Identification of Data Privacy Challenges and Development of Solutions for the Edge Components of the Telemedicine Datapath

PhD Thesis

Zoltán Szabó
Supervisor: Dr. Vilmos Bilicki

A THESIS SUBMITTED FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
OF THE UNIVERSITY OF SZEGED

Szeged
2023

Contents

List of Figures

List of Tables

Preface

I have vague memories about that time when my father brought home our first computer. I might have been around 3 or 4 years old. It was a frightening device, with its constant noise, weird text flashing on its enormous monitor, and numerous ways it could go wrong. Still, as the years went by and new operating systems came along—not to mention more programs, video games and later, internet access—I started to use it more and more and finally accepted that it was the ultimate tool for human creativity—a device that can be used to realize and try out ideas, concepts, and theories, to communicate and cooperate with people all around the world, to access vast databases full of information, and to improve productivity. I became a sort of black sheep in my family; with my father being a doctor, my mother a pharmacist, and my sister a biologist, I chose IT and software development as my calling.

It is no small irony that when I started working for the Department of Software Engineering at the University of Szeged, I found myself working on medical applications and discovered the vast and challenging field of e-health that offers many opportunities to improve traditional health care, the quality of patient care, and facilitate and relieve the work of healthcare workers. However, as I worked, I was gradually faced with challenges and problems: with every major step forward and advancement in e-health, came a new difficulty or problem to solve. And none of these was as prominent as the issue of privacy and access control, guaranteeing the protection of health data on increasingly dynamic infrastructures while meeting every requirement of the given use case scenario. When the idea of working towards a PhD degree came up, the opportunity to do this almost presented itself as my field of research: to assess the needs of healthcare applications for access control and to develop solutions that might one day result in a generic, standardized answer to the problem. In the six years since then, the research has been sometimes easier, sometimes harder, while the need for it has been demonstrated by the growing number and impact of telemedicine applications as well as by unfortunate events such as the COVID-19 epidemic. Sometimes I hit a dead end, sometimes I was stuck on a problem for weeks or months, and sometimes I worked long nights and weekends without any significant progress or results, but I never gave up.

However, it is not only my own merit, and I can consider myself lucky since many people have helped me along the way over the years. First and foremost, I would like

to thank my supervisor, Dr. Vilmos Bilicki, for his mentoring and support over the years, for always providing me with good ideas, helping me overcome difficulties, and steering my research in the right direction. This dissertation would not have been possible without the support of my partner, Szilvia Lovai, who has supported and encouraged me with endless dedication, patience, and love throughout the years. I would also like to thank my parents and my sister for being there for me and encouraging me, as well as my relatives and friends for helping me overcome problems and never letting failures or difficulties discourage me. I owe a debt of gratitude to Zsombor Hunyadi, Zoltán Richárd Jánki, Péter Seffer, and all the current and former members of the Inclouded team, who either are or were the best colleagues I could have wished for over the years.

Zoltán Szabó, August 2023.

Chapter 1

Introduction

In recent decades, the integration of healthcare and informatics has evolved rapidly. With smart devices, IoT sensors, and widely accessible databases and information, it is simpler than ever before for patients and healthcare professionals to monitor and analyze the various therapies, procedures, and biological functions, as well as collect data from these areas.

To support these tendencies, there has been a growing interest in defining a general standard format for the storage and handling of healthcare data, with the Fast Healthcare Interoperability Resources (FHIR) standard from the Health Level 7 (HL7) organization representing the pinnacle of this effort. Nonetheless, as a result of this rapid evolution and the growing popularity of the FHIR standard, a significant issue has become increasingly prominent in the domain: the lack of generalized, appropriate access control and privacy management that can complement and support these standards and the applications that use them. While the creators of the FHIR standard only provided suggestions based on traditional access control methodologies, such as Attribute-Based Access Control (ABAC) and Role-Based Access Control (RBAC), the challenges and requirements of the telemedicine domain have proven to be more complex and demanding. Multiple researchers began developing methods for a secure, dependable, and dynamic solution for telemedicine access control, and we have seen numerous possibilities. However, these solutions typically only support a singular aspect of the domain's infrastructure, which includes the various stations of the telemedicine data path.

In the following dissertation, I present my research, methods, and findings regarding the design of a solution that can offer to meet some of these challenges and assist developers and researchers in designing more standardized, optimal, and effective implementations.

