CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

A Monitoring System for Laboratory Workstations

Bachelor's thesis in Computer science and engineering

Nils Gralén
Albin Karlsteen
Nicholas Menor Löwegren
Edvin Palmqvist
Elof Paulsson

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2026

Bachelor's thesis 2026

A Monitoring System for Laboratory Workstations

Nils Gralén
Albin Karlsteen
Nicholas Menor Löwegren
Edvin Palmqvist
Elof Paulsson

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2026

Nils Gralén Albin Karlsteen Nicholas Menor Löwegren Edvin Palmqvist Elof Paulsson

Supervisor (Handledare): Ilias Chanis, Computer Science and Engineering
Examiners: Patrik Jansson and Arne Linde, Department of Computer Science and Engineering
Graded by teacher (Rättande lärare): Örjan Askerdal, Department of Computer Science and Engineering

Bachelor's Thesis 2026
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2026

Nils Gralén, Albin Karlsteen, Nicholas Menor Löwegren, Edvin Palmqvist, Elof Paulsson

Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

University computer laboratories rely on a large number of shared workstations that must remain operational and available to students. However, hardware-related incidents, such as component theft, unexpected shutdowns, and network disconnections, can be difficult to detect using IT monitoring solutions. The proposed monitoring system was implemented as a centralised client-server architecture consisting of lightweight monitoring clients, a central server, a persistent database, and a graphical user interface for administrative oversight. The system combines heartbeat monitoring, hardware validation, chassis intrusion detection, and automated recovery mechanisms to provide centralised situational awareness and alarm handling. The system was evaluated through passive testing and active threat simulations performed on laboratory computers at Chalmers University of Technology. The evaluation demonstrated successful detection of simulated tampering events, stable heartbeat communication, low resource utilization, and reliable alarm handling with minimal interference with normal laboratory usage. Although the system remains a proof of concept prototype, the project demonstrates that event-driven hardware monitoring can provide an effective foundation for improving hardware security and operational awareness in shared university computer laboratory environments.

Sammanfattning

Datalaboratorier vid universitet är beroende av ett stort antal delade arbetsstationer som måste förbli i drift och tillgängliga för studenter. Hårdvarurelaterade incidenter, såsom komponentstöld, oväntade avstängningar och nätverksavbrott, kan dock vara svåra att upptäcka med IT-övervakningslösningar. Det föreslagna övervakningssystemet som utvecklades implementerades som en centraliserad klient-server-arkitektur bestående av övervakningsklienter med låg utnyttjande av berökningsresucurser, en central server, en databas för persistent lagring och ett grafiskt användargränssnitt för administrativ övervakning. Systemet kombinerar pulsslagövervakning, hårdvaruvalidering, detektering av chassintrång och automatiserade återställningsmekanismer för att ge centraliserad situationsmedvetenhet och larmhantering. Systemet utvärderades genom passiv testning och aktiva hotsimuleringar utförda på laboratoriedatorer vid Chalmers tekniska högskola. Utvärderingen visade att systemet kunde detektera simulerade manipuleringshändelser, stabil pulsslagskommunikation, låg resursutnyttjande och tillförlitlig larmhantering med minimal störning vid normal laboratorieanvändning. Även om systemet fortfarande befinner sig på prototypstadiet visar projektet att händelsestyrd hårdvaruövervakning kan ge en effektiv grund för att förbättra hårdvarusäkerhet och driftsmedvetenhet i delade universitetsmiljöer för datorlaboratorier.

Keywords: Physical monitoring system, computer monitoring, monitoring system, client-server, Spring boot, JAVA, Go, Intel Active Management Technology.

Acknowledgements

We would like to express our gratitude towards our supervisor Ilias Chanis for his guidance and thoughtful feedback throughout the development of the project. His feedback and technical expertise have been instrumental in shaping both the direction and quality of this work.

We also extend our appreciation to Lars Noren for providing us the necessary equipment and valuable insights throughout the project.

Finally, we would like to thank the IT Department at Chalmers University of Technology for their cooperation, practical support and useful insights provided during the interview.

Nils Gralén Albin Karlsteen Nicholas Menor Löwegren Edvin Palmqvist Elof Paulsson, Gothenburg, June 2026

Contents

Glossary

Term	Description
ACID	Database properties ensuring Atomicity, Consistency, Isolation and Durability of transactions.
API	Application Programming Interface. A defined interface used for communication between software systems.
API Key	Authentication token used by clients when communicating with the server.
Client	Software running on a monitored host responsible for collecting telemetry and reporting events.
DBMS	Database Management System. Software used to store and manage structured data.
DCM	Dell Command Monitor. Dell-specific software used for monitoring and configuring firmware settings.
DTO	Data Transfer Object. Design pattern used to transfer structured data between system layers.
GUI	Graphical User Interface used by administrators to monitor devices and alarms.
Host	A monitored computer running the client software.
HTTP	Hypertext Transfer Protocol used for communication between networked systems.
HTTPS	Secure version of HTTP using encryption through TLS.
ICS	Internet Calendaring and Scheduling format used to import room booking information.
JDBC	Java Database Connectivity. Standard interface for communication between Java applications and databases.
JSON	JavaScript Object Notation. Lightweight format used for exchanging structured data.
ME	Management Engine, Independent hardware subsystem in Intel AMT-enabled computers used for remote management.
CCM	Client Control Mode, AMT provisioning mode that allows for restricted AMT access.
ACM	Admin Control Mode, AMT provisioning mode that allows for full AMT access.
MITM	Man-in-the-Middle Attack, where communication is intercepted or modified by an attacker.
NGINX	Web server and reverse proxy used to handle HTTPS communication and forward requests to the backend.
ORM	Object-Relational Mapping. Technique for mapping database tables to programming language objects.
PostgreSQL	Relational Database Management System used by the monitoring system.
REST API	Architectural style where resources are accessed through standard HTTP methods.

Term	Description
Reverse Proxy	A server that receives requests and forwards requests to back-end services.
RPC	Remote Procedure Call. Mechanism for executing procedures on a remote system.
SSE	Server-Sent Events. Technology used to push real-time updates from the server to the GUI.
Supabase	Cloud platform used during development for hosting the PostgreSQL database.
TLS	Transport Layer Security. Cryptographic protocol used to secure network communication.
VPro	Intel platform branding indicating support for enterprise management technologies such as AMT.
Windows Service	Background application managed by Windows that runs independently of user sessions.
WMI	Windows Management Instrumentation. Windows framework used for accessing system information and events.
WS-Man	Web Services Management. Protocol used for communication with Intel AMT.

1
Introduction

1.1 Background

Universities and higher education institutions rely heavily on computer laboratories to support teaching, coursework, laboratory exercises, and independent student work. At Chalmers University of Technology, a large number of computers are distributed across multiple computer laboratories, many of which are accessible for extended hours. The availability and reliability of these systems are therefore critical to ensure that students can complete their studies without unnecessary interruptions.

In recent years, recurring issues related to theft of hardware components, such as memory modules and graphics cards have been observed in these computer laboratories. Such incidents often result in computers becoming unusable, sometimes without immediate detection. These issues make it difficult for IT personnel to ensure that computers are operational when needed and to respond quickly to abnormal events. Although these computers are equipped with built-in monitoring capabilities, the information provided by these systems is often fragmented and difficult to aggregate. The lack of a centralised monitoring solution makes it challenging to obtain a clear overview of the current status of computer laboratories, detect abnormal events in a timely manner, and distinguish between normal usage patterns and abnormal behaviour. As a result, abnormal events such as unexpected power loss, network disconnects, or hardware removal may go unnoticed for an extended period of time.

There is therefore a need for monitoring solution that provides a centralized overview of computer labs' status, detects unusual events such as power loss or network disconnects, and supports a timely response from IT and security personnel.

1.2 Purpose

The purpose of this project is to design and implement a monitoring system for computer laboratories capable of detecting and reporting abnormal hardware-related events, such as unexpected power loss or network disconnections. The system aims to provide a centralised overview of laboratory rooms and their connected devices in order to improve situational awareness for IT staff and enable faster response to potential incidents, including hardware tampering or theft.

As laboratory computers, henceforth referred to as hosts, already run software-based monitoring tools, this project does not focus on software-level security mechanisms such as malware detection, intrusion prevention, or data exfiltration protection. The scope is limited to hardware and infrastructure monitoring. The system is not intended to replace existing solutions or serve as a comprehensive cybersecurity system for the Chalmers computer laboratories. The final product that is delivered by this project is intended as a prototype rather than a production ready system.

1.3 Existing Solutions

This section reviews existing IT asset monitoring and physical security solutions to investigate the need for a custom built monitoring system. In order to identify any gaps in existing solutions, the software is evaluated against this project specific threat model, as detailed in 2.1. During an interview with the IT department at Chalmers, they mentioned that they currently do not utilize any software to detect or prevent physical hardware theft. Instead, their existing security measures focus primarily on digital usage, especially monitoring network traffic to prevent access to illicit or illegal digital content.

1.3.1 Remote Monitoring and Management

Remote Monitoring and Management (RMM) is utilized by IT administrations to remotely track hardware health, network connectivity, and software patching in near real-time. By deploying local agents on the monitored devices, RMMs continuously gather telemetry from the system. When an abnormal event is detected, the system automatically triggers alerts via administrative dashboards or email notifications [1].

Examples of such software include ConnectWise Automate [2] and PRTG Network Monitor [3]. According to the official Paessler documentation, PRTG utilizes various "sensors" on the endpoint devices to gather network device uptime, bandwidth usage, hardware information/status and log events among others. The central server then collects these data by querying the endpoint devices at scheduled intervals. Because these platforms are designed to track thousands of complex metrics across servers, databases, and virtual environments, deploying them strictly for physical hardware theft would introduce system bloat [3].

Although highly effective for general IT administrations, software like PRTG does not align with the requirements of this project's physical security model. These standard network monitors rely on scheduled querying intervals, which introduces unacceptable latency. This allows for the removal of a hardware component between querying cycles without immediate detection. Furthermore, these software are designed to monitor device hardware health and performance, lacking native integrations with physical security features. For example, motherboard chassis intrusion switches is one of those features not integrated, meaning custom functionality would have to be implemented to achieve the necessary event-driven alerts. Finally,

deploying such RMMs for large campus environments would introduce enterprise licensing fees.

1.3.2 Anti-Theft Software Solutions

Another approach to device security involves endpoint tracking and anti-theft software, such as Absolute Computrace [4] and Prey [5]. These software programs are primarily designed and used for post theft recovery. They rely on agent software installed on the host machine, which, if stolen, waits for the internet connection. Once internet connection is established, it uses WI-FI triangulation to locate the device or capture camera images of unauthorized users [5].

However, as highlighted in recent research, standard tracking software is flawed as it strictly relies on internet connectivity to locate a device. A custom Internet of Things (IoT) hardware attachment was proposed to bypass the need for OS-level internet connection using pressure sensors and independent GPS tracking [6].

Although the IoT approach works for laptops it does not address the specific threat model of Chalmers lab computers. These lab environments utilize desktop computers where the threat is not theft of an entire device, but rather certain internal components (such as RAM or GPU). Because these do not provide interfaces for physical hardware events, such software cannot prevent or alert on component tampering.

1.4 System Requirements

In order to evaluate the success of the proposed monitoring system, a set of requirements has been established. The project will be considered successful if the following criteria have been implemented and met:

1. Detect physical tampering of the host system and trigger a corresponding alarm.
2. Easy deployment on the host in the form of an install script, without any manual configuration outside of the script.
3. Continuously run the host while maintaining a minimal resource footprint (CPU/RAM) to prevent decrease of host performance.
4. The students should not be able to terminate the host process, or get access to any sensitive information.
5. Detect any hosts that might go offline.
6. Detect defined states in which the host might be in.
7. Make an attempt to reach and turn on the host after being disconnected.
8. Securely transmit telemetry over the network utilising HTTPS.
9. The host should successfully be forced back on the internet when manually disabled.
10. Alarm if there are multiple hosts going offline at the same time within the same room.

11. The system must generate zero false positive alarms during the testing phase to ensure the alarm log remains actionable.
12. Have an overview of the connected devices, which ones are currently online and any alarms that might have occurred.
13. Display alarms and the ability to mark them as "handled" on a Graphical User Interface (GUI).
14. The ability to create reports on certain alarms in a GUI.

