ended the loading state and began retrieving content.

Further interaction with the app revealed three additional Facebook-owned IP addresses, each of which was also added to the allowlist. Once all four IPs were permitted, full functionality of the app was restored. The final firewall rules required for full WhatsApp functionality were as follows:

Table 3: 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 repeated for Telegram. After launching the app, an initial blocked IP, `149.154.167.91`, was detected and allowed. This enabled the loading of chat content. When attempting to view an image that was not stored locally, another IP, `149.154.167.222`, appeared in the blocked log. After allowing this IP, the image downloaded successfully.

Table 4: 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 allows forensic examiners to bring devices online in a controlled, observable, and auditable manner without granting unrestricted internet access. It also confirms that at least one core use case - enabling and monitoring messaging app behaviour - is functional with the current implementation.

3.2.9. 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.

Settings on the evidence device:

- Disable Proxy Setup (e.g. Apple Private Relay) - These can hide real network activity and interfere with IP tracking.
- Disable MAC Address Randomization - Any feature that changes the MAC address (such as "Private Wi-Fi Address" on iOS) should be turned off to maintain consistent identification across network sessions.

In addition, the DHCP server should be configured to maintain IP leases for an extended period. This minimizes the assignment of new IP addresses to the same device and minimize the need for later adjustments in device management.

3.2.10. Conclusion of Section 3.2:

With the successful validation of the proof-of-concept through extensive prototyping and testing, it was demonstrated that the core use cases - such as selective app-based access, IP logging, and rule enforcement - can be reliably supported in the planned network environment. These results confirmed the technical possibility of the project and laid a solid foundation for the next phase of development.

The following sections describe the full implementation of the system.

3.3. Backend Implementation

The system's backend is organized into three primary functional areas: DHCP, Firewall and Logs. Each area has a dedicated module. The underlaying data model supports all system logic.

3.3.1. Backend Data Schema

The system's data model is composed of the tables **`DestinationMetadata`**, **`Device`**, **`DeviceAllowedISP`**, **`DeviceLease`**, **`DNSRecord`**, **`FirewallLog`**, **`FirewallRule`**, **`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, support for concurrent write operations, and reliable transactional behaviour. During development and prototyping, SQLite was used due to its simplicity and zero-configuration setup.

Django abstracts database-specific operations, allowing seamless transitions between SQLite and PostgreSQL with minimal effort. Using simple commands such as **`make-`**

migrations and **migrate**, the database schema can be initialized or adapted automatically, making it easy to develop with SQLite and deploy to PostgreSQL without code changes.

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

3.3.2. 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 dynamic IP assignments.

3.3.3. api_firewall_sync.py - Device-Based Firewall Enforcement

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

- Determining the currently active IP address of a device using information from the **`DeviceLease`** model.
- Referencing the allowed ISPs from **`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.

3.3.4. api_firewall.py - Firewall API Abstraction

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`** - adding new rules
- **`delete_rule_by_uuid`** - deleting rules
- **`check_rule_exists`** - checking for existing rules
- **`apply_firewall_changes`** - applying changes to the firewall
- **`source_ip_adjustment`** – change source ip to a new source ip

The coordination module **`api_firewall_sync.py`** delegates firewall-related actions to this API module, allowing the core logic to remain focused on rule evaluation without handling protocol details.

3.3.5. 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 and **`ip-api.com`**, and stores structured log data into the **`FirewallLog`** model.
Additionally, it maintains an in-memory cache **`config.IP_TABLE`** for fast enrichment lookups and updates the **`DestinationMetadata`** and **`MetadataSeenByDevice`** tables to track visibility and relationships between IP addresses and Metadata.

3.3.6. ssh_dns_parser.py – Live DNS Parsing

This module is responsible for parsing 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 OPNsense Unbound DNS does not provide an accessible API for live DNS logging, a different strategy had to be implemented. For this reason, DNS parsing is handled in a dedicated, standalone module.

Overview of Functionality

- Establishes a remote SSH connection to the OPNsense firewall and executes a persistent **`tcpdump`** session on interface igc1 to monitor DNS traffic **`tcpdump -i igc1 port 53 -n -l`**
- Parses incoming log lines into DNS queries and responses
- Buffers unresolved queries in memory with a short expiration time
- Matches responses to previously seen queries
- Extracts resolved IPs from responses and stores them in the **`DNSRecord`** database model.
- Updates **`last_seen_at`** on repeated observations of known records.