In Chapter 2, I present the background of my research and an overview of the various tools, standards, and concepts I have used during my work. Following this is Chapter 3, where I introduce my findings concerning the nature and requirements of the telemedicine data privacy domain; a formal definition of the policy categories required to cover the use cases; an analysis of the role and importance of the edge networks of the cloud-integrated infrastructures and the unique challenges they add to the problem; a formal categorization of the edge types based on their capacity for data handling and access control; an examination of smartphone based peer-to-peer networks as special cases of the edge; my design and implementation of a framework to meet the established requirements, and an analysis of the results I have obtained after running several evaluations on test environments simulating the processing edge and caching edge. At the end of the chapter, I also introduce an open-source simulation tool for patient flow to simulate the various scenarios in the health care system and analyze the possible impact of the latencies my access control solution has on the process.

The focus of Chapter 4 shifts to frontend applications at the end of this telemedicine data path. I introduce a novel, Large Language Model (LLM)-based methodology to identify vulnerabilities where leakage of sensitive, confidential data might occur due to improper compartmentalization of functionality and bad developer practices. This domain has proven to be exceptionally challenging so far, as this type of detection not only requires the semantic analysis and interpretation of the static source code, which is already a challenge on its own due to the complex nature of current frontend frameworks, but also requires the proper detection of sensitive data elements in the application and an assessment of how well they are handled or protected with the available tools of the framework. To address this, I introduce formal taxonomies for both data sensitivity and component protection based on the impact a potential leakage of the data might have. I introduce a series of prompts, implement these categorizations using techniques of prompt engineering, and form an analysis pipeline. I then present the results of the sensitivity detection using models of the Generative Pre-Trained Transformer (GPT) model, namely the GPT-3.5 and GPT-4, via API calls and prompts. The subject of the first evaluation is a preliminary on a dictionary containing variable names, followed by a second set of 292 web application components obtained from open source Angular projects. This is followed by the results of a similar experiment on the 292 components to detect protectivity levels, and then, based on the results of the two evaluation sets, the results of the full vulnerability detection for possible data leakage, with the GPT-4 achieving an overall success rate of 88%.

Finally, Chapter 5 includes the Conclusion of my work, and the summary of my results.

1.1 Contributions

The ideas, figures, tables and results included in this thesis were published in scientific papers (listed at the end of the thesis). I am responsible for the following contributions:

Chapter 3.: I examined the structure of a cloud-integrated telemedicine infrastructure and analyzed the privacy and access control requirements of the domain. Based on these requirements, a taxonomy of the required access control policy types was formally defined. I explored the capabilities of the edge networks beyond the cloud and identified two types of edge infrastructure where access control policies need to be enforced: the processing edge and the caching edge. I participated in a research group concerning the peer-to-peer capabilities of smartphones, in which he has examined state-of-the art solutions and helped with the debugging and error analysis of the application used in the data collection campaign. Testing environments simulating the processing edge and an offline, separated caching edge were established, and test policies based on the defined categories were evaluated on them by me, observing latencies and resource requirements. Finally, I have developed a unique simulation tool based on open source libraries to simulate the patient flow in a healthcare environment that can be used to validate the latencies caused by the access control solutions. I assisted in the validation of the simulation tool using publicly available statistics from hospitals.

Chapter 4.: I examined the various vulnerabilities concerning frontend applications and the difficulties of detecting them using only static code analysis. I defined a formal categorization of data sensitivity and component protection based on the impact of an eventual data leakage. I developed a series of prompts based on these categorizations for large language models in an attempt to detect the CWE-653 vulnerability, where improper isolation of the code handling sensitive information might lead to unauthorized access and usage of said data. I validated the sensitivity categorization and the analytical capabilities of the GPT-3.5 and GPT-4 models by establishing a variable-name dictionary that has been analyzed and categorized by the models. I sampled a number of open source Angular projects, which were then analyzed by the developed prompts for occurrences of the CWE-653 vulnerability. I examined the results and analyzed the errors and anomalies at each step.

Chapter 2

Background

In the following Chapter, I present the background and motivations for the research that has served as the basis for the dissertation.