2
Methodology

2.1 Threat Model

To provide a structured understanding of the problem the system aims to address, the following threat model was developed. The model is inspired by the methodology proposed by Khalil et al. [7], which provides a systematic approach to threat modelling of cyber-physical systems. For brevity, the full step-by-step methodology is not presented; instead, the following sections summarise the key outcomes relevant to this project.

2.1.1 Actors and Assets

The assets that the system is meant to protect are the many workstations that are available across the many computer laboratories across Chalmers campus. These can be further subdivided into the components that are inside the computer chassis. The assets are not identical, however the following is a standard set of policies for all computers that are accessible on Chalmers that was established during the interview with the Chalmers IT-department.

1. All assets are placed within a metal box that is secured with a commercial grade padlock.
2. All assets are only able to access the internet through a physical Ethernet wire, no other connectivity devices are installed on the asset such as Bluetooth or Wi-Fi cards.
3. All users except IT personnel are unable to alter the assets power state. I.e a regular user can not shutdown or put the computer into sleep mode.
4. All assets are Intel VPRO compatible. See section 2.3.5 for explanation of what that entails.
5. All assets are equipped with an intrusion switch that detects when the chassis is opened.

For this threat model, it is assumed that an attacker has the ability to physically access and tamper with the assets, including disconnecting power or network connectivity by removing cables. The attacker is assumed to be aware that the assets are monitored, but is not assumed to have detailed knowledge of the internal implementation of the monitoring system. However, the attacker is assumed to have sufficient technical capability to attempt to infer system behaviour and exploit observable weaknesses.

2.1.2 Trust Boundary

Since the assets are not supposed to be able to be tampered with, it is assumed that any physical intrusion into an asset chassis is hostile. Therefore the system will require information that shows if an authorised individual is currently utilizing the asset in case of legitimate hardware access. The same applies if the system is shutting down, as this is a state the asset is not supposed to enter unless an administrator has accessed it.

A more complicated issue is how to determine if an asset disconnected from the network has been tampered with or not. It is not a reasonable assumption that a network has 100% uptime, it must therefore be assumed that assets will sometimes disconnect from the network without any malicious actor being at fault. Therefore an asset will not be assumed to have been tampered with until after a certain amount of time has passed.

Furthermore it is possible that students for lab purposes must disconnect the assets from the network. Therefore it will be assumed that if a room is booked for teaching purposes, any disconnects are non-malicious. This is deemed to not present a major vulnerability as a Chalmers employee must be present. It is highly unlikely that an attacker would try to tamper with the assets when someone is present who could easily identify them.

2.1.3 Attack Vectors

There are numerous different ways a potential attacker may attempt to steal hardware or otherwise interfere with the assets. The difficulty rating is based on how difficult it was deemed to perform on the hosts that were provided by the CSE department for testing. Likelihood represents the probability that an attacker would attempt the attack in the context of a university computer laboratory environment. Impact describes how severely the attack interferes with the systems ability to monitor the asset. These criteria are based on the teams observations and assumptions rather than any proper research.

Table 2.1: Different attack vectors

Action	Difficulty	Likelihood	Impact
Physically disconnecting the internet	Easy	High	High
Disconnecting the internet through software	Easy	High	High
Shutting down the asset	High	Medium	High
Pulling the power cable	Easy	High	High
Opening the asset chassi	Easy	High	Low
Stealing the entire asset	Hard	Low	High
Stealing hardware components	Medium	High	High
Replacing hardware	Hard	Medium	Medium
Man-in-the-middle attacks	Easy	Medium	High

As an attacker could combine the different attack vectors in many different combinations. The main mitigation strategy is therefore not to try and detect any series of combinations but rather to try and detect each individual event and, depending on the available context, determine if an alarm needs to be raised. It is also important to know that multiple assets could be attacked simultaneously.

2.2 System Features and Policies

The following design features and policies were incorporated into the system to address the threats identified in Section 2.1 while fulfilling the system requirements established in Section 1.4. Together, these mechanisms form the basis of the system's monitoring, detection, and recovery capabilities.

1. **Client-Server:** The system utilizes a central server to centralize the monitoring by collecting information from the clients running on monitored hosts. This approach centralises management and analysis allowing the client to consume a minimal amount of host resources.
2. **No User Interference:** As long as the boundaries established in section 2.1.2 are not violated, there will be no attempts to interfere with the asset when they are in use, such as for example automatic attempts to re-establish network connection when a host is in use.
3. **Heartbeat Monitoring:** The client sends a periodic lightweight signal to inform the system that it is active. This allows the system to monitor the connectivity of all monitored hosts.
   Mitigates: Internet disconnection, Shutdown, Power cable unplug. Stealing the entire asset
4. **Hardware Verification:** By checking the hardware components and comparing them to an established baseline during appropriate points, it can be discovered if the asset has been tampered with.
   Mitigates: Stealing and swapping hardware components.
5. **Message Encryption:** Communication between system entities is encrypted to protect transmitted data from unauthorized interception or modification during transit.
   Mitigates: Man-in-the-middle attacks.
6. **Intrusion Detection:** For assets equipped with a chassis intrusion switch, a signal can be generated whenever the chassis is opened. Since removal of hardware components may cause the host system to crash or become unstable, the client cannot reliably depend on software-based monitoring alone to detect such events. The intrusion switch therefore serves as the primary mechanism for detecting potential hardware tampering or theft.
   Mitigates: Opening the asset chassis, stealing or swapping hardware components.
7. **Automatic Asset Recovery:** The system can remotely instruct an asset to power on and subsequently verify recovery by waiting for a response from the client. This mechanism provides additional information regarding the state of the asset. If recovery fails, it indicates that the host is either disconnected from the network, disconnected from power, or otherwise non-functional. Success-

ful recovery additionally allows the system to resume monitoring and collect updated system information.
Mitigates: Pulling the power cable, disconnecting the internet, shutting down the asset, stealing the entire asset.

8. **Host Network Recovery:** The client can instruct the host to restart its network adapter when a network disconnection is detected. This allows the host to automatically attempt recovery from certain connectivity failures, thereby improving system stability and reducing false positives caused by temporary network issues.
   Mitigates: Disconnecting the internet through software.
9. **Room Aggregation:** Since multiple monitored hosts are typically located within the same laboratory, suspicious activity affecting one host may also affect nearby systems. The system therefore considers the state of all monitored hosts within a room when evaluating alarms, allowing correlated events across multiple hosts to be identified more effectively.
10. **Room Schedule Context:** The room schedule is incorporated as contextual information in the alarm evaluation process. By considering whether a laboratory is actively booked, the system can distinguish between anomalies likely caused by legitimate user activity and events that may indicate unauthorized tampering, particularly in situations where an asset is not communicating with the system for the reasons outlined in Section 2.1.2.

2.3 Technology Selection

The monitoring system relies on multiple technologies across both the client and server components. The selection of these technologies was guided by a set of vital requirements such as security, stability, performance and compatibility with the laboratory environment. The following subsections present the rationale behind the chosen operating system, network infrastructure, back-end framework, database solution, and client-side technologies.

2.3.1 Debian

The operating system chosen to host the server is the Linux distribution Debian. It was selected due to its stability, security-focused design, and widespread use in server environments [8]. Compared to more rapidly updated distributions, Debian provides a more predictable and controlled environment, which is beneficial for long-term running backend services. Full control over the operating system was considered essential to allow configuration of security settings, manage dependencies, and ensure consistent system behaviour.

2.3.2 Communication

In order to facilitate robust event driven communication between the server and client, the backend was implemented as a RESTful (Representational State Transfer) API. REST is a software architecture style that provides a standardized method

for networked applications to communicate. In this case, between the server and the client using standard HTTP methods [9].

To ensure secure communication, the system utilises HTTPS, which relies on Transport Layer Security (TLS) to encrypt data exchanged between the sender and the recipient, preventing eavesdropping [10]. This is particularly important as the system's endpoints utilizes HTTP request headers for authentication (see 3.3.1). Without HTTPS, these headers would be vulnerable to interception through network sniffing, potentially allowing unauthorized entities to spoof device communication.

A reverse proxy using NGINX was selected to enforce secure communication between the server and clients over HTTPS [11]. NGINX is responsible for handling incoming network traffic, terminating TLS connections, and forwarding the backend application.

2.3.3 Java and Spring Boot

Spring Boot is a framework that was chosen primarily because it provides an embedded server, making stand-alone deployment much easier. Furthermore, it provides native support for RESTful endpoints, through the @RestController annotation. Its embedded server utilizes a thread pool to manage concurrent HTTP requests. This ensures that simultaneous requests received are processed efficiently without requiring manual thread synchronization. It also seamlessly integrates with the ORM Spring Data JPA (as detailed in Section 2.3.4) [12]. Finally, selecting Java and Spring Boot led to smoother development because of the team's existing knowledge in API construction.

2.3.4 PostgreSQL and Supabase

A Database Management System (DBMS) organizes data into structured tables with predefined relationships, allowing for complex querying and strict data validation [13]. PostgreSQL was chosen as the DBMS due to its popularity, extensive documentation and the team's prior knowledge, which sped up the prototyping. Additionally, it complies with ACID (Atomicity, Consistency, Isolation, Durability) properties, which is desirable for a security monitoring application, as it guarantees data integrity and reliable database transactions [14].

During the development process the database was hosted by Supabase, a cloud-based infrastructure provider. Supabase allows for standard JDBC (Java Database Connectivity) connectivity between the backend logic and the hosted PostgreSQL database [15]. This architectural decision allowed remote access to get an overview of triggered alarms of the monitoring system during live testing to evaluate the system's effectiveness. In the case of a real world deployment, the database would not be hosted by Supabase.

To integrate the relational database with the backend logic, Object-Relational Map-

ping (ORM) was utilized. ORM is a programming technique that acts as a bridge between object-oriented programming languages and relational databases [16]. This pattern was implemented using the Spring Data JPA framework, which provides an ORM abstraction layer [16]. Standard usage of Spring Data JPA repository methods automatically generates parametrized queries, which is the primary defence against SQL injection vulnerabilities [17]. This is a crucial security requirement for the system.

2.3.5 Intel Active Management Technology

Intel Active Management Technology (AMT) is a solution for allowing a host to be remotely administered even when deactivated. Systems that are compatible with AMT are marked by the Intel VPRO Enterprise marking. AMT is an Intel specific system that runs separately from the OS, therefore requiring specialized hardware. The primary component is a specialized microprocessor [1] located on the motherboards chipset called a Management Engine (ME) that can be configured in multitude of ways. The most critical setting is called the control mode which has 2 options, Admin Control Mode (ACM) and Client Control Mode (CCM). ACM is the setting that allows unrestricted access to the features of the ME but requires a more rigorous provisioning process. CCM is far easier to set up but deliberately restricts what features can be used in order to safeguard the integrity of the host system.

The ME has full and unrestricted access to the host system, including BIOS/UEFI settings and the system hardware components, and therefore poses a security risk to the security of the host system. Due to this risk, the code on the ME is cryptographically signed to ensure its integrity. All interaction is therefore limited to sending and receiving messages from either the network or the CPU. The messages are constructed using the WS-Man protocol stack and are sent using HTTPS [2]. While communication details vary according to the provisioning state, digest Authentication is consistently required, ensuring that commands are only accepted from entities that can prove knowledge of the ME administrator password.

2.3.6 Windows Systems

The client utilizes 2 systems that are incorporated into the OS by default. The first is Windows Management Instrumentation (WMI). This is a system included in all versions of the Windows operating system since Windows 2000, a detailed explanation of WMI can be found at [18] . WMI functions similarly to a database in that it has a repository that can be queried for information by utilizing WMI Query Language, which uses SQL-like syntax. However, rather than acting as a traditional database, WMI serves as an abstraction layer. This layer allows users and programs to retrieve information about a host system, such as the CPU temp, the amount of

[1] The ME runs its own specialized micro-OS called MINIX-3.

[2] Communication with the ME is no longer possible over HTTP as that functionality has been discontinued by intel for security reasons

memory installed or when the last shutdown sequence occurred.

The second is what is known as Windows Services. A Windows Service is an executable that is being run by the Windows service manager. This allows a program to be run, receive or send commands independently of any user sessions under the Windows system account. Giving a program the highest level of privileges possible for the Windows OS. It also ensures that the program is automatically started upon the Windows boot.