The following illustrates a real-world DNS resolution captured by the **`dns_ssh_parser.py`** module for an Azure cloud domain:

```
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:   onedscolprdeus19.eastus.cloudapp.azure.com. -> 52.168.117.175
```

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

3.4. Frontend Implementation

The frontend design process began with a simple hand-drawn sketch, which helped visualize the intended layout and interaction flow. It quickly became clear that the interface should be divided into several logical sections: a header, a main content area (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, keeping the layout uniform.
It is implemented using Django's templating system, combining standard HTML and CSS with Django views and models. See Figure 5 for an overview about the dashboard. The header includes two main blocks: Views and Tools:

- Views: This category includes data-driven interfaces such as log overviews (blocked or allowed traffic) and grouped views.
- Tools: Includes interactive utilities for managing evidence devices, viewing leases, performing lookups, and manually adjusting firewall rules.

A device selection dropdown is positioned on the left side of the header. This dropdown enables the user to filter all views and interactions based on the selected device.

3.4.1. 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 views
- Custom Django views for interactive logic

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

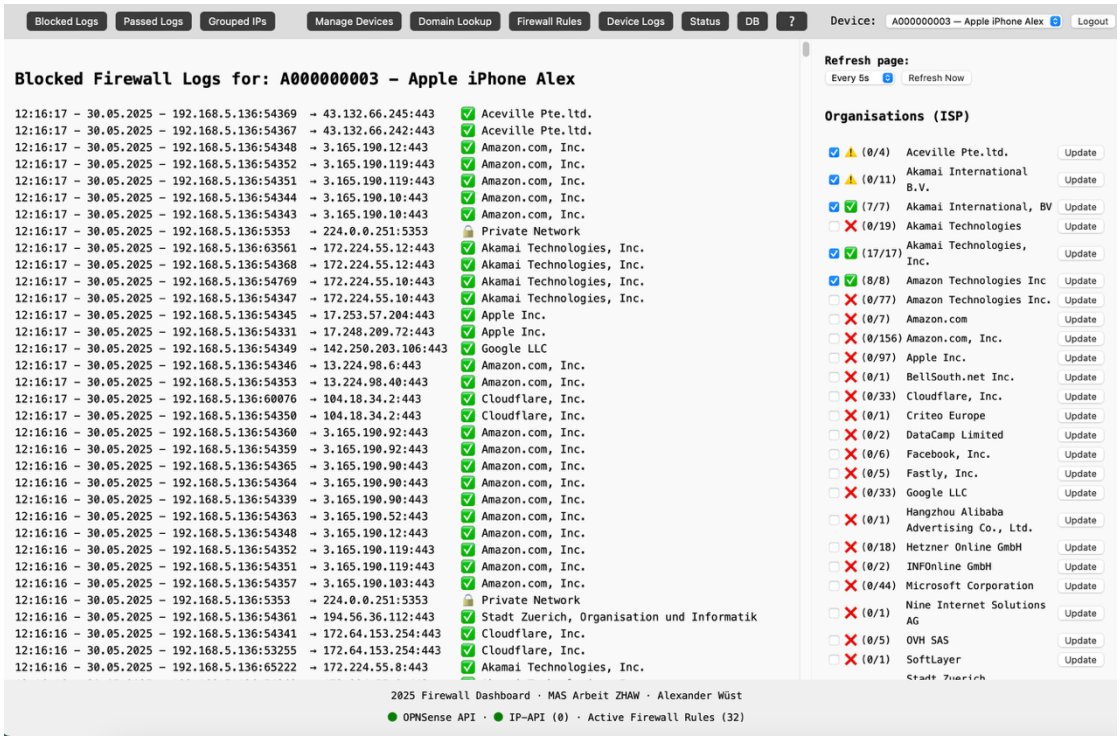

Figure 5: Frontend interface – Blocked Logs View.

3.4.2. Backend Integration

All frontend views are backed by Django function-based views that interact with the backend modules described in section 3.3. For instance:

- The Log Viewer retrieves enriched data from the **`FirewallLog`** and **`DestinationMetadata`** models, which are continuously updated every few seconds with newly parsed log entries. IP enrichment is performed simultaneously in a separate thread, ensuring that metadata such as isp, location, and DNS information becomes available with minimal delay.
- Actions in the "Organisations (ISP)" side panel triggers logic in **`api_firewall_sync.py`**, which in turn uses **`api_firewall.py`** to apply, read, adjust or delete firewall rules via the OPNsense API.
- Device and lease data is linked dynamically through the **`DeviceLease`** model.

This integration ensures consistency between the frontend and backend and allows forensic examiners to retrieve almost live network data with short delay.