2.1 Data Privacy Challenges of the Telemedicine Domain

2.1.1 The Telemedicine Domain

In recent years, the healthcare industry has undergone a transformational evolution, fueled in part by technological advancements and the proliferation of cloud-based solutions. Telemedicine, which utilizes the power of the internet to provide medical services and information at a distance, is a particularly significant innovation. The expansion of e-health and telemedicine services offers numerous advantages, including the elimination of geographical barriers to care, the improvement of patient engagement, and the potential reduction of costs. However, as with many innovations, the potential benefits are accompanied by obstacles.

Providing secure and controlled access to patient data stored in the cloud is one of the most pressing issues associated with telemedicine. Given their sensitivity, the confidentiality and integrity of medical records are of the utmost importance. Years ago, traditional access control mechanisms such as Role-Based Access Control (RBAC) and Attribute-Based Access Control (ABAC) were used to manage information access. However, in the context of telemedicine, these solutions disclose certain deficiencies.

Before discussing the obstacles, it is essential to comprehend what e-health entails. E-health is an umbrella term for all digital health tools and services, including electronic health records (EHR), mobile health applications, and telemedicine services. When we say 'e-health' we are referring to the tools, applications, online platforms and infrastructures that host, manage, and deliver these digital health services.

Telemedicine, a subset of e-health, refers specifically to the remote delivery of healthcare services. Utilizing video conferencing tools, online messaging systems, and other forms of digital communication, it permits patients to consult and share medical records with healthcare providers without the need for physical visits.

The sensitive nature of the data managed in the e-health cloud is inherent. The stored information, including personal identifiers, medical histories, prescriptions, and diagnostic results, could be harmful if it fell into the wrong hands. Unauthorized access or data intrusions may result in violations of privacy, financial misconduct, and even potential injury to patients.

Given the implications, it is evident that access control mechanisms must be robust. They must ensure that only authorized personnel have access to the required data, and even then, only to the data relevant to their role.

RBAC

RBAC [46] is one of the most popular access control techniques. It functions by designating users to roles and associating permissions with these roles. A user's role determines the resources or data to which they have access. Nevertheless, the healthcare industry is dynamic. Patients are transferred from one healthcare provider to another, physicians may consult with specialists from other departments or institutions and specific roles might change in the context of time and location. In such situations, the rigid structure of RBAC can become a bottleneck, since constant adaption might be too slow and error-prone. Also, a common critique of RBAC is the tendency to give more permissions than necessary for certain roles to ensure smooth operations. In an environment that is as sensitive as e-health, this could lead to potential security risks.

ABAC

ABAC [122] determines access based on attributes. These attributes may be associated with the user, the resource, or the environment. Compared to RBAC, it provides more granular control, allowing for more nuanced access decisions. However, ABAC is not devoid of obstacles. The flexibility is obtained at the expense of complexity. Installation, management, and maintenance of ABAC systems require specialized knowledge and can be resource-intensive. Moreover, due to the dynamic nature of attributes and the requirement for real-time evaluations, ABAC systems can occasionally experience performance issues.

The Challenge

Due to the inherent limitations of RBAC and ABAC, an integrated and comprehensive approach to access control in the e-health cloud is unquestionably required. Healthcare frequently necessitates contextual access. During an emergency, a doctor may require access to a patient's medical history, regardless of their allocated function or attributes. Patients frequently consult multiple healthcare providers and specialists, necessitating the integration and communication of systems. Additionally, the healthcare industry is constantly evolving. Access control systems must be adaptable enough to accommodate these alterations without jeopardizing security. Before we can progress further, it is essential to investigate and implement comprehensive access control solutions that can address the unique challenges posed by the digital healthcare environment. Only then will we be able to maximize the potential of the e-health cloud while maintaining the security and confidentiality of patient data.

2.1.2 The Telemedicine Datapath

The Telemedicine Data Path incorporates a variety of components, including hybrid databases, user applications, the Internet of Things (IoT), and periphery computing, among others. While integral to the seamless administration of healthcare services, each element presents its own set of challenges.

Hybrid Cloud

Adoption of hybrid clouds in telemedicine denotes a combination of both public and private cloud structures. This combination enables healthcare organizations to take advantage of the scalability and cost-effectiveness of public clouds while also leveraging the enhanced security features of a private cloud for more sensitive data. The agility of the hybrid cloud model enables healthcare institutions to scale resources flexibly in response to fluctuating demand. Managing both public and private cloud environments can be complicated due to their dual character. Moreover, the transmission of data between these platforms can introduce latency, a major concern in the healthcare industry, where time is frequently of the essence.