2.3.7 Dell Command Monitor

Dell Command Monitor (DCM) is a management tool for Dell manufactured computers that allows a user to modify the BIOS/UEFI [19]. This allowed for better and more efficient configuration of firmware settings than restarting the host and configuring the BIOS manually. Important to note that it only works on hosts that are manufactured by Dell as it requires Dell specific firmware in the host BIOS.

2.3.8 Go

To ensure that the project requirements were fulfilled, careful consideration was put into deciding which programming language was deemed most suitable for the clients development. It was chosen due to its simplicity, minimal resource requirements, built in support for concurrency and the fact that Go compiles directly to machine code. Furthermore, Go is supported with a multitude of different libraries and packages that could be used to communicate with the client OS, such as WMI or Remote Procedure Call (RPC). Go was therefore deemed to be an excellent fit for the needs of the project. Other languages that were considered included Python and C#, primarily because they supported libraries that could be used to interface with the WMI. However, these languages all had specific drawbacks that were deemed to make them an inferior choice towards using Go.

2.3.9 Vue.js

Vue.js is a JavaScript frontend web framework used for creating interfaces and Single-Page Applications. The Document Object Model (DOM) is an application programming interface, representing an HTML document as a tree of objects, enabling languages to dynamically interact, modify and update the document's content. Vue is reactive, meaning that it automatically updates the DOM by tracking the underlying state changes. It also utilizes a component based architecture, facilitating the creation of reusable and testable UI components [20]. These technical capabilities, combined with the teams prior familiarity with the framework, were the primary reasons Vue was chosen for the creation of the GUI. Axios, a promise-based HTTP client, was the chosen approach for retrieving data from the created API endpoints on the server [21].

2.4 Evaluation Strategy

To ensure the delivery of an efficient and reliable hardware monitoring system, a comprehensive testing strategy was implemented in accordance with the requirements outlined in Section 1.4. Since the system needs to operate continuously without interrupting or disturbing standard lab activities, the testing focused on both active threat simulations and passive long term stability. Therefore, the testing of the system as a whole was divided into three distinct phases: Active testing, simulating real physical threats; passive testing, evaluating the performance of the system; GUI testing, validating the created dashboard. These tests were performed with four lab computers in room ED-4209, where a few lab sessions were held each week.

2.4.1 Passive Testing

This phase was designed to test the stability of the system and establish a performance baseline over an extended period of time. This was carried out by observing the system during normal use of laboratory computers, in order to evaluate network stability and assess whether false alarms were generated too frequently. This was performed for one week continuously on the four computers in room ED-4209, with lab sessions held on one of the days for real data usage. During this period the following metrics were evaluated:

1. **Network stability:** Evaluate network reliability and frequency of client side disconnects (Validates Requirement 5).
2. **Heartbeat consistency:** Measure the number of heartbeats the client attempted to send with the amount successfully logged by the server (Validates Requirement 5).
3. **False Positive Rate:** Monitor alarm frequency under normal conditions in order to ensure that the system does not trigger unnecessary alarms or too many false positives (Validates Requirement 11).
4. **Resource Utilization:** Tracks CPU and RAM usage on the client to ensure that it does not interfere with students normal usage (Validates Requirement 3).
5. **Data Integrity:** Verifies the consistency of the database and secure transmission of telemetry over time (Validates Requirement 8).

2.4.2 Active Testing

This phase was designed to test the systems core security logic and event driven architecture. This was performed by simulating different physical hardware threats in order to evaluate if the system could properly detect events, as well as distinguish actual alarms and any false positives that might arise. The following simulations were performed on the computers in room ED-4209:

1. **Intrusion Detection:** The host chassis is opened to verify if a chassis intrusion alarm is triggered and transmitted within 30 seconds (Validates Require-

ment 1).
2. **Network Reconnection:** The host is disconnected from the internet to see if the client forces it back on (as detailed in Section 3.2.3) (Validates Requirement 9).
3. **Standard Reboot Cycle:** The host is turned off and restarted to see that it triggers the correct events and verifies that upon startup, it sends a valid startup message (see Section 3.2.2) (Validates Requirement 6).
4. **Remote Wake Up:** The host is turned off to see that the server tries to turn it back on using the ME (implemented in Section 3.4.2). If not successful, it triggers an alarm stating the computer is unreachable (Validates Requirement 7).
5. **Administrator Override:** The host is turned off by an administrator login, resulting in no alarms, the device being registered as inactive, and the server properly ignoring future alarms (Validates Requirement 6).
6. **Hardware Mismatch Detection:** When the host is turned off the RAM is taken out, and the ME successfully tries to turn the computer on. If no startup message or hardware mismatch is received (see Section 3.4.4 for validation logic), the server triggers an alarm (Validates Requirement 1 & 6).
7. **Client Security:** No non-admin user can access the client files, including stopping the background service (described in Section 3.2.1) or reading the created files (Validates Requirement 4).
8. **Heartbeat Timeout:** A host is purposely disconnected from the network and the server ensures it triggers an alarm after the 5-minute threshold has elapsed (see Section 3.4.2 for timeout logic) (Validates Requirement 5).
9. **Mass Disconnect:** Multiple hosts are simultaneously taken offline to test the servers mass disconnection threshold, verifying that a mass disconnect alarm is triggered when more than 25% of the rooms computers are not reachable (see Section 3.4.1 for a justification of this threshold) (Validates Requirement 10).
10. **Client Deployment:** The install script is installed across multiple hosts to validate that the script successfully registers the device, saves the important environment variables to the correct spot, configures the background service and initiates telemetry without any manual configuration.

2.4.3 GUI Testing

To validate the usefulness of the GUI, a usability evaluation was conducted with one of the members responsible for the CSE division's computers. This test focused on the intuitive navigation and core functions of the created dashboard. The participant was asked to complete a series of predefined scenarios, using a "think-aloud" method, where their feedback was documented. The scenarios consisted of the following:

1. **General Navigation:** The user was asked to navigate the dashboard to evaluate the intuitiveness of the layout and the real-time device overview (Validates Requirement 12).

2. **Alarm Observation:** A chassis was opened to simulate a chassis intrusion, evaluating the GUIs ability to clearly alert the user (Validates Requirement 13).
3. **Alarm Management:** The user was instructed to navigate to the alarm page, locate the newly triggered alarm and mark it as "handled" (Validates Requirement 13).
4. **Report Creation:** The user was asked to draft a report linked to the simulated alarm (Validates Requirement 14).
5. **Report Modification:** The user was instructed to navigate to the dedicated report overview page, locate the previously created one, and edit its contents (Validates Requirement 14).

3

Implementation

The implemented monitoring system follows a centralised client–server architecture derived from the methodological design. Distributed monitoring clients are deployed on laboratory computers and continuously communicate with a central evaluation server. The purpose of this architecture is to operationalize the detection strategy while minimizing impact on normal laboratory usage.

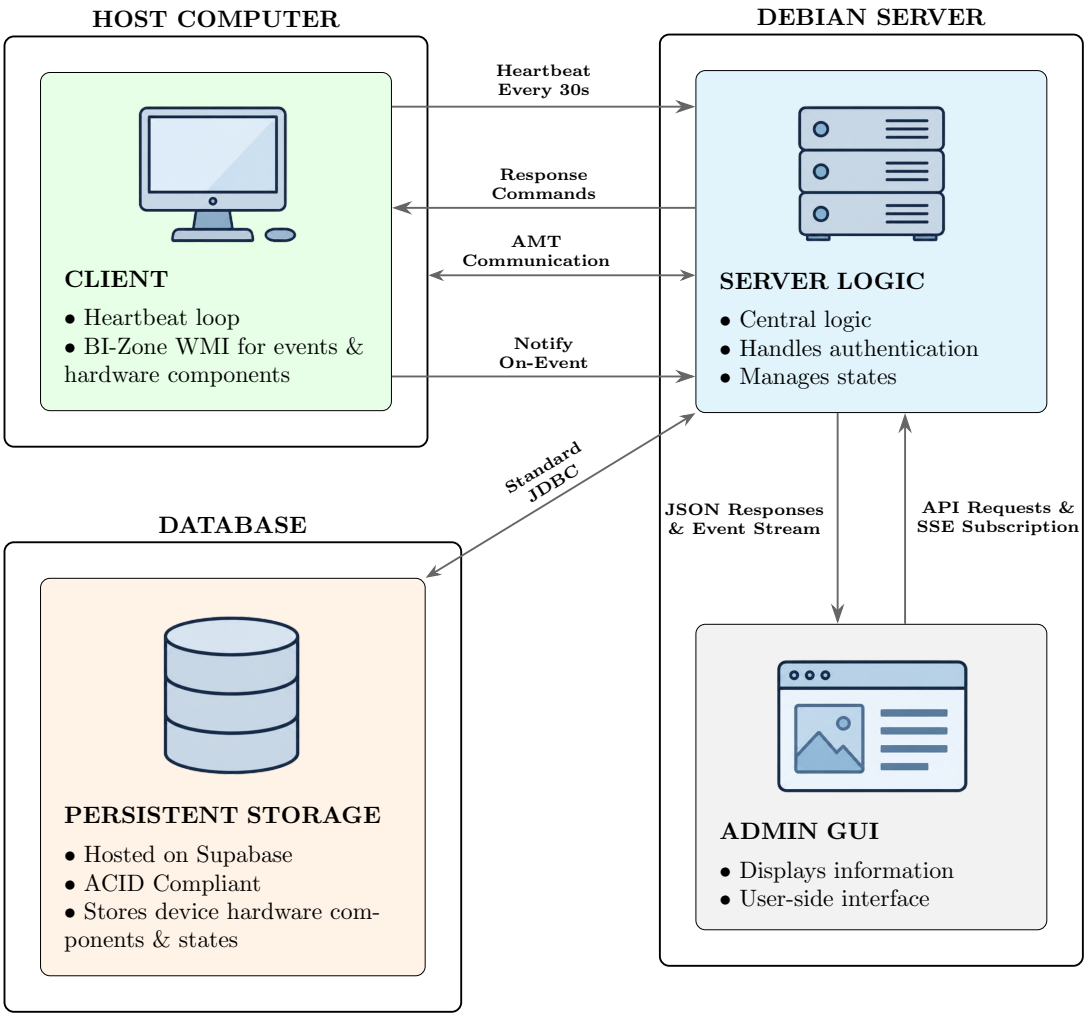

Figure 3.1: Overview of proposed system architecture. (Icons within each container created with ChatGPT, 2026)

3.1 System Architecture Overview

The system consists of four primary components, as visualized in figure 3.1: the monitoring client, the central server, the database, and the graphical user interface. Each component fulfils a distinct role within the overall architecture. The client operates as a Windows service responsible for collecting hardware and system events and forwarding relevant information to the server . It performs only local event detection and does not execute complex evaluation logic, thereby preserving performance and preventing tampering with core detection mechanisms. The central server functions as the primary processing unit. Through a RESTful API, it handles client registration, telemetry ingestion, alarm evaluation, remote management operations via AMT, and persistent data storage. The database maintains device registrations, hardware baselines, room associations, scheduling data, alarm records, and administrative reports, enabling historical traceability and state comparison. The GUI provides administrators with a centralised real-time overview of system status and active alarms, allowing incident handling and report generation.

Communication between components follows a request–response model over HTTPS using authenticated API calls. After registration and credential assignment, each client begins transmitting periodic heartbeat messages and event reports to the server. Incoming data is evaluated against the implemented alarm logic, contextualized when necessary, and persisted in the database. When alarm conditions are confirmed, notifications are forwarded to the GUI for administrative action. This lifecycle ensures separation of concerns between data acquisition, evaluation, storage, and visualization while maintaining architectural simplicity and reducing attack surface by avoiding exposed listening services on the client side.

3.2 Client Implementation

The client is a host-based application responsible for collecting system and hardware information from the monitored host and transmitting it to the server. In addition to monitoring system events, the client manages communication with the server and performs limited recovery-related actions depending on the current host state.

The client is intentionally designed to operate independently of the host's ME. This separation was implemented to minimize exposure of the ME password to the host operating system by removing the need to store it locally. Reducing the potential of the host becoming compromised while also simplifying deployment and password management.