3.4.3. Functional Views

The frontend consists of several specialized views. Below are the key components:

3.4.4. Blocked Logs View

This view displays all blocked network connections associated with the currently selected device. Each log entry is presented in a single-row format, providing the following key information: **`timestamp`**, **`date`**, **`source IP address`**, **`destination IP address`**, and enrichment details. The enrichment field displays either a placeholder such as "Unknown, lookup pending", indicating that metadata is not yet available or the name of the associated organization **`isp`** once enrichment has been completed like **`Amazon.com, Inc.`**.

In the left side panel, an Update dropdown allows users to configure automatic page refresh intervals, ensuring that the displayed log data remains current without manual intervention. Alternatively, a manual refresh button is also provided to trigger an immediate reload of the view on demand.

Just below users can activate or deactivate checkboxes next to each listed **`isp`** in the sidebar to define which ISPs should be allowed for the selected device. These selections are not immediately enforced but are stored in the **`DeviceAllowedISP`** model. When the Update button next to each **`isp`** is pressed, the backend is triggered to process the changes, adding or removing the corresponding firewall rules as needed. At the bottom there is a button for all **`isp`** fields which will force all isp's to be updated. This operation may take a noticeable amount of time, especially in cases where hundreds of destination IPs need to be evaluated and updated within the system and on the OPNsense firewall.
To improve clarity regarding firewall rule enforcement, icons have been added next to each entry to indicate the current rule status. These icons display the number of applied rules versus the total number of available rules—for example, 3/12 indicates that 3 out of 12 rules have been applied. This allows the examiner to quickly determine whether additional rules need to be added (e.g., 9 remaining) or removed, depending on whether the corresponding checkboxes are marked or unmarked.
This feature was implemented as a direct result of feedback following an initial demonstration conducted in our forensic laboratory.

3.4.5. Passed Logs View

This view is based on the Blocked Logs View but instead displays all allowed ("pass") traffic entries. Its primary purpose is to provide an overview of permitted network activity, offering insights into the device's communication.

3.4.6. Grouped IPs View

The Grouped IPs View, see Figure 6, differs significantly from the previously described log views. It is designed to present a live snapshot of recent IP communication, categorized by their enrichment metadata such as ISP, organization, and location.

Upper Section - Unfiltered IPs (Blue)
In the upper (blue) section, all IP addresses that were observed in the last N seconds are displayed. The time window can be selected using a dropdown menu on top, which controls the refresh scope. A "Refresh" button is available to reload the data manually, and a "Delete Seen Metadata" button allows the user to clear the current overview state and start fresh. Each IP entry includes metadata such as the time it was first seen and last seen. These timestamps reflect both successful and attempted communications. This view is particularly useful for real-time monitoring, forensic examiners can let a device run for a few minutes, then open an application to generate traffic and observe new connections as they appear.

Lower Section - Grouped by ISP (Orange)
The lower section of the interface, highlighted in orange, presents a grouped view of destination IP addresses by **`isp`**. It includes a sortable table that displays various enriched metadata fields for each IP entry. The following columns are provided:

- **`ORG`** – The name of the organization associated with the IP address.
- **`ISP`** – The organisation behind the IP address.

- **`DNS Request`** – The request which was used to lookup the ip address, with a tooltip showing additional context and resolution details.
- **`Location`** – Geolocation information such as country or region.
- **`First Seen`** – The timestamp indicating when the IP was first observed in communication.
- **`Last Seen`** – The timestamp of the most recent communication with the IP.
- **`DNS Reverse`** – The reverse DNS name associated with the IP, if the lookup was successful.

Each row also includes a button on the left side that allows the user to manually add or remove the specific IP address from the firewall rule list. The button visually indicates the current rule status: a "+" is shown if no rule has been applied yet (allowing the user to add it), and a "–" is displayed if the rule is already active (allowing the user to remove it). This mechanism provides direct, intuitive control for managing individual IP addresses.