User Applications

User applications serve as the initial point of patient contact. From simple appointment scheduling interfaces to complex teleconsultation systems, these digital platforms bridge the divide between patients and healthcare providers. Their rise has democratized healthcare access, allowing patients to gain access to medical services from virtually anywhere. However, as the number of digital touchpoints grows, so do the obstacles. Today's plethora of user applications necessitates a high level of interoperability to ensure seamless data exchange. In addition, given the sensitivity of medical information, these applications must adhere to stringent security standards to prevent data breaches and unauthorized access.

Internet of Things

The Internet of Things (IoT) is woven into the fabric of contemporary telemedicine. Patient care has been revolutionized by the proliferation of interconnected devices, such as wearable health monitors and intelligent medical apparatus. These devices facilitate real-time monitoring of health, paving the way for preventative medical interventions. But with this data deluge comes the difficulty of effective data management. In addition, the always-connected nature of these devices makes them vulnerable to cyber threats, highlighting the need for stringent security measures.

Edge Computing

Edge computing is perhaps one of the most intriguing components of the Telemedicine Data Path. Edge computing concentrates on processing data closer to its source, such as an Internet of Things (IoT) device, as opposed to immediately routing it to a centralized cloud. This proximity processing significantly reduces latency, allowing time-sensitive medical decisions to be made in real-time. Additionally, it optimizes bandwidth utilization, relieving the strain on central servers. However, edge computing's decentralization is not without its challenges. It can be difficult to ensure data consistency between peripheral devices and centralized servers. Due to the limited character of the computational resources at the periphery, they may not always be suitable for sophisticated analytical tasks. Additionally, the distributed nature of periphery computing heightens security concerns. And most importantly, edge devices typically lack comprehensive security protocols, making them susceptible to potential breaches.

2.1.3 The FHIR Standard

Interoperability was mentioned several times as a recurring requirement of the e-health domain in previous chapters. The prominent Fast Healthcare Interoperability Resources (FHIR) [3] standard has been touted as a possible remedy to this problem. FHIR was developed by Health Level Seven (HL7) organization as a standard for the exchange of healthcare data. FHIR, unlike its predecessors, employs the most recent web standards and prioritizes simplicity and ease of implementation. It seeks to strike a balance by providing detailed clinical models and assuring that systems can read and write to its specifications without difficulty. By utilizing a modular approach, FHIR enables the consistent representation of clinical concepts, making it simpler for systems to comprehend shared data.

The optimism surrounding FHIR derives primarily from its design philosophy. By adopting modern internet conventions and ensuring compatibility with a variety of platforms, including EHR systems, mobile applications, and cloud services, FHIR has made significant strides in promoting interoperability. Its structure, which is based on resources, which are data elements with a standardized method of definition and representation, ensures that different systems can interpret these resources uniformly. Additionally, this design ensures that the addition of new resources or modifications to existing ones will not disrupt existing systems.

However, as with any ambitious undertaking, the standard faces obstacles. One of the most significant problems is the misconception that FHIR can address all interoperability issues by itself. While it provides a comprehensive framework for data exchange, its practical implementation must take workflows, user interfaces, and local requirements into account. Institutions must invest in training, development, and testing to ensure that the integration of FHIR yields significant operational benefits.

In addition, despite FHIR's vow to simplify data exchange, many healthcare institutions continue to utilize legacy systems due to the sector's complex landscape. Transitioning to FHIR necessitates both technological and organizational adjustments, which is easier said than done. Although the standard is robust, it is still evolving, and organizations may be hesitant to adopt a framework that may be subject to significant changes in the future.

The absence of an access control standard is a particularly conspicuous omission in the FHIR design. While FHIR provides a comprehensive framework for data exchange, it does not dictate who should access the data and under what circumstances. Given the sensitivity of healthcare data, access control is of the utmost importance. As previously discussed, traditional mechanisms such as Role-Based Access Control (RBAC) and Attribute-Based Access Control (ABAC) have their limitations, particularly in the dynamic world of telemedicine. Because FHIR lacks a standardized access control mechanism, and only provide room for ABAC and RBAC-based integrations, institutions must rely on external systems or custom solutions to ensure data security. This not only adds layers of complexity, but also risks fragmenting access control implementations across institutions, paradoxically exacerbating the interoperability problem.