3.2.1 Client Architecture

The client is implemented using an event-based concurrent architecture, illustrated in Figure 3.2. The primary execution context is implemented as a Windows Service, allowing the client to start automatically during system boot and continue operating independently of any active user session. Running the client as a service also

allows it to respond to system shutdown events in a controlled manner. However, this does not provide protection against unexpected power loss caused by the host being disconnected from its power source.

The primary service thread spawns a number of child monitoring threads responsible for observing different categories of system events, such as heartbeat timing and network state changes. Event monitoring is primarily implemented using the WMI-Bi-Zone library [23], which allows the client to subscribe to changes within the WMI repository. This event-driven approach reduces the need for continuous polling and allows the monitoring system to react to changes asynchronously. Communication between monitoring threads and the central Client Monitor component shown in Figure 3.2 is implemented using Go channels. When a monitored event occurs, a temporary worker goroutine is spawned to handle communication with the server. This prevents network communication from blocking the monitoring components and reduces the risk of missed events.

The client stores configuration and logging data under the Windows `ProgramData` directory. The `.env` file is used to persist important configuration information, primarily the API key required for authentication with the server and the servers IP-address. Persistent storage is necessary to ensure that the client retains its configuration across system reboots and shutdowns. Access permissions for these files are restricted to administrators in order to reduce the risk of unauthorized access or modification.

A dedicated logging file is also maintained by the client. This was primarily introduced for debugging and development purposes, as Windows services are not attached to a normal terminal session and therefore cannot easily provide runtime console output. While logging activity during normal operation is limited, the logging system was retained in the final prototype to support troubleshooting and system diagnostics.

3.2.2 Power State Monitoring

Monitoring of host power-state events is handled by the primary service component of the client. Since the client operates as a Windows service, it is able to receive power-related control messages from the Windows Service Control Manager (SCM), including the `PreShutdown` and `Shutdown` events. Upon receiving a shutdown-related event, the client requests additional time from the SCM before system shutdown proceeds. This allows the client to notify the server that the host is shutting down before the operating system becomes unavailable.
Because the client is configured as a Windows service, it automatically starts during the Windows boot process. During startup, the client sends a notification to the server indicating that the host has become active again and transmits an updated list of detected hardware components. This allows the server to synchronize the host state after reboot and verify whether any hardware changes occurred while the system was offline.

The power states hibernation and sleep are not monitored. It was attempted during development to implement such monitoring. However, due to how Windows handles such power state transitions it became difficult to accurately detect them. It was therefore decided that the systems automatic recovery functionality discussed in 3.4 was enough to protect against such issues. This does mean that if an unauthorized user is able to put a host into sleep, the system is unable to inform the administrators.

3.2.3 Network Monitoring and Recovery

The client continuously monitors the network connectivity state of the host. If the client detects that the host has disconnected from the network, it can attempt to restore connectivity by restarting the network adapter. This action is conditional on the current usage state of the host. If the host is not actively in use, the client automatically attempts to restart the network adapter. If the host is in use, the client avoids performing automatic network recovery actions in order to minimize potential disruption to the user. However, this restriction is overridden if the client detects events such as chassis intrusion or shutdown events.

When the client detects that the host has reconnected to the network, it transmits an updated list of detected hardware components to the server. This allows the server to synchronize the current hardware state of the host after a period of network disconnection. The `Client Manager` additionally filters duplicate network events before processing them. This was necessary because Windows may generate multiple identical network status events for a single connectivity change, which would otherwise result in repeated recovery attempts and redundant event transmissions.

3.2.4 Session Monitoring

In order to avoid client interference with normal host operations, it is required to determine if a host is being actively used. To this end, the client checks every 5 seconds if a user is currently logged on, and if said user is an administrator. Normally, this would be done by checking the session token that the client is running on. However, as the client is configured as a Windows service, the session that is running the client is the system account. As a result, the client cannot directly determine user activity from its own execution context. To address this limitation, the Use State thread in figure 3.2 must extract every session token currently active. It then checks if the `isExplorer` field is true for each individual token. Since the Windows Explorer process is associated with user sessions, this allows the client to determine whether a user is currently logged on to the host. `Use State` then checks if said user account belongs to the admin group to determine whether it is an administrator.

3.2.5 Intrusion Detection

Implementing chassis intrusion detection proved to be more difficult than initially expected. The intrusion switch is supposed to update the WMI repository when the

chassis is opened. However, due to what is suspected to be firmware issues on the test hosts, this did not occur. Therefore, DCM is installed on the host system to monitor the motherboard's chassis intrusion switch and generate standard Windows Event Log events upon any state change. This allows the client to listen for any new events with Windows Event Log numbers that corresponds to `ChassisIntrusion` or `ChassisIntrusionNormal`. Upon detecting such an event, the client immediately transmits an alert to the server. This means that the Intrusion detection feature only works on Dell manufactured hosts, a core limitation of the system.

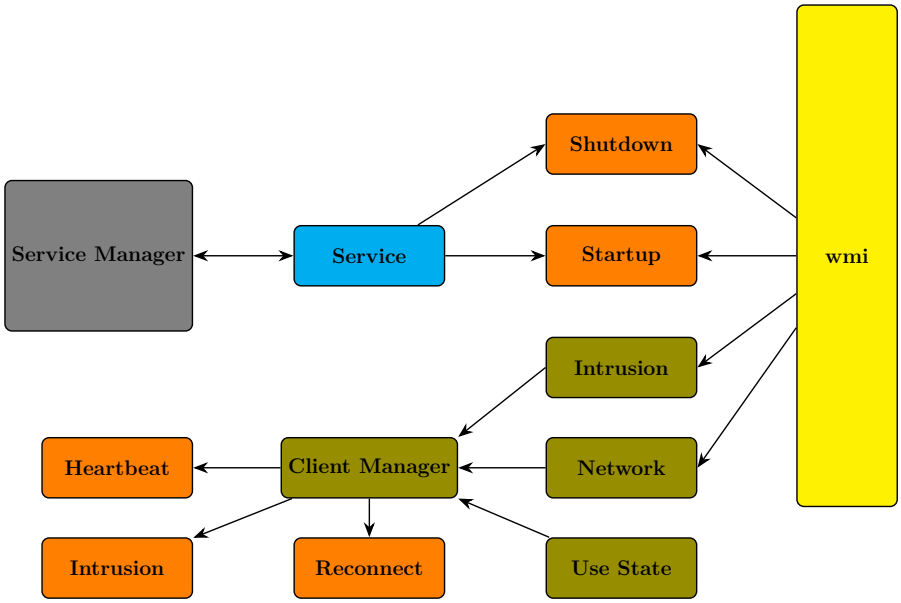

Figure 3.2: Client Architecture Diagram, Green nodes are monitoring threads, Orange nodes are temporary messaging threads

3.3 Server Implementation

This section focuses on the internal design of the central server. The server is responsible for telemetry, validating and processing incoming data, executing alarm evaluation logic, and coordinating remote management functionality. The following subsections describe how the server implements communication through its RESTful API, enforces authentication, integrates AMT, and structures the alarm handling logic. The room scheduling information was acquired from TimeEdit through an ICS calendar link, which was periodically parsed by the server and stored in the database for contextual alarm evaluation.

3.3.1 API Endpoints

The RESTful API is structured to handle the entire life of a monitored device, from initial registration to active and passive threat detection. It consists of POST request endpoints utilized by the clients. All endpoints utilize HTTP Request Headers for authentication through API-Keys before processing any data. As mentioned previously, the underlying Spring Boot architecture provides the server with native

concurrency. Therefore, when receiving multiple requests at once, the server allocates a separate execution thread from its thread pool for each incoming HTTP request. This ensures that the server can manage multiple active devices without blocking the main execution thread.

During the installation script described in Section 3.7.1 the client utilizes the `/register` endpoint, which introduces the device and its specific hardware components into the system. This allows the system to establish a "normal" hardware baseline to compare future telemetry against. During this registration, the API Key is created and sent back as a response, but is also hashed and mapped to the correct device in the database. There is also a corresponding `/unregister` endpoint for removing a device through the uninstall script. Furthermore, the Data Transfer Object (DTO) design pattern was utilized to decouple the database entities from the network layer, mapping JSON payloads. This ensures that the API only accepts predefined data structures, preventing over posting vulnerabilities and maintaining strict data integrity.

Furthermore, there are endpoints for heartbeats and reports where the client sends its continuous heartbeat and hardware checks. The `/heartbeat` endpoint responds with a command, which instructs the client to act in certain ways, such as sending a report of its hardware components. This allows the server to issue commands to the client without deploying a RESTful API endpoint on the client side. Since the client needs to be as lightweight as possible, hosting a local listening server on the client was deemed unnecessary for this specific use case. Lastly, there are endpoints for all different client events, which trigger different alarm logic, more on this in Section 3.4.

Finally, a few endpoints were set up for the created GUI to utilize for communicating with the server. These endpoints consist of GET endpoints for devices, devices hardware specifications and alarms for retrieving information while loading the dashboard. Additionally, POST and PUT endpoints are used when an administrator interacts with the dashboard to add a room, submit a report or mark alarms as handled. In order to ensure that the dashboard remains reactive without constantly refreshing the page, a Server-Sent Events architecture was implemented (more on SSE in Section 3.6.1). The GUI establishes a connection to the server events through the `/stream` endpoint, allowing for near instant updates on certain state changes.

3.3.2 Recovering the Host with AMT

To implement a host recovery mechanism, the server communicates directly with the ME of the host system. When initiating a restart, the server retrieves the ME password and the IP address of the target and invokes a Go script through the GO Command Line Interface (CLI). Go was selected for this interaction because it provides a suitable library for constructing messages that conform to the WS-Man standard, something not natively supported in Java. The IP and password are passed as arguments to the CLI, which then attempts to execute the remote

restart. If the restart command succeeds, the server enters a short waiting period of 90 seconds to allow the host to complete its boot sequence. This additional delay accounts for scenarios where the host can boot into BIOS but not into Windows.

3.4 Alarm Handling Implementation

The alarm handling functionality is implemented as a centralised server-side service responsible for evaluating incoming telemetry and determining whether alarm conditions should be generated. This service operates continuously and processes both periodic heartbeat data and event-based report from monitored clients. The implementations follows a layered evaluation process in which anomalies are validated, contextualized, and, when applicable, subjected to automated recovery attempts before escalation, as seen in figure 3.3. This ensures consistent alarm handling while reducing unnecessary alarm generation.

3.4.1 Heartbeat Monitoring and Disconnection Evaluation

The primary availability monitoring mechanism is based on periodic heartbeat messages transmitted by each client. Each heartbeat is sent every 30 seconds. This interval was selected as a trade-off between detection latency and network overhead. A shorter interval would increase network traffic and processing load without significantly improving practical detection time, while a longer interval could delay identification of genuine disconnections. The 30-second interval therefore provides near real-time monitoring while maintaining system efficiency. Upon receiving a heartbeat, the server updates the device's last activity timestamp in the database. A scheduled evaluation task runs at fixed intervals (30 seconds) and compares the current time against each device's last recorded heartbeat. If it exceeds the 5 minute time threshold, an alarm is sent and the device is marked as disconnected in the GUI. The five-minute threshold was selected to provide sufficient time for the device to re-establish connectivity and transmit a new heartbeat before escalating the event.

Disconnected devices are grouped according to their associated room. If more than 25 percent of the devices within a room are classified as disconnected during the same evaluation cycle, the system generates a mass disconnection alarm. During testing, 25 percent provided a reasonable balance between responsiveness and false-positive mitigation. In a full-scale deployment, this parameter should be calibrated according to room size and empirical operating data. This grouping logic enables detection of broader infrastructure issues while preventing isolated connectivity interruptions from triggering large-scale alerts.

3.4.2 Automated Recovery Using AMT

Before escalating the disconnection into a final alarm state, the system attempts automated recovery using AMT. When a device first misses a heartbeat, it is marked locally in the server as disconnected. After 2 minutes have passed since the last heartbeat a recovery attempt is initiated. The recovery attempt is tracked using

a timeout mechanism. If the device successfully reconnects within a five-minute window, the event is treated as a temporary failure and no alarm is created. If the timeout expires without successful reconnection, an alarm indicating failed remote recovery is generated. This staged escalation is implemented to reduce false positives caused by temporary network instability or temporary power interruptions.