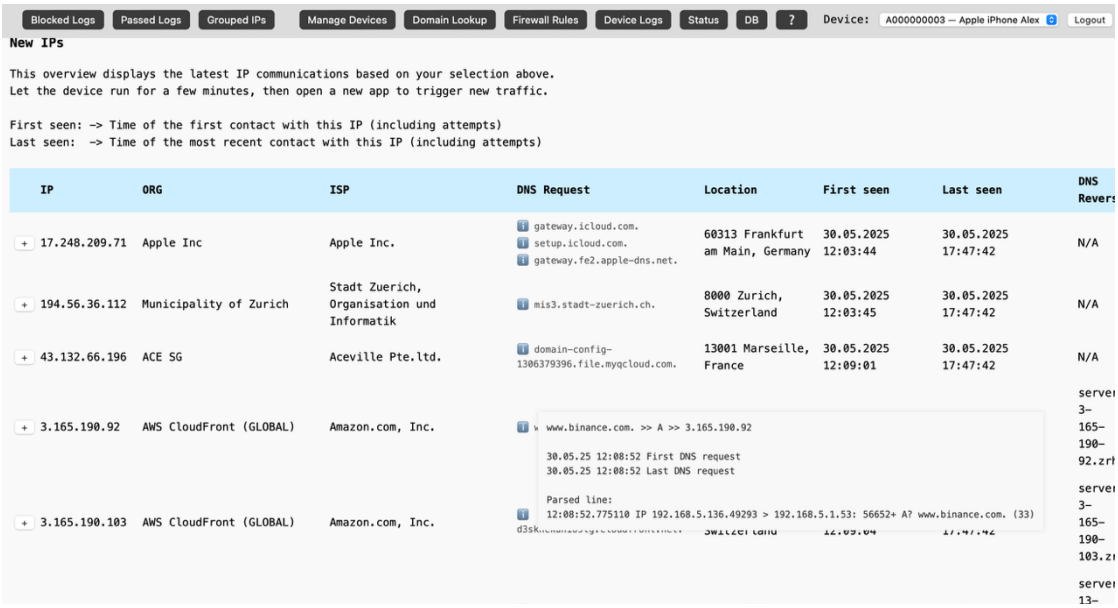

Figure 6: Frontend interface – Grouped IP's View

3.4.7. Manage Devices View

The Manage Devices view plays a central role in configuring which devices are monitored by the system, see Figure 7. It provides functionality to add, archive, and unarchive devices, as well as to assign or hide DHCP leases. Devices are required to assign at least one lease for log analysis and firewall rule enforcement in the frontend views.

Add and Approve Devices

New devices are registered by submitting the following metadata:

- Device ID (unique identifier)
- Description (like "Samsung Galaxy S22")
- Examiner (responsible forensic analyst)
- DNS server (used for DNS-based rule enforcement).

As long as a device is not archived, it is listed under Active Devices. The currently assigned network leases can be viewed by expanding the dropdown associated with each device entry.

For consistent MAC address identification, it is recommended to disable the “Private WLAN Address” feature on iOS devices. This ensures that the device maintains a consistent hardware identifier across sessions. This is mentioned under the help section which is linked directly on the page.

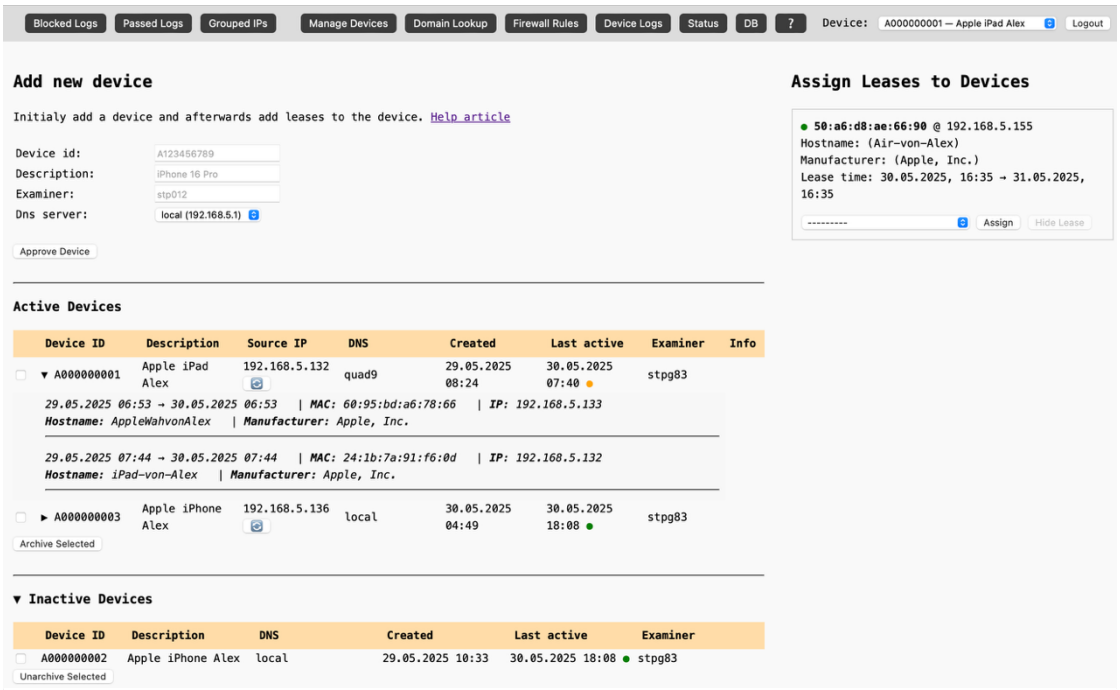

Figure 7: Frontend interface – Device Management View

Device Lists

The interface separates devices into two categories:

- Active Devices: Listed in Dropdown to assign leases and to filter on views.
- Inactive Devices: Archived devices that no longer appear in live views

Each row displays metadata such as the **`Device ID`**, **`Description`**, **`DNS`**, **`Created`** **`Last active`**, **`Examiner`** and **`Info`**. Devices can be archived or unarchived.