Moreover, despite the fact that FHIR is designed to be adaptable and flexible, this adaptability can occasionally be a double-edged sword. Different institutions may implement FHIR in slightly different ways, resulting in variances that may impede the exchange of data in a seamless manner. Despite the fact that the standard provides guidelines, the complexity of healthcare data, coupled with varying regional regulations and requirements, can result in disparate FHIR implementations.

2.2 Static Detection of Potential Data Leaks

2.2.1 Software Vulnerabilities and the CWE database

Web applications have become the cornerstone of enterprises, governments, and institutions worldwide in the digital age. As the complexity of these applications increases, so do the vulnerabilities associated with them. If left unchecked, these vulnerabilities can expose systems to a variety of malicious activities, including data breaches and unauthorized system access. To ensure the security of digital assets, it is essential to comprehend the nature of these vulnerabilities and the instruments available to detect and mitigate them.

The Common Weakness Enumeration (CWE) [76] database is one of the most important resources pertaining to software vulnerabilities. The CWE, created and maintained by the MITRE Corporation, provides a standardized language for describing software security vulnerabilities. It provides an exhaustive taxonomy of known vulnerabilities, facilitating communication and collaboration between developers, testers, and security professionals on security issues. The CWE database functions as both a reference and a guide, assisting organizations in identifying potential code vulnerabilities and implementing best practices to prevent exploitation.

The CWE database is a valuable resource, but it also highlights the extensive and complex landscape of software vulnerabilities. Common vulnerabilities, such as SQL injection and cross-site scripting, are cataloged in the database alongside esoteric and subtle flaws that may not be immediately apparent. Those tasked with ensuring software security face a formidable challenge posed by this scope and variety.

Static code analysis is one of the primary methods for identifying vulnerabilities in software. This procedure involves scrutinizing the application's source code without executing it, in quest of problematic patterns or constructs. The allure of static code analysis lies in its capacity to rapidly scan large codebases and identify potential vulnerabilities before the software is executed. Nevertheless, as effective as this technique may be, it has limitations.

Static analysis lacks the runtime context by its very nature. Certain vulnerabilities, particularly those involving comparisons or dynamic data, may only manifest themselves under particular runtime conditions. Without the context provided by actual execution, such vulnerabilities may be missed by static analysis. Cases, where a deeper understanding of the functionality of the software, the context of its usage and the contents of its data are required, complicate the matter even further.

In addition to being a set of instructions, software code is a representation of logic and intent. Although static analysis tools are proficient at identifying patterns, they may grapple with subtleties and complexities that require a comprehension of the developer's intent. In the case of insufficient comparisons, the logic underlying why certain comparisons are made or not made may be profoundly ingrained in the functionality of the application. A tool that scans lines of code may not completely comprehend this intent, resulting in potential omissions.

In addition, the flexibility and diversity of contemporary programming languages, while a benefit for programmers, add an additional layer of complexity. Constructs, syntax, and semantics are distinct between languages. The manifestation of a vulnerability in one language may vary in another. While the CWE database provides a generalized description of vulnerabilities, it is a monumental undertaking to translate these descriptions into specific patterns for each language.

Also, the dynamic nature of software development practices and the constant introduction of new frameworks and libraries result in a constantly altering landscape of potential vulnerabilities. To remain pertinent, static code analysis tools require consistent updates. However, even with regular revisions, the overwhelming rate of change in the software industry can cause detection capabilities to become deficient.

Due to these difficulties, several vulnerability types could have been detected manually or via intensive testing until recently. However, the outlook for this domain changed radically with the arrival of Large Language Models (LLM).

2.2.2 LLM capabilities

Large Language Models (LLMs) are the most recent and significant innovation in the expansive field of artificial intelligence. The enormous scale and complexity of these models represent a paradigm shift in how machines process, generate, and comprehend human language, including programming code. Their development has not only widened the scope of natural language processing (NLP), but it has also paved the way for advanced vulnerability detection in software code, illuminating potential security flaws that might have gone unnoticed otherwise.