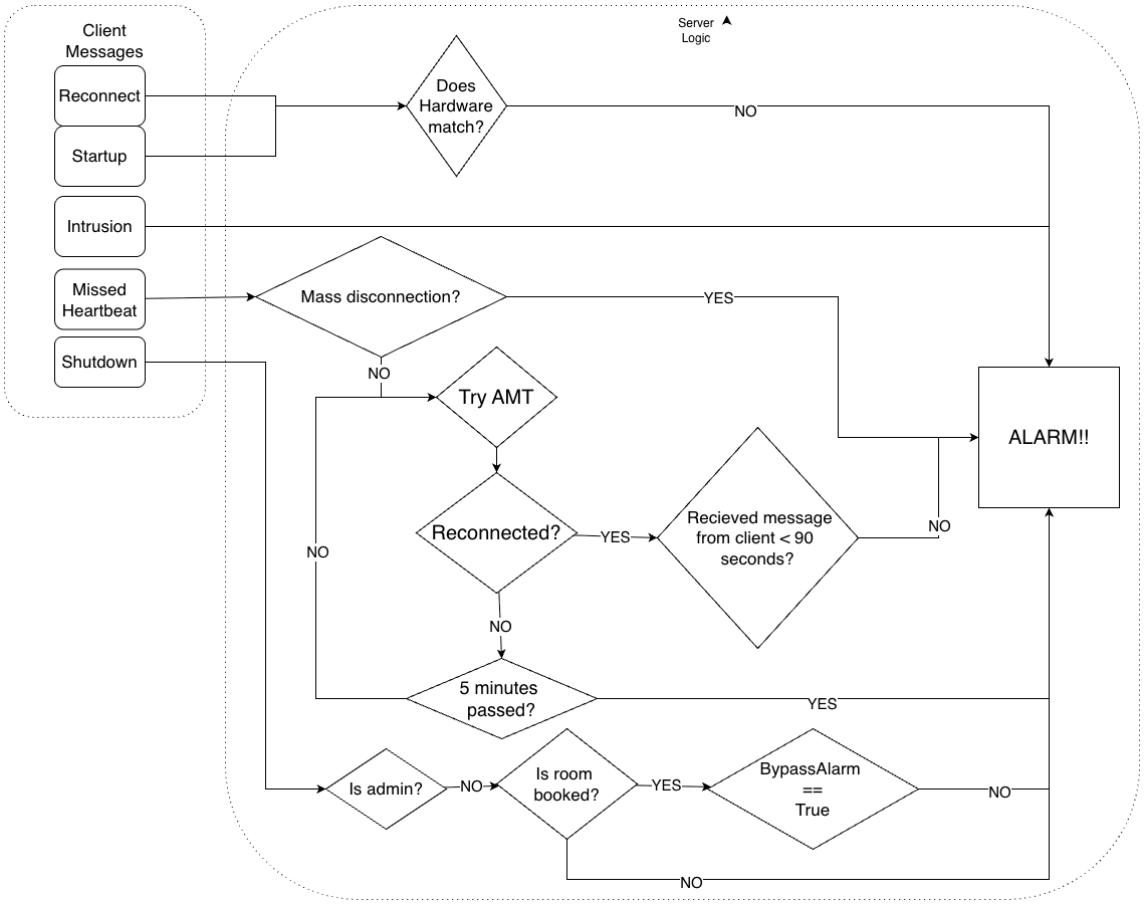

Figure 3.3: Alarm handling logic.

3.4.3 Shutdown Event Handling

In addition to heartbeat monitoring, the server processes shutdown events reported by the client. When a shutdown event is received, the device's last-seen timestamp is updated, and the shutdown type is evaluated. Shutdown events are categorized as shutdown, restart and unknown. Administrative shutdowns are logged without generating alarms, as they are considered authorized actions. A normal shutdown is handled as a suspicious activity, since a normal user should not be able to shutdown a computer in the laboratory rooms, and therefore it triggers an alarm. Unexpected shutdown types are treated conservatively and result in alarm creation. This classification logic ensures that legitimate maintenance does not trigger alarms while abnormal shutdown behaviour is properly investigated.

3.4.4 Hardware Validation and Mismatch Detection

Hardware integrity is validated by comparing incoming hardware reports with previously registered baseline configurations stored in the database. These comparisons include verification of hardware identifiers and storage drives. If discrepancies are detected, such as missing storage drivers or altered hardware components, a hardware mismatch alarm is generated. Hardware validation is triggered during startup and reconnect event, to ensure that core hardware components remain intact over time.

3.4.5 Chassis Intrusion and Contextual Alarm Suppression

If a client reports a chassis intrusion event, the alarm service immediately generates a high-severity alarm without staged recovery. Since internal hardware access is directly associated with the threat model of component theft, such events are treated as strong indicators of potential tampering. The system does not attempt automated recovery in this case, as restarting the device would not mitigate or clarify the event. Instead, the alarm is immediately persisted and forwarded to the GUI to allow rapid administrative response. This design decision reflects a risk-based prioritization strategy, where events directly related to physical integrity are escalated without delay.

Before persisting certain alarms, the system performs contextual validation using room scheduling information stored in the database. If a room is currently booked, specific alarm types may be suppressed to prevent false positives caused by legitimate user activity. This contextual filtering step is executed before the final alarm creation and ensures that alarms remain relevant and actionable.

3.4.6 Alarm Lifecycle

When an alarm condition is confirmed, an alarm entity is created and stored in the database. An alarm notification is then broadcast via the SSE pipeline (see Section 3.6.1), triggering an update in the GUI to display the new event. This allows administrators to monitor events in real time. Each alarm follows a consistent lifecycle, see figure 3.3:

1. Detection of irregular condition
2. Contextual evaluation and optional recovery attempt
3. Alarm creation and database persistence
4. Alarm notification is picked up by the GUI and broadcast as a new alarm

This structural implementation ensures traceability, consistency, and integration with the overall monitoring architecture.

3.5 Database Implementation

The database serves as the persistent storage layer for the monitoring system. It is responsible for storing registered device profiles, hardware specifications, authentication tokens, system alarms and evaluation reports.

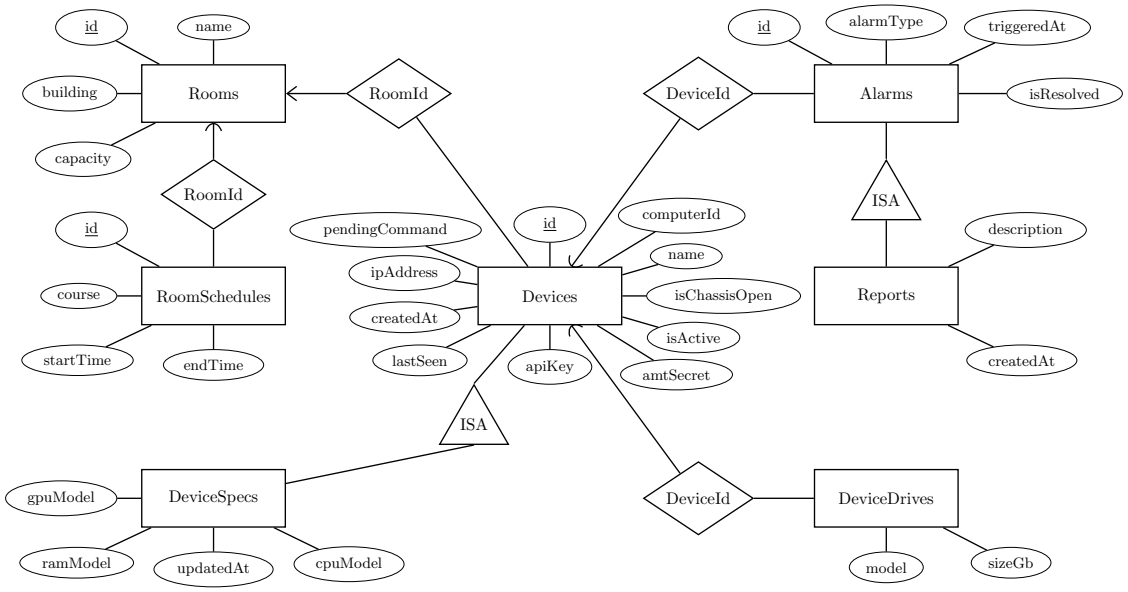

Figure 3.4: ER-diagram of the database structure.

The ER-diagram as seen in Figure 3.4 demonstrates the foundational structure of this systems data model through an entity-relationship diagram. The central entity is the `Device`, which holds the essential connections and telemetry endpoints for each monitored machine. Each device also uses `DeviceSpec` and `DeviceDrive` for storing components, such as GPU or Drives, allowing the system to verify the presence of physical hardware. The primary log for detected physical security or connection events is the `Alarm` entity, while the `Report` entity allows administrators to add description and resolutions to those triggered alarms. Finally, `Room` and `RoomSchedule` are utilised to store the locations and active schedules for the monitored computer labs.

The interaction between these entities is illustrated through their relationships. To optimize structure, `DeviceSpec` is modeled as a direct extension of `Device` (shown by the ISA relation), representing a strict one-to-one subtyping where the specs share the parents primary key. `DeviceDrive` has a Many-to-One relation with `Devices`, enabling a single computer to register a different number of drives. Once a potential theft is detected, the incoming alert populates a new entry in the `Alarm` entity. This entry is now linked to the specific `Device` through the `DeviceId` relation, logging the exact moment and type of security breach. The `Report` also utilises an ISA relation to act as a One-to-One extension of the `Alarm`, providing an extended report functionality for specific alarm entries. Finally, both `Device` and `RoomSchedule`

have a Many-to-One relationship with the `Room` entity through `RoomId`, linking it to specific rooms.

3.6 GUI Implementation

The administrative dashboard was implemented utilising Vue.js and axios to retrieve the displayed data through its dedicated api.ts file. The frontend was structured using a modular component based architecture to facilitate reusability and testing. This was done by implementing one high level routing component for each page, which were populated by smaller, isolated and reusable components. The created pages were: `DeviceView` which acts as the main page, giving an overview of the connected devices and their status; `AlarmView` which gives an overview of the occurred alarms with functionality for handling alarms and creating reports; and lastly, `ReportView` for an overview over the created reports. Some of these isolated components consist of telemetry metric cards for statistics, different data rows for an overview and modal windows for further information, which all accept data through predefined component properties (props).

3.6.1 Server-Sent Events

On the server side, a Server-Sent Events (SSE) architecture was implemented and managed by the SseService Class. As mentioned above, when the dashboard is first loaded, the frontend subscribes to the `/stream` endpoint, leading the server to generate a Spring SseEmitter object configured with an infinite timeout. These active emitters are stored in a CopyOnWriteArrayList allowing multiple administrators to be connected at once. Whenever the backend server detects a certain event, such as a triggered alarm, added or deleted device, the service loops through the list of emitters and broadcasts the event type along with the corresponding object.

3.6.2 Vue Composable

In order for the dashboard to be accurate it needed real-time updates. This was implemented by creating a centralised Vue Composable, useMonitoring.ts. This acts as the frontend single source of truth, by keeping track of the different states that might occur. It initializes asynchronous data fetching through the Axios HTTP client upon first load, and establishes the persistent SSE connection to the server. To make sure incoming telemetry through the SSE pipeline instantly updates the global state, the dashboard uses Vue's reactive references, ref and computed. This automatically triggers re-renders of the affected DOM elements.

3.6.3 Data Aggregation and UX Patterns

In order to ensure an optimized User Experience (UX), two frontend patterns were implemented within the GUI. First, to prevent the alarm overview from being packed with the same triggered alarms, data aggregation logic was implemented into the useMonitoring composable. This dynamically groups mass disconnection alarms

within the same room and at the same time together, as seen in Figures 3.5 and 3.6.

Figure 3.5: Two alarm rows displaying the data aggregation logic on a mass disconnection alarm.