When a device is archived, all associated firewall rules are automatically removed. This reduces the total number of active firewall rules, helping to maintain a manageable rule set and ensuring that the API remains responsive.

Assign Leases to Devices

The right-side panel displays the available und unassigned DHCP leases. Each lease entry includes the following information:

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

Each lease can be assigned to one device only, linking it in the backend to the corresponding **`Device`** model. This allows log entries from that IP/MAC combination to be correctly associated with the device across all frontend views. Alternatively, leases can be hidden via the Hide Lease button, if the specific lease is inactive.

3.4.8. Domain Lookup View

The Domain Lookup view provides a interface for resolving a domain name into its associated IP addresses and corresponding `isp` information. This functionality supports manual investigation by allowing forensic examiner to see which infrastructure or service providers are behind a given domain. For example the Domain `binance.com` resolves to the results shown in Table 5. This information can support further decisions regarding which firewall rules should be applied.

Table 5: 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

3.4.9. Firewall Rules View

The Firewall Rules View displays all pass rules currently active for a selected device. Since the system blocks all traffic by default, only pass rules are added. The rules listed here were automatically added based on the `isp` logic or manually by the user.
Each row includes:

- Action (always "PASS")
- Protocol, Source IP, Destination IP, and Port
- Associated ISP (if available)
- Timestamp when the rule was added
- Flags for manually or due to DNS-based logic added rules
- Button to remove a existing rule

Forensic examiners can selectively remove firewall rules directly from this view. However, DNS rules are protected from deletion to preserve domain resolution for allowed services. Only manually added or automatically generated rules will be removed.

3.4.10. Device Logs View

The Device Logs view displays a detailed, timestamped log of all observed traffic for a selected device. Each entry includes:

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

For improved readability, entries are color-coded, red indicating blocked traffic and green indicating allowed traffic. This enables quick visual differentiation.
A planned enhancement will allow device-based log export functionality, enabling forensic examiner to export complete traffic records for individual devices. This export will include a timeline view that correlates the addition and removal of firewall rules with corresponding blocked and allowed connections.

3.5. Integration with OPNsense

All communication with the OPNsense firewall is handled through its REST API, allowing for remote creation, removal, and management of firewall rules, as well as access to log and DHCP lease data. The backend communicates with OPNsense over HTTPS using Python's requests library, with all API interactions abstracted into the **`api_firewall.py`** and **`api_dhcp_parser.py`** modules to maintain separation of concerns.

Authentication and Security
To ensure secure access, the system uses two mechanisms:

- API credentials the **`API_KEY`** and **`API_SECRET`** are stored in a local **`.env file`** and accessed through Django's environment configuration.
- A client-side certificate file **`certificate_crt.pem`** is used to verify the identity of the requesting system and to establish a secure connection with the firewall.

All API calls include both the HTTP basic auth credentials and the certificate for full authentication. The use of Django's settings and environment variables ensures that sensitive information is not hardcoded and remains easily configurable.

API Endpoints Used
The following OPNsense API endpoints are integrated:

- 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`**

These endpoints are wrapped in reusable helper functions within the **`api_firewall.py`** module.

Optimized Request Flow and Performance
To reduce overhead, the module maintains a shared requests.Session() instance with preconfigured authentication and headers. This avoids redundant setup for each individual API call and improves overall efficiency.

Verbose logging is available for diagnostic purposes and can be adjusted in terms of verbosity. Full debugging may produce excessive output that could overwhelm the console, especially when having higher traffic. To address this, newer components - such as DNS-related processing - have been modularized, allowing for targeted debugging. For example, DNS-specific logs can now be enabled independently.

Error Handling and Consistency

All API interactions are wrapped in exception handling using requests.RequestException to gracefully handle network-related errors. Inconsistencies between the firewall state and the local database are addressed by updating internal representations to reflect the most current known state.

Specific adjustments trigger a `verify_opnsense` flag, which is used to ensure that a rule's state remains consistent across both the OPNsense API and the Django database. If a mismatch is detected, the rule is removed - following the principle that it is generally safer to remove a rule than to risk applying an incorrect one.