LLMs date back to the beginnings of machine learning and natural language processing. Initial models were small and had limited capabilities, but they laid the groundwork for what would follow. In terms of both complexity and capabilities, these models began to evolve as computational power and data sets multiplied. The advent of neural networks, and deep learning in particular, marked a turning point. Instead of relying on manually constructed features, models can now directly learn representations from data, resulting in a more accurate and nuanced understanding of language.

As these models grew, so did their ability to comprehend context, semantics, and the complexities of language. OpenAI's GPT series and Google's BERT and Bard demonstrated the potential of LLMs. They were able to produce coherent text, respond to queries, and even engage in simple conversations. Beyond these applications, however, a more substantial possibility was emerging: the capacity of LLMs to comprehend and generate programming code.

While structured and logical, programming code is also a form of language. It communicates purpose, functionality, and logic. With the proper training (or prompting in case of models such as GPT-3.5, GPT-4 and ChatGPT), the same models that could comprehend human language narratives could decipher the semantics of code. This aptitude created an abundance of opportunities. LLMs could assist developers by suggesting optimizations, documentation, and even generating code fragments in response to natural language queries. In addition to these applications, however, a potentially significant application might be the ability to detect vulnerabilities in static code.

As discussed previously, static code analysis entails scrutinizing software code without executing it. While traditional static analysis tools look for known problematic patterns or constructs on a semantic level, they frequently lack a deeper understanding of context and developer intent. This is where LLMs become relevant. With their comprehension of both semantics and context, LLMs can delve deeper into the code and comprehend its logic and intent. This enhanced knowledge enables them to recognize vulnerabilities that may not be readily apparent based on patterns alone.

For example, whereas a traditional static analysis tool could identify a potential SQL injection vulnerability based on a specific pattern, an LLM could comprehend the context as a whole. It could determine if user inputs are being sanitized, if the code segment in question is part of a larger authentication mechanism, or if it is part of a deprecated function that is no longer in use. This analysis can result in more accurate detection of vulnerabilities, reducing false positives and illuminating potential issues that may have been neglected.

In addition, LLMs can be trained on massive datasets containing known vulnerabilities, remedies, and associated discussions. This training enables them to remain current on the most recent vulnerability trends, patterns, and mitigation techniques. When analyzing code, they can utilize this extensive knowledge base to identify not only known vulnerabilities, but also potential zero-day dangers that share similarities with known issues.

However, the application of LLMs in static code vulnerability detection is not devoid of obstacles. The interpretability of these models is among the primary concerns. Given their size and complexity, it can be difficult to comprehend why an LLM flagged a particular code segment as vulnerable. Developers and security professionals may require a rationale for any issues that are identified, which LLMs in their current guise may struggle to provide.

Moreover, although LLMs are adept at comprehending context and semantics, they are not perfect. There is a possibility of false negatives, in which the model overlooks a real vulnerability. Vulnerability detection gaps may result from relying solely on LLMs without human oversight or without supplementing them with traditional analysis tools.

Nevertheless, their potential and complexity make them ideal research instruments for exploring new domain aspects, such as vulnerability detection, to determine whether we might be able to solve problems that were previously considered to be virtually insurmountable.

Chapter 3

Thesis Group I: Data Privacy at the Edge of the Telemedicine Datapath

In the following thesis group, I examined the requirements of the telemedicine data path for access management. I defined the conditions that must be met at each designated point of the telemedicine data path in order to reconcile security requirements and responsiveness. I defined a four-element taxonomy for the categories of policy enforcement to be implemented. I presented the edge as a critical case for access control, its two primary categories for access control, the processing edge and the storage edge, and our results on a special case of the edge, the analysis of the stability of smartphone peer-to-peer networks. Then, I defined two experimental environments, one for the processing edge and one for the storage edge, as well as my experimental results in said environments using a practical implementation of the access control framework in terms of latency. Lastly, I presented my patient flow simulator built with open source libraries, which can simulate the throughput of patients treated in a hospital ward and can be used to generate validation parameters for the developed access control framework in order to validate the system's delay tolerance.