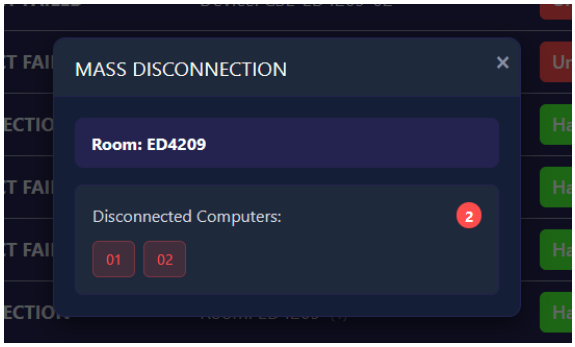

Figure 3.6: Displaying the modal window of a mass disconnection alarm.

Second, to maintain a responsive interface, the design pattern Optimistic UI was implemented. This means that upon an administrator marking an alarm as handled or unhandled, the frontend instantly updates the DOM, assuming a successful server response. If the API request were to fail, a rollback function would catch the error and alert the user, reverting the UI to its previous state.

3.7 Deployment and Infrastructure

The server was deployed on Debian and configured as a system service, enabling automatic restarts in the event of application failures and crashes. However, in cases where the operating system itself becomes unavailable, such as power loss, manual intervention is required to restart the hardware and restore system operation.

The server was hosted behind an NGINX reverse proxy, which routed incoming requests to the server. HTTPS support and TLS termination were handled through the reverse proxy configuration described in section 2.3.2.

3.7.1 Client Installation and Configuration

To simplify deployment of the client across multiple hosts, an installation script was developed using Windows PowerShell. The script was primarily created to automate the otherwise complex and repetitive configuration process required during development, where frequent updates often required reinstallation of the Windows service. The installation process automatically creates the required directory structure for

configuration and log files, configures the client to run as a Windows service, and establishes an initial connection with the server. The initial connection is, as mentioned in Section 3.3.1, done automatically through the `/register` endpoint. This registration requires a Master Secret, a password for IT personnel that is not stored anywhere and used as an initial API Key, in order to ensure that no unauthorized entities are entered into the database. Upon successful registration, the server responds with an API Key used for future authentication.

In addition to the basic installation process, the script disables fast startup to ensure that system shutdown events can be detected reliably. Sensitive parameters, such as the master secret, ME password, and registration flags, are passed as runtime arguments to the clients executable rather than being stored persistently on the host system. This allows the client to establish safe communication with the server and retrieve an API key. This reduces the risk of exposing sensitive information, as the parameters are discarded after installation has completed. The script also configures file permissions such that only administrators are able to access the generated configuration and log files.

The installation script is responsible for the configuration of the host's ME and installation of DCM. If the host is AMT compatible, the user is given the option to enable automatic recovery functionality. If enabled, the script downloads the *RPC GO*, utility available at [24], and uses it to configure the ME by enabling CCM and setting the administrator password. The utility is stored temporarily during installation and removed once the installation process has completed. If the ME has already been configured prior to installation, the current ME password must still be provided by the user, as the password cannot be retrieved programmatically after provisioning. DCM is installed under `Program Files/Dell/DCM`. The installation process assumes that both the client executable and the DCM installation package are located in the same directory as the installation script.

An accompanying uninstall script was implemented to support clean removal of the client. The uninstall process removes the Windows service, deletes associated configuration and log files, and unregisters the host from the server database. However, the uninstall script does not remove DCM or reset the ME configuration. This decision was made to simplify redeployment during development and reduce the time required to install updated versions of the client.

4
Results

The proposed hardware monitoring system was evaluated according to the evaluation strategy described in section 2.4. The system successfully demonstrated high stability and accuracy, fulfilling the core requirements defined in the System Requirements (Section 1.4). A detailed view of requirements mapped to corresponding tests and observed results can be seen in Table 4.1.

4.1 Passive Testing Results

During the one week testing period in room ED-4209, the system established a highly stable performance baseline. In terms of communication reliability, the data demonstrated a 100% heartbeat transmission consistency rate, meaning every heartbeat the client attempted to transmit was received by the server. Furthermore, the database also remained consistent throughout the testing period, and the utilization of HTTPS ensured that all transmitted telemetry remained secure against interception.

A critical success factor for the system was its ability to operate without disturbing standard laboratory activities. Over the testing period the system generated zero false positive alarms, proving that standard student usage and minor network errors do not overflow the alarm log. Finally, the client proved to be very lightweight, ensuring the host performance is not impacted. As illustrated in Figure A.2, the RAM usage increased by a negligible 0.07 percentage points (from 14.24% to 14.31%). While the client introduced slight CPU usage, it remained at 0.68% (an increase of 0.29 percentage points). This confirmed that the hosts performance remains unaffected.

4.2 Active Testing Results

Each of the ten active hardware threat scenarios described in Section 2.4.2 was manually simulated in order to validate the event driven design. Firstly, physically opening the client chassis successfully generated Windows Event Logs, which the client intercepted and transmitted to the server, and alarmed within 10-25 seconds. In order to validate the hardware comparison logic without altering the university equipment, the registered hard drive was removed from the database. This led to the server successfully triggering a true positive alarm upon startup as well as during the twice a day hardware checks. The system also correctly triggered timeout

alarms after five minutes of no heartbeats, and escalating to mass disconnection upon exceeding the offline threshold for a room.

One important factor was for administrators to be able to turn off the device for maintenance without triggering any alarms. The system successfully did this by distinguishing between different shut off types and user privileges. Upon the host being turned off by regular users, the system was able to use the ME to remotely execute startup attempts. It correctly triggered an alarm after the 90-seconds threshold of no startup message, a scenario which was simulated by removing the RAM, leading the host to boot into BIOS. It also proved to be able to force the hosts network back on when being manually turned off in windows.

Another crucial factor was for the background service and its generated files to be easily set up without manual configuration, while being inaccessible for non administrator users. The install script, as seen in Figure B.1, proved to be effective without any manual configuration, and successfully locking down the background service and created files for any non administrator users.

4.3 GUI Testing Results

The five GUI testing scenarios described in Section 2.4.3 were executed with the participant during a "think-aloud" test session to validate the intuitiveness and core functions of the created dashboard. The feedback provided indicated that the most valuable feature for IT personnel is maintaining an overview of connected devices and their online/offline status. As seen in Figure 4.1, the participant noted that the dashboard provided a clear visual representation of this data, as well as it felt intuitive to navigate through the three tabs.

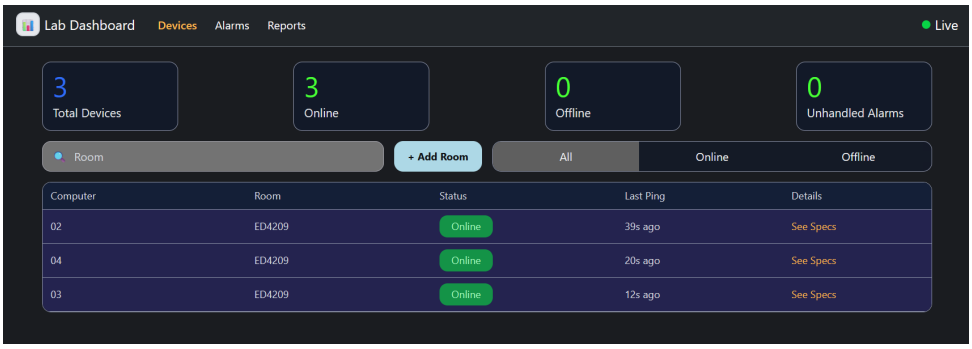

Figure 4.1: The main page with a device overview.

Another important feature was the ability to clearly see new alarms. This was simulated to the participant by opening a chassis, triggering a chassis intrusion alarm. The dashboard was observed to clearly display the new threat, specifically the shift of the "Unhandled" card turning from a green zero to a red one, as illustrated in Figure 4.2. The workflow for marking an alarm as "Handled" was also deemed straightforward. The participant also found it intuitive to create a new report directly from the alarm page (see Figure 4.2), as well as navigating to the dedicated report page to edit its contents.

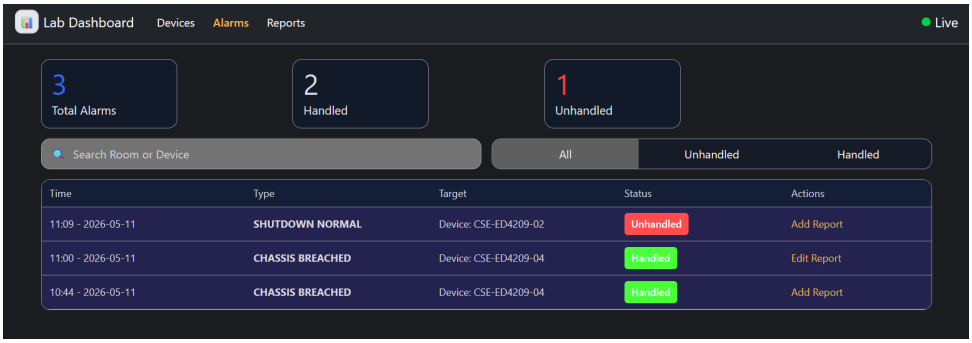

Figure 4.2: The alarm page with an alarm overview.

4.4 Requirement Validation and Testing Results

In the following Table 4.1, a detailed overview of each requirement and how it was evaluated can be seen. It also shows the observed results and whether each requirement was fulfilled.

Table 4.1: Summary of Requirements Validation and Testing Results

Requirement	Test Scenario	Observed Result	Fulfilled?
Req 1: Detect physical tampering	Active Test 1 & 6	System generated Windows Event Logs and transmitted alarm within 10-25 seconds.	Yes
Req 2: Client setup process	Active Test 10	The system was easily set up with the created install script, without any manual configuration.	Yes
Req 3: Client resource utilization	Passive Test 4	RAM usage increased by 0.07 percentage points, CPU usage averaged 0.68%.	Yes
Req 4: Client security on host	Active Test 7	No non-admin user can terminate the client or access the created files.	Yes
Req 5: Detect offline hosts	Passive Test 1, 2 & Active Test 8	The system achieved a 100% heartbeat transmission consistency rate. It also correctly triggered alarms after the 5-minute threshold.	Yes
Req 6: Detect defined states	Active Test 3, 5 & 6	The system correctly distinguished the different states at shut off, restart and startup.	Yes
Req 7: Automated recovery	Active Test 4	The system was able to use the ME to remotely execute startup attempts when necessary, and correctly triggered an alarm after the 90-seconds threshold of no startup message.	Yes
Req 8: Securely transmit telemetry	Passive Test 5	The database remained consistent throughout the testing period, utilizing HTTPS also secured the transmitted telemetry.	Yes
Req 9: Force reconnect	Active Test 2	When manually disabling the network, the client successfully forced the network back on.	Yes
Req 10: Mass disconnection	Active Test 9	The server correctly escalated to a mass disconnection alarm when the offline threshold for a room was exceeded.	Yes
Req 11: False positive rate	Passive Test 3	The system generated zero false positives during normal lab activities.	Yes
Req 12: Device overview	GUI Test 1	Participant easily identified online/offline devices and navigated tabs intuitively.	Yes
Req 13: Display and handle alarms	GUI Test 2 & 3	GUI clearly displayed new alarms, with a straightforward handling logic.	Yes
Req 14: Create reports	GUI Test 4 & 5	The GUI provided an intuitive workflow for creating and editing reports.	Yes

5
Discussion

The evaluation results that were presented in the previous chapter indicate that the core objectives established in Section 1.4 have been met. While the system would need a larger testing suite during a real long term deployment in order to evaluate its reliability and deem it viable, the current outcomes suggest a functional prototype of a monitor system. Therefore, this proof of concept successfully demonstrates the proposed event-driven architecture.

5.1 System Limitations

Although the system demonstrates the feasibility of detecting hardware-related anomalies in laboratory environments, several architectural, operational, and security-related limitations must be acknowledged.

5.1.1 Scalability

The system was validated in a limited-scale environment consisting of a single laboratory room with a relatively small number of devices. While this was sufficient to verify core detection logic and event handling, it does not fully represent the diversity of hardware configurations, usage patterns, and network conditions present in a full campus deployment. Testing was conducted during a period of relatively low student activity, meaning the system was not exposed to realistic high-usage conditions. As a result, it was not possible to evaluate performance under peak load or to perform comprehensive stress testing representative of normal campus usage.