All firewall rules are regularly fetched from OPNsense. Each is compared against the database using its unique UUID, which serves as the single source of truth. Any rules that still exist on the firewall but are not marked as active in the database are removed from the firewall. This ensures that no unintended rules persist.

Following this initial cleanup, the `verify_opnsense` flag is used to perform a direct 1:1 check for individual rules where needed. This hybrid approach combines the performance benefits of targeted verification with the safety of a full synchronization pass—offering a high level of consistency and persistence between systems.

3.6. 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
There are two primary mechanisms by which pass rules are introduced into the firewall:

Policy-based ISP Grouping:
When a forensic examiner approves one or more ISPs for a given device, the system identifies all previously blocked IPs associated with those ISPs and automatically creates corresponding pass rules. This ensures that traffic to approved destinations is allowed, even if previously denied.

Manual Rule Addition in the Frontend:
In the Grouped IPs View, users can manually add or remove pass rules for specific IP addresses. This allows single control over each ip address in exceptional cases, independent of the ISP grouping logic.

3.6.1. Source IP Handling and Automatic Rule Reassignment

As devices in DHCP environments frequently change IP addresses, the system actively adjusts existing rules when a device's source IP changes. If a device transitions from one IP to another like from `192.168.5.155` to `192.168.5.170`, the coordination logic:

- Detects the new lease assignment
- Reassigns any active pass rules from the previous IP to the new IP
- Updates both the firewall and the database 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.

3.6.2. Rule Evaluation and Creation Logic

The system evaluates which IPs need to be permitted based on the following logic:

1. The currently active IP of the device is determined using the **`DeviceLease`** model.
2. The user-approved ISPs for the device are fetched.
3. All previously blocked destination IPs that match these ISPs are collected.
4. For each of these destination IPs:
    - The system checks whether a rule already exists in the DB.
    - If not, it creates a new rule in the **`FirewallRule`** model and triggers an API call to add it to the firewall.

If the rule cannot be added to OPNsense, it will not be added to the database to maintain consistency.

3.6.3. Device Archival and Rules Removal

When a device is archived:

- All active pass rules related to the device are removed from the firewall
- These rules are marked as ended in the database by setting the **`end_date`**.
- It will not show up in the header dropdown menu anymore.

When a device is later unarchived:

- No rules will be applied to the device.
- It will again show up in the header dropdown.

This behaviour ensures that inactive devices do not accumulate stale firewall entries, helping to reduce rule processing overhead.

3.6.4. Rule Cleanup and Removal

After adding new rules, the system evaluates all currently active (non-manual) rules for the device. If any of these rules point to destination IPs that no longer belong to an allowed ISP, the rule is stored for removal. These rules are then removed from the firewall and marked as ended in the database. The system uses the **`delete_multiple_rules()`** function to perform efficient batch deletions and then calls **`apply_firewall_changes()`** to finalize the updates in the OPNsense firewall.

3.7. Logging

The system does not implement a separate or dedicated logging layer. Instead, it retrieves structured data from the Django database. Log entries are sourced from the OPNsense firewall, parsed, enriched, and stored as records in the backend. This design provides reliable, queryable access to historical activity without requiring additional logging infrastructure.

The system focuses exclusively on IP-level metadata and does not perform any inspection of traffic content or payloads. Its primary purpose is to document which destination IP addresses a device attempted to contact, whether the connection was allowed or blocked.

3.7.1. 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 active or removed?
- Which IPs did the device attempt to contact, and were those connections allowed?
- Which ISP was responsible for a given destination IP?

3.7.2. Database-Backed Structure

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

- The `FirewallLog` model stores raw connection attempts (blocked or allowed), enriched with IP-level data.
- The `DestinationMetadata` model holds enrichment results such as ISP, geolocation, DNS, and organization.
- The `FirewallRule` model records rule application, including timestamps for creation and removal.
- The `MetadataSeenByDevice` model links enriched IPs to specific devices for grouping and time-based visibility.

Together, these models allow full historical reconstruction of a device's observed network activity and the corresponding firewall policy state.

3.7.3. Operational Visibility

Frontend views such as the Blocked Logs View, Grouped IPs View, and Device Logs View rely entirely on this stored data. Color-coded entries indicate blocked (red) and allowed (green) connections. 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.

3.7.4. Planned Enhancements

Several features are planned to extend the usefulness of the logging system:

- Export of firewall rule history per device, including timestamps for when rules were added or removed
- Export of blocked and allowed connection history for documentation or even audit purposes
- Export of firewall rules history for documentation

These additions are intended to support forensic analysis and formal reporting.

3.8. 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 summarizes the most relevant problems and the corresponding solutions or workarounds.

3.8.1. 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.