Publications related to this thesis: [J1], [J2], [J3], [C5], [C6], [C7], [C8], [F9], [F10],[F11], [F12]

3.1 Introduction

In the mid-2010s with the emergence of the Fast Healthcare Interoperability Resources (FHIR) standard from HL7 [3], it seemed that we finally had the necessary tools to create e-health applications and databases that not only meet their respective institutional requirements, but also conform to international standards, making a networked health infrastructure more feasible. FHIR achieved this by defining a set of over 90 document templates that can be implemented in both JSON and XML formats and used to describe the entire healthcare workflow from administration to the daily events that a general practitioner or nurse is confronted with. FHIR has also made these documents customizable to meet specific requirements and cover specific areas. These attractive aspects have made FHIR the most popular and widely used healthcare communications standard from HL7 to date.

However, the FHIR standard had some alarming shortcomings [17] [27]. Although some of these have been addressed in the course of the various updates to the standard, one of the most pressing is still an open problem - namely the lack of clearly defined access control and security. While FHIR generally accepts custom extensions and adaptations of its standardized document types, it provides only a light template and some minor guidelines for security policy enforcement. This has led to a ”free-for-all” problem in the development of e-Health applications, with almost everyone developing their own solutions, which greatly corrupts the original concept of interoperability. With the introduction of the GDPR [34], the increasing integration of IoT and intelligent devices into the healthcare workflow [55] and in some cases the decision to use a heterogeneous backend [103] for handling accessibility and sensitive data, the complexity of this issue has increased. Another complicating factor is that the most popular current technologies for the backend are serverless and native cloud infrastructures. These present two main problems: with the advent of fog/edge computing, data processing takes place much closer to the end devices and user locations, and in these cases a large amount of sensitive data is stored in a public cloud.

The two main approaches recommended by the official documentation of the FHIR standard for these challenges are the attribute- or role-based policy controls, ABAC[123] and RBAC [45], respectively. With RBAC, the developer is able to assign specific roles to users that determine the level of access and possible operations in the system. Some typical roles in a medical system would be those of a doctor, nurse, patient, family member, and so on. In contrast, ABAC uses specific attributes of the user or the requested data to determine whether access should be granted.

However, in the current network topology, which combines IoT, intelligent devices, edge computing, private and public clouds, these methods in themselves are far from sufficient. To meet these needs, it is essential to develop a hybrid approach that combines the strengths of these two classical methods. Furthermore, it is important that these enforcement points can be placed at any part of the infrastructure to deal with the sensitive nature and processing requirements of the data. For example, while fog endpoints require a complete FHIR object, the connecting IoT devices may not be able to handle such complex data structures. In the case of a hybrid cloud solution, the data could also pass through a public cloud between the private cloud and end users, where naturally stricter policies and encryption are required than for the isolated, private parts of the infrastructure, as shown in Figure 3.2.