5.1.2 Network Dependency and Detection Latency

The monitoring strategy relies heavily on network connectivity for heartbeat transmission and event reporting. If a device is physically disconnected from both network and power, detection depends on heartbeat timeouts and potential automated recovery attempts. Although the selected heartbeat interval provides near real-time monitoring, a short detection delay is inherently unavoidable in network-based monitoring systems. Immediate detection cannot be guaranteed in scenarios involving abrupt power removal or deliberate network isolation. Furthermore, temporary network instability may result in transient disconnections. While timeout thresholds and staged escalation reduce unnecessary alarms, the balance between responsiveness and robustness inherently involves trade-offs.

5.1.3 Static Detection Strategy and Parameter Calibration

The implemented alarm logic is rule-based and statically defined. Detection thresholds such as heartbeat intervals, disconnection timeouts, and room-based percentage limits were calibrated during development rather than derived from statistical analysis. No adaptive or machine-learning-based detection mechanism is incorporated. While this improves transparency and predictability, it limits the system's ability to dynamically adapt to evolving usage patterns or detect more subtle behavioural anomalies. In a full deployment scenario, these parameters would require continuous tuning based on operational data.

5.1.4 Manufacturer and OS Compatibility Limitations

Since DCM had to be installed on the host to ensure that the intrusion detection feature could be implemented as intended, as described in Section 3.2.5. The client can only function properly on Dell manufactured computers. This did not present an obstacle to the clients development as all the hosts that were provided were manufactured by Dell. However, this does present a major issue towards deploying the system in a real world context. Similarly, the client is only intended to be operated on Windows based hosts. While not as critical of a limitation, it does present a weakness. Further discussion on how to remedy these limitations are presented in Section 5.2.4.

5.2 Future Development Potential

As the purpose of this project was to develop prototype system rather than a production-ready solution. Several aspects require further investigation and refinement before deployment in a real-world environment can be considered. The following subsections discuss a number of areas that were identified during the project as having significant potential for future development.

5.2.1 False Positive Mitigation

False positives are a major concern in monitoring systems, as frequent incorrect alarms reduce trust in the system and increase the likelihood that legitimate alarms will be ignored. In this project, the primary strategy for reducing false positives has been to focus on events that are considered unlikely to occur during normal host usage. Contextual information, such as room scheduling, is also incorporated into the decision-making process to distinguish between expected and suspicious behaviour.

However, the system has not been evaluated in an environment that reflects the normal day-to-day usage of university computer laboratories. As a result, it is currently unclear whether the implemented detection logic results in an acceptable false positive rate under realistic operating conditions. Further work should therefore investigate how the event data produced by the system can be analysed to improve

the distinction between legitimate and malicious activity.

Methods to perform such analyses could include statistical modelling combined with machine learning to identify patterns of usage of the monitored machines. Methods like these could be used to allow the system to dynamically adapt to alarm thresholds based on observed behaviour rather than predefined thresholds. Such techniques were considered in the planning phase of the project. This was however not implemented as it was deemed to not be enough time left at the point implementation would have begun. Furthermore, the lack of available monitoring data would have made implementation of such features difficult.

5.2.2 Peripheral Monitoring

The project has been intentionally limited to monitoring components located within the chassis of a monitored host. As a result, no functionality exists for detecting peripheral devices such as mice, keyboards, or displays. This limitation reduces the overall coverage of the system and may be considered a weakness in scenarios where peripheral tampering is relevant. Feedback from the Chalmers IT department indicated that peripheral monitoring would be desirable in a physical security monitoring system.

However, this functionality was not included in the current implementation due to concerns regarding false positive detections. Peripheral devices may be frequently connected and disconnected during normal use, resulting in potentially unreliable detection and classification challenges without introducing noise or requiring more complex device identification logic.

5.2.3 AMT Utilization

As AMT compatibility is standard across Chalmers laboratory computers, further work should investigate how AMT functionality can be utilized to expand the system's monitoring suite. Due to not being part of the original plan, AMT support was incorporated at a late stage of the project. As such only a limited subset of its functionality was explored. It therefore remains unclear whether all functionality currently provided by the client application could instead be implemented directly through the ME.

Particular attention should be given to the use of ACM rather than CCM as the provisioning mode. As discussed in section 2.3.5, ACM provides greater access to the functionality of the ME and may therefore enable more advanced monitoring and recovery features. This could potentially reduce the system's dependency on the client application running within the operating system, thereby increasing resilience against software-level interference and host system failures.

The project did attempt to investigate ACM provisioning. One possible approach is the use of a certificate server, which was not pursued due to the additional cost

associated with obtaining certificates from Intel-approved vendors. Another possibility is USB-based provisioning, where a configuration file is loaded during system boot. This was attempted but did not succeed for reasons that were unable to be established. Future work should investigate which provisioning approach is most suitable for large-scale deployment within the Chalmers laboratory environment.

5.2.4 Manufacturer and OS Compatibility Improvements

As discussed in section 5.1.4, the current system is limited to Dell-manufactured Windows-based hosts. A major area for future improvement is to extend compatibility to additional hardware manufacturers as well as alternative operating systems such as Linux.

Extending manufacturer support is expected to be the less complex of these two improvements. At present, only the client's intrusion detection component depends on Dell-specific software, namely DCM. It is currently unclear whether the issue described in Section 3.2.5 is specific to Dell hardware or if it is a more general limitation across different manufacturers. If the behaviour is not Dell-specific, similar vendor-provided utilities or interfaces could potentially be used to achieve equivalent functionality on other systems. The current implementation is hardcoded to use DCM, and future work should focus on abstracting this dependency. This could be achieved by allowing the client to select or configure manufacturer-specific monitoring tools during client installation, rather than relying on a hardcoded solution.

Supporting Linux-based systems presents a more significant challenge. As the project scope was limited to Windows based machines, it has not investigated whether there exists support for similar hardware level monitoring for Linux-based systems. A separate client implementation would be required without major alterations to the existing client's dependencies and architecture. Currently the client implementation relies heavily on Windows-specific functionality such that it can not be run or compiled on non Windows based machines.

5.2.5 Scalability

Scalability refers to the system's ability to handle an increasing number of monitored devices without significant performance or reliability degradation. Although the current implementation is designed as a proof of concept with a limited number of workstations, several architectural decisions were made with future scalability in mind.

The System follows a centralised client-server architecture, where each client independently communicates with the server through RESTful API calls. This design enables horizontal scaling on the client side as adding more monitored devices does not require modifications to existing client. Each device operates independently and only communicates with the server when events occur or at fixed heartbeat intervals, reducing unnecessary network traffic compared to continuous polling systems.

The current implementation relies on a single server instance, which limits horizontal scalability and introduces a single point of failure. A more scalable approach would involve distributing the system across multiple nodes, for example through containerization or cloud-based infrastructure.

5.3 Vulnerabilities

Despite the implemented safeguards, several potential vulnerabilities remain within the system architecture. The remaining vulnerabilities are primarily related to design choices, network dependency, and authentication handling.

5.3.1 Policy and Access-Control Vulnerabilities

The integration of room scheduling data into the alarm suppression logic introduces a policy-based vulnerability. Certain alarm categories are suppressed when a room is actively booked in order to reduce false positives during legitimate usage. However, if a malicious actor is aware of this suppression mechanism, they could intentionally reserve a laboratory through publicly accessible scheduling systems. In order to reduce alarm sensitivity during a planned tampering attempt. Although such exploitation would leave traceable booking record and is unlikely to remain undetected over extended periods, it could delay immediate alarm generation. In addition, the GUI communicates with the server through API endpoints that currently lack dedicated authentication for administrative access. Without proper network-level restrictions, unauthorized users could potentially access system state information or interact with administrative functions. Although this does not directly affect the hardware integrity of monitored devices, it represents an access-control weakness. A production deployment would therefore require stronger authentication mechanisms or strict IP-based access restrictions to secure the administrative interface.

5.3.2 Network Isolation and Timed Disconnection

Another vulnerability arises from the system's reliance on network based heartbeat communication. An attacker could intentionally disconnect a device from the network prior to tampering with its hardware. While missed heartbeats are detected and may trigger recovery attempts, alarm generation depends on predefined timeout threshold. A carefully timed attack could exploit this detection window, allowing limited physical manipulation before escalation occurs. Although such delays are typically short due to the selected heartbeat interval they cannot be fully eliminated in network-based monitoring systems.

5.3.3 API Key Extraction and Heartbeat Spoofing

Authentication is implemented through API keys and stored on monitored devices. If an attacker with administrative access to a host system were able to extract the API key, it would be theoretically possible to send forged heartbeat messages to

the server, simulating normal operation. In practice, sustained spoofing would be difficult to maintain, as hardware validation and event reporting mechanisms would likely reveal inconsistencies over time. However, short-term spoofing could delay immediate detection of a disconnection or tampering event.

5.4 Ethical aspects

It is important to critically reflect on the ethical principles guiding the system in a real-world deployment context. Monitoring systems in educational environments must carefully reconcile security objectives with the preservation of an open and trusting academic atmosphere. Even when technically limited to hardware integrity and availability, the presence of such a system may influence how users perceive their autonomy within shared spaces.

If deployed at scale, the system would introduce a formalized monitoring layer within laboratory environments. Although restricted to technical indicators, continuous monitoring may affect how students experience the space. A key deployment challenge is ensuring that enhanced security measures do not undermine user confidence. To minimize negative impact, several measures would be essential. Clear and accessible communication should inform users about what is and is not monitored. Unfortunately this can be used to gain knowledge about the system and be exploited by unauthorised people, but transparency out weighs the risks. Emphasizing that no keystrokes, screen activity, or personal data are collected would help prevent misunderstandings. Furthermore, data retention policies should be carefully defined in a deployment scenario. Limiting long-term storage of telemetry data and anonymizing identifiers would reduce privacy risks while preserving operational value. Should advanced analytics or machine learning be introduced in future iterations, further ethical evaluation would be required.

Another important consideration is function creep. Systems initially developed for narrowly defined purposes may gradually expand in scope over time. While the current implementation is limited to hardware-related data, the architectural framework could theoretically be extended to collect additional types of data. Without clear regulatory boundaries, such extensions could unintentionally shift the system from infrastructure protection toward behavioural monitoring. Maintaining strict technical and policy constraints is therefore essential to preserve the original intent of the project.

Ultimately, the ethical viability of the monitoring system depends not only on its technical limitations but also on its regulatory frameworks. Clear documentation, transparent communication with stakeholders, and strict role separation are necessary to ensure that the system remains aligned with its intended purpose.

6
Conclusion

This project demonstrated the feasibility of an event-driven monitoring system designed to detect abnormal hardware-related events in university computer laboratories. By combining heartbeat monitoring, chassis intrusion detection, hardware validation, and centralised alarm handling, the implemented prototype was able to provide improved situational awareness and faster detection of potential tampering incidents. The system was intentionally limited to hardware-level monitoring and not intended to replace existing cybersecurity infrastructure or provide protection against software-based threats, such as malware or network intrusion. Instead, the project focused on evaluating how a lightweight proof of concept solution could support the detection and deterrence of hardware theft in shared laboratory environments.

The proposed solution was implemented as a centralised client-server architecture consisting of lightweight monitoring clients deployed on laboratory computers, a central evaluation server, a persistent database, and a graphical user interface for administrative oversight. This architecture enabled event-driven monitoring through periodic heartbeats and real-time hardware event reporting, while maintaining minimal impact on normal user activity. Integration of automated recovery mechanisms and contextual alarm evaluation further demonstrated how the system could move beyond passive monitoring to active response and improved operational reliability.

The result demonstrated that the prototype was capable of reliably detecting and reporting simulated hardware-related threat scenarios defined in the project requirements. During testing, the system successfully identified events such as chassis intrusion, heartbeat loss, unexpected shutdowns, hardware mismatches, and mass disconnections while maintaining stable operation and low resource utilisation on monitored hosts.

Despite successful development, several limitations remain. The system was evaluated within a relatively small laboratory environment consisting of only four computers, over a one-week testing period, and is currently based on Dell-specific functionality and Windows-based hosts. Future work should therefore focus on improving scalability, reducing false positives under real-world operating conditions, and extending compatibility across hardware manufacturers and operating systems. Despite this, the project demonstrates that an event-driven monitoring approach can provide a solid foundation for improving hardware security, operational awareness, and incident response within shared university computer laboratory environments.

Bibliography

[1] Intel Corporation, "What Is Remote Monitoring and Management (RMM)?", Intel. [Online]. Available: https://www.intel.com/content/www/us/en/learn/what-is-rmm.html. [Accessed: Mar. 22, 2026].