3.8.2. IP Enrichment Bottlenecks

The system uses ip-api.com to enrich destination IPs with metadata such as ISP, organization, and location. 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 enrichment request. 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.

3.8.3. 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. REFERENCE TO CHAPTER 4 ---- OTHER APPROACH

3.8.4. MAC Address Randomization and IP Variability

Modern devices, particularly iOS and macOS, 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.

3.9. Summary

This chapter showed detailed the implementation of the prototype system developed during this thesis. Starting from a concept validated through preliminary testing, the solution evolved into a modular Django-based application that interacts with the OPNsense firewall through its REST API. Core components include the backend logic for dynamic rule management, IP enrichment, and log parsing, a database-backed architecture for persistent state tracking, and a frontend that enables forensic examiners to control, monitor, and visualize device-specific network activity in near real time.
Key challenges addressed during development included handling device variability (MAC/IP changes), managing external API limits, and overcoming performance issues when applying large rule sets. Despite these limitations, the system successfully demonstrates that, IP-based firewall control for mobile devices is technically possible - without relying on deep packet inspection or intrusive monitoring.

4. Analysis and Results

4.1. Use Case Demonstrations

This section presents practical demonstrations of 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 first describe the shared test environment and device preparation, followed by several representative use cases based on common mobile applications (e.g., WhatsApp, Telegram).

4.1.1. Test Environment Setup

This section describes the general environment and device configuration used for all following use cases. All tests were conducted in a controlled WLAN environment with default-deny firewall rules, requiring manual approval of outbound connections.

Network and System Configuration

- Firewall: OPNsense (Ultimate Unicorn 25.1) with all outbound traffic blocked by default
- Access Point: TP-Link EAP610, isolated VLAN
- Web Application: Custom Django-based interface running locally
- Rule Handling: Rules applied via OPNsense API
- IP Enrichment: Done via ip-api.com with memory/database caching
- DNS: Only approved DNS resolvers (1.1.1.1, 9.9.9.9) allowed

Device Preparation (Used for All Tests)

Device	iOS Test Phone	Android Test Phone
Model		
OS Version		
DNS		
Private Relay		
MAC Random.		
Notes		

Figure 8: Test Devices Overview

4.1.2. WhatsApp Access (iOS / Android)

Objective

The objective of this test is to evaluate whether WhatsApp can operate under a restrictive firewall environment by selectively allowing only the required IP connections. Specifically, the test aims to verify the following:

- Access to the chat overview after launching the app
- Retrieval of actual, uncached chat messages from the server
- Download and playback of previously received media content that is no longer stored locally on the device

Steps

Connected iPhone to WLAN with all traffic blocked by default.

Opened WhatsApp; observed no messages loading.
Monitored Blocked Logs View and grouped IPs by ISP.
Approved “XXXXX” in the sidebar.
Waited for backend rule application (~2 minutes).
Result

Observations

4.1.3. Telegram Access (iOS / Android)

Objective

The objective of this test is to evaluate whether Telegram can operate under a restrictive firewall environment by selectively allowing only the required IP connections. Specifically, the test aims to verify the following:

- Access to the chat overview after launching the app
- Retrieval of actual, uncached chat messages from the server
- Download and playback of previously received media content that is no longer stored locally on the device

Steps

Connected iPhone to WLAN with all traffic blocked by default.
Opened Telegram; observed no messages loading.
Monitored Blocked Logs View and grouped IPs by ISP.
Approved “XXXX” in the sidebar.
Waited for backend rule application (~2 minutes).
Result

Observations

4.1.4. Potato Access (iOS / Android)

Objective

The objective of this test is to evaluate whether Potato can operate under a restrictive firewall environment by selectively allowing only the required IP connections. Specifically, the test aims to verify the following:

- Access to the chat overview after launching the app
- Retrieval of actual, uncached chat messages from the server
- Download and playback of previously received media content that is no longer stored locally on the device

Steps

Connected iPhone to WLAN with all traffic blocked by default.
Opened Potato; observed no messages loading.
Monitored Blocked Logs View and grouped IPs by ISP.
Approved “XXXX” in the sidebar.
Waited for backend rule application (~2 minutes).
Result

Observations

4.1.5. Photos App – iCloud Media Retrieval (iOS)

Objective

The objective of this test is to evaluate whether iOS Photos app can operate under a restrictive firewall environment by selectively allowing only the required IP connections. Specifically, the test aims to verify the following:

- Verifying if thumbnails and full-resolution images not cached on the device can be fetched from iCloud
- Identifying which IP addresses or ISPs are involved in the iCloud media download process
- Examining whether iCloud connectivity can be restricted in such a way that prevents an Apple remote wipe command from reaching the device, while still allowing access to cloud-based photos

Steps

Connected iPhone to WLAN with all traffic blocked by default.
Opened Photos App; observed the pictures are not loaded.
Monitored Blocked Logs View and grouped IPs by ISP.
Approved “XXXXX” in the sidebar.
Waited for backend rule application (~2 minutes).

Result

Observations

4.1.6. Binance.com – Account Access and Transaction Attempt (iOS / Android)

Objective

The objective of this test is to evaluate whether the binance.com app can operate under a restrictive firewall environment by selectively allowing only the required IP connections. Specifically, the test aims to verify the following:

- Retrieval and display of current account balance and recent transactions
- Attempt to initiate an outgoing transaction (withdraw funds to a wallet)

Steps

Connected iPhone to WLAN with all traffic blocked by default.
Opened Binance App; observed the pictures are not loaded.
Monitored Blocked Logs View and grouped IPs by ISP.
Approved “XXXXX” in the sidebar.
Waited for backend rule application (~2 minutes).

Result

Observations

4.2. Firewall Rule Propagation Delay

When applying a large number of rules via the OPNsense API, such as after toggling an ISP group with hundreds of destination Ips, the system experienced notable delays. During this period, the frontend appeared unresponsive, and the firewall did not yet enforce the new rules.

Findings:

Empirical testing showed that:

Adding a single rule takes approximately 0.2 seconds when the firewall has no active rules.

As the number of active rules increases, latency also increases:

+1 second per rule with several hundred rules active

+2 seconds per rule when the rule set exceeds 700 entries

Removing rules takes between 0.2 and 1.2 seconds, depending on the total number of active entries.

The final apply step becomes slower with more rules, as each rule is verified by OPNsense before being committed.

In a stress test involving the application of 600 rules at once, the complete process took 10 to 15 minutes, which is not typical for everyday usage but becomes relevant in multi-device or bulk-editing scenarios.

Workaround:

Session reuse and selective API call reduction helped reduce latency. However, the OPNsense API does not support batch creation or deletion of rules in a single request.

Future Consideration:

Research in the OPNsense community forum suggests an alternative approach: generating a full firewall config file, injecting all rules at once, and re-uploading the config. This method bypasses the API's sequential behaviour and could be significantly faster when working with larger rule sets (e.g., 30–50+ rules). There were no tests made yet with this approach.

5. Discussion and Conclusion

Remote Wipe… some research business device…

References

[1] M. Mohler, ‘Der Faradaysche Käfig’. Accessed: Jan. 13, 2025. [Online]. Available: https://lp.uni-goettingen.de/get/text/833

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

[3] 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

[4] 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

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

[6] ‘nftables wiki’. Accessed: May 14, 2025. [Online]. Available: https://wiki.nftables.org/wiki-nftables/index.php/Main_Page

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

[8] 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/

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

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

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

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

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

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

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

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

[17] ‘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

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

[19] ‘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

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

[21] ‘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

[22] ‘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
[23] ‘Android Business Device Solutions Directory - Android Enterprise - EMMs’, Android Enterprise. Accessed: May 15, 2025. [Online]. Available: https://androidenterprisepartners.withgoogle.com/emm/
[24] ‘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
[25] Apple Inc, ‘iCloud Private Relay Overview’, p. 11, Dec. 2021.
[26] Elaunira SARL, ‘Affordable IP Geolocation and Threat Intelligence Pricing - Ipregistry’. Accessed: May 15, 2025. [Online]. Available: https://ipregistry.co
[27] Kloudend, Inc., ‘ipapi - IP Address Lookup and Geolocation API | No SignUp’. Accessed: May 15, 2025. [Online]. Available: https://ipapi.co
[28] ipwhois.io, ‘IP Geolocation API - Pricing’. Accessed: May 15, 2025. [Online]. Available: https://ipwhois.io
[29] ipify.org, ‘IP Geolocation API - Try Our IP Location API Free Of Charge’. Accessed: May 15, 2025. [Online]. Available: https://geo.ipify.org
[30] JFreaks Software Solutions, ‘IP Geolocation API Pricing’, IP Geolocation API Pricing. Accessed: May 15, 2025. [Online]. Available: https://ipgeolocation.io
[31] ipdata.co, ‘IP Geolocation API with Threat Intelligence’. Accessed: May 15, 2025. [Online]. Available: https://ipdata.co
[32] ip-api.com, ‘IP-API.com - Geolocation API’. Accessed: May 15, 2025. [Online]. Available: https://ip-api.com
[33] IPinfo, ‘Trusted IP Data Provider, from IPv6 to IPv4’. Accessed: May 15, 2025. [Online]. Available: https://ipinfo.io
[34] 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/

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.