[2] ConnectWise, LLC, "ConnectWise RMM," ConnectWise. [Online]. Available: https://www.connectwise.com/platform/rmm. [Accessed: May 17, 2026].

[3] Paessler GmbH, *Welcome to PRTG | PRTG Manual*. Paessler - The Monitoring Experts, 2026. [Online]. Available: https://www.paessler.com/manuals/prtg/welcome_to_prtg. [Accessed: Mar. 22, 2026].

[4] Absolute Software Corp., "Absolute Secure Endpoint," Absolute. [Online]. Available: https://www.absolute.com/platform/secure-endpoint. [Accessed: May 17, 2026].

[5] Prey Inc., "Effective & reliable device tracking to get full fleet visibility," Prey. [Online]. Available: https://preyproject.com/features/device-tracking-and-location. [Accessed: Mar. 30, 2026].

[6] R. Santika, B. A. Pramudita, N. F. Puspita, and S. Sumaryo, "Laptop Anti-Theft System with Tracking and Image Capture Device Based on Internet of Things Technology," in *2023 International Conference on Data Science and Its Applications (ICoDSA)*, 2023, pp. 53–58, doi: 10.1109/ICoDSA58501.2023.10276600.

[7] S. M. Khalil, H. Bahsi, H. O. Dola, T. Korõtko, K. McLaughlin, and V. Kotkas, "Threat Modeling of Cyber-Physical Systems - A Case Study of a Microgrid System," *Computers & Security*, vol. 124, p. 102950, Jan. 2023, doi: 10.1016/j.cose.2022.102950.

[8] The Debian Project, "Our Philosophy: Why we do it and how we do it," Debian. [Online]. Available: https://www.debian.org/intro/philosophy. [Accessed: Apr. 29, 2026].

[9] N. Ferdiansyah, D. A. Rahayu, A. Karim, and R. Permala, "Development Of Web-Based Application Using RESTful API As Utilization Tool Of AIS Data LAPAN Satellite," in *2022 IEEE International Conference on Aerospace Electronics and Remote Sensing Technology (ICARES)*, 2022. [Online]. Available: https://ieeexplore.ieee.org/document/9993545

[10] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," Internet Engineering Task Force, RFC 8446, Aug. 2018. [Online]. Available: https://datatracker.ietf.org/doc/rfc8446/. [Accessed: Apr. 16, 2026].

[11] F5, Inc., "NGINX," NGINX. [Online]. Available: https://nginx.org/. [Accessed: Apr. 29, 2026].

[12] Broadcom Inc., "Spring Boot Overview," Spring Boot. [Online]. Available: https://docs.spring.io/spring-boot/index.html. [Accessed: Apr. 13, 2026].

[13] H. Garcia-Molina, J. D. Ullman, and J. Widom, *Database Systems: The Complete Book*, 2nd ed. Pearson Education, 2013.

[14] The PostgreSQL Global Development Group, "About," PostgreSQL. [Online]. Available: https://www.postgresql.org/about/. [Accessed: Apr. 9, 2026].

[15] Supabase Inc., "Database," Supabase Docs. [Online]. Available: https://supabase.com/docs/guides/database/overview. [Accessed: Apr. 9, 2026].

[16] C. Tudose and C. Odubăşteanu, "Object-relational Mapping Using JPA, Hibernate and Spring Data JPA," in *2021 23rd International Conference on Control Systems and Computer Science (CSCS)*, 2021, pp. 424–431, doi: 10.1109/CSCS52396.2021.00076.

[17] Cheat Sheets Series Team, "SQL Injection Prevention - OWASP Cheat Sheet Series," OWASP. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html. [Accessed: Apr. 9, 2026].

[18] Microsoft Corp., "Windows Management Instrumentation," Microsoft Learn. [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/wmisdk/wmi-start-page. [Accessed: Apr. 21, 2026].

[19] Dell Inc., "Alerts in Dell Command Monitor 10.3," Dell Support. [Online]. Available: https://www.dell.com/support/manuals/sv-se/command-monitor/dellcommandmonitor_rg_10.3/alerts-in-dell-command-monitor-103?guid=guid-f58103ed-b74f-4547-b538-be1f9eaba081&lang=en-us. [Accessed: Apr. 21, 2026].

[20] Vue.js, "Introduction," Vue.js. [Online]. Available: https://vuejs.org/guide/introduction. [Accessed: May 10, 2026].

[21] Vue.js, "Using Axios to Consume APIs," Vue.js v2 Cookbook. [Online]. Available: https://v2.vuejs.org/v2/cookbook/using-axios-to-consume-apis.html?redirect=true. [Accessed: May 11, 2026].

[22] W. Kangas, H. Khan, A. Lindell, J. Nemeth, E. Quach, and M. Zanjani, "A Network-Driven Approach to Theft Detection in Shared Computing Environments," Bachelor thesis, Dept. Comput. Sci. Eng., Chalmers Univ. Technol. and Univ. Gothenburg, Gothenburg, Sweden, Rep. CSE 25-16, 2025. [Online]. Available: http://hdl.handle.net/20.500.12380/310697

[23] BI.ZONE, "wmi: WMI client for Go," pkg.go.dev. [Online]. Available: https://pkg.go.dev/github.com/bi-zone/wmi. [Accessed: Apr. 21, 2026].

[24] Device Management Toolkit, "rpc-go: Remote Procedure Call library for Go," GitHub. [Online]. Available: https://github.com/device-management-toolkit/rpc-go. [Accessed: Apr. 21, 2026].

A

Evaluation Visualizations

The tests were performed utilizing Windows Performance monitor. The same computer, located in ED-4209, was used for both tests and was left idle for the duration. The graphs were created using a custom python script created with the help of the Gemini AI model.

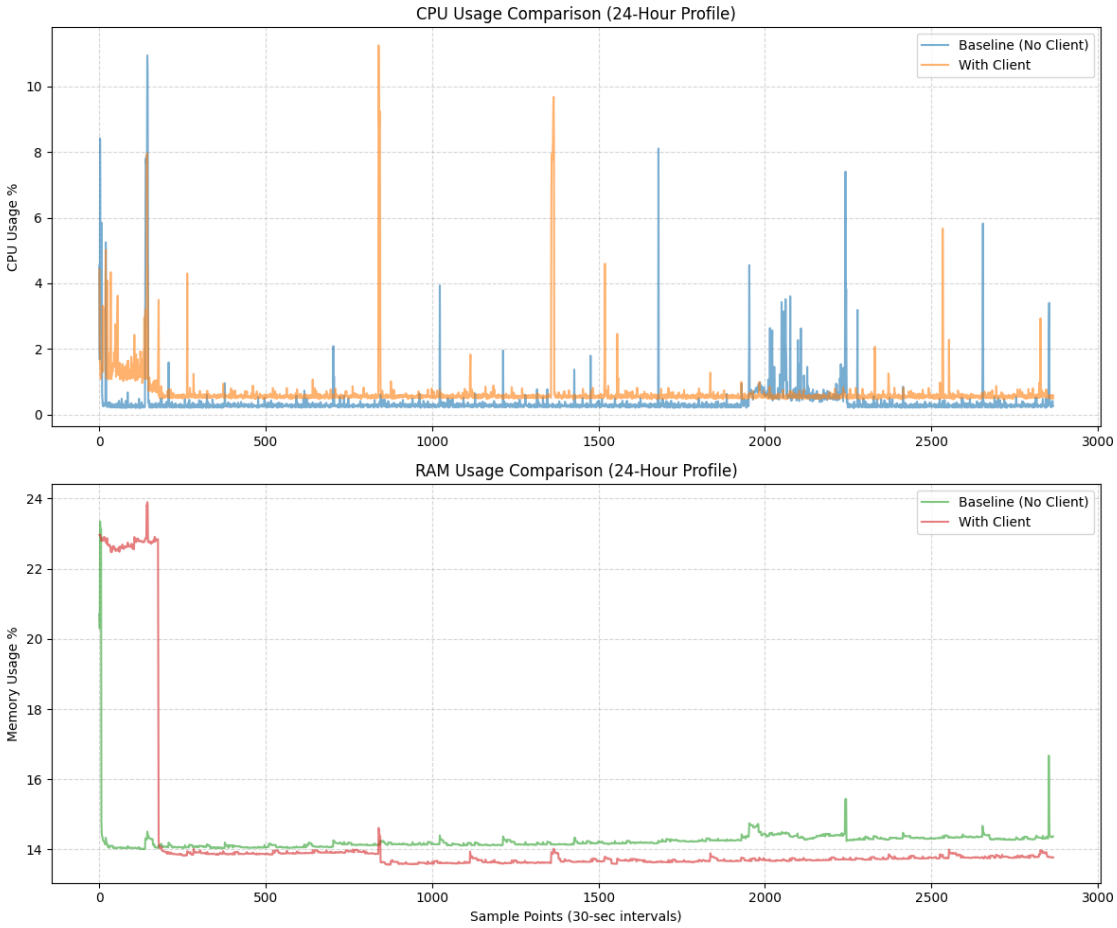

Figure A.1: Comparison of CPU and Memory Utilization

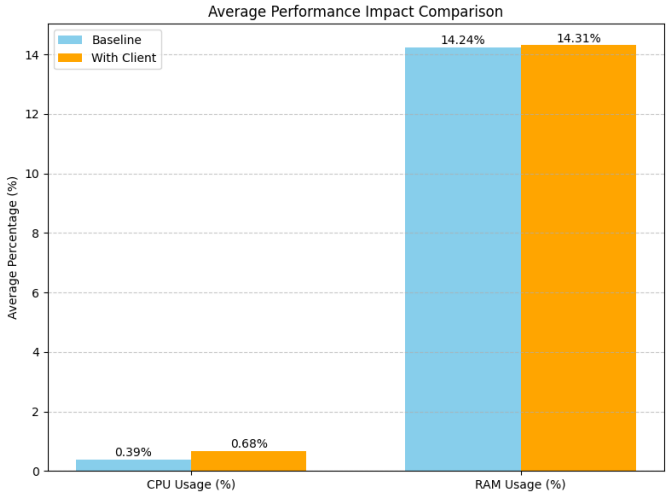

Figure A.2: Comparison of Average Performance Impact of RAM and CPU Utilization

B

Install script

Successful installation of the client on the host using the installation script, including AMT configuration by setting the password during the installation process.

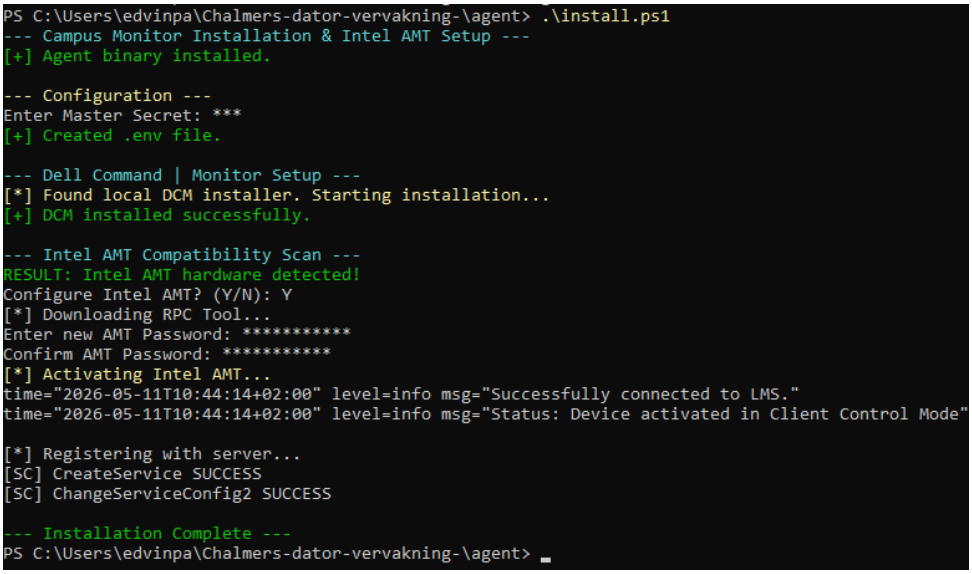

Figure B.1: Successful client installation and AMT configuration