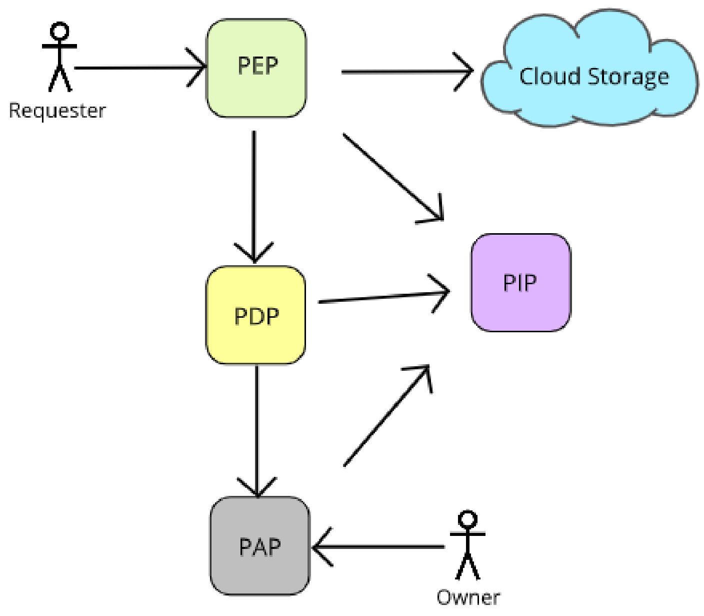


Figure 3.1: *The Policy Enforcement Flow between end users and the cloud*

A popular concept for such enforcement points is the concept of Policy Enforcement Point (PEP), developed by the standards organization OASIS as part of its eXtensible Access Control Markup Language standard [44], an extension of the classic ABAC model, also known as policy-based access control or PBAC. While we are committed to developing a custom, fine-grained security solution that is not subject to the strict limitations of the XACML standard, the concept of the PEP-based architecture is well suited to our needs. In the complete model shown in Figure 3.1, the responsibilities for access control and security enforcement are distributed across several components. The Policy Enforcement Point (PEP) is the key to the model that enforces policies and allows or denies access to resources. Administrators can define given policies at the Policy Administration Point (PAP), which are evaluated and stored by the Policy Decision Point, based on the user's identity or multiple identities, and recognized by the Policy Identification Point (PIP). The PDP's decision is handled and enforced by a PEP. This model also provides some room for customization, because the exact structure of these nodes can be defined by the developers, and the nodes have the ability to fuse multiple elements of the infrastructure into one.

My research aimed to combine these approaches while separating the access control process from the backend and frontend and putting it on the path of the data between the cloud and the end users.

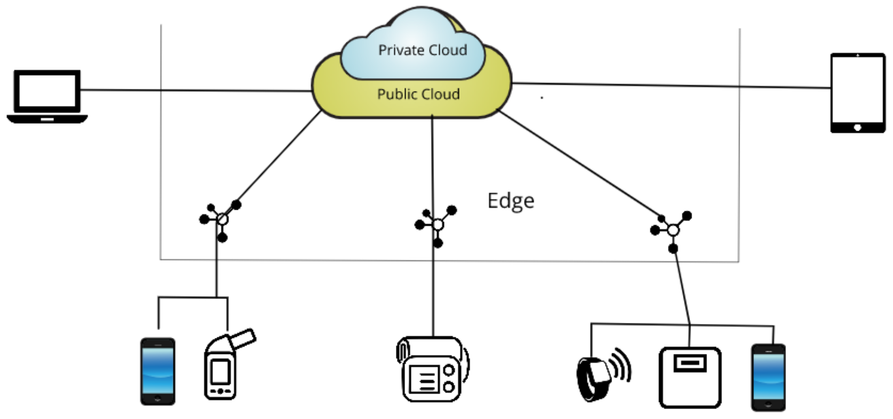


Figure 3.2: *A hybrid edge computing infrastructure with a public and a private cloud*

3.2 Related Works

While the authors of a 2013 comparative study based on 775 reviewed articles found that RBAC [43] was the most popular approach to manage access control in healthcare, this trend changed significantly with FHIR, and the preference between ABAC and RBAC became the de facto choice of the development team rather than industry standards.

For example, the developers of the application atHealth [98] succeeded in implementing a role-based methodology for their mobile application in 2017, recognising the lack of security in FHIR. However, there also were implementations of the ABAC model for access control to health records [82] in the same year.

To further complicate the issue of these two models, as early as 2008 [87] there were critics who noted that access control in healthcare systems is sufficiently complex to justify situation-based decisions, with the classical concept of roles and attributes oversimplifying the issue. When we conducted our first experiments in 2018 in this area [106], we also found that neither ABAC nor RBAC as such is sufficient to meet the needs of practitioners and clinical applications, because even though roles are important elements of security, they cannot cover every situation without specific, contextual information.

Although there are platforms, such as the popular SMART project [28], which offer a solution in the form of a full OAuth2 integration into their FHIR database, the use of such frameworks usually comes at the expense of a certain degree of freedom in the choice of health infrastructure components. There have been several attempts to define hybrid solutions both in healthcare [88] and more generally in multi-modal, heterogeneous environments [20]. A key concept of the domain is the requirement to control access not only to entire documents, but also to specific fields and attributes in documents. The proposed architecture by Rezaeibagha, F. et al [92] is specifically designed to move sensitive data from a secure private cloud to a public one, while maintaining security.

In 2016, Pusselwalage H. S. G. et al. [89] published their approach for an ABAC methodology that bases its policies not only on the attributes of the data but also on the attributes of the user, treating the different levels of access and the classical roles in healthcare as attributes. They combined the two models to some extent, while also highlighting special cases such as unregistered users or registered users without a specific role. In 2018 Joshi M. et al. [62] used a similar approach with roles treated as attributes, but instead of granting full access, their solution also transformed the requested data to match the requester's access level. The developers of the SOCIAL platform [94] also discussed some interesting ideas about treating the requesting device as an important component of ABAC with a combination of the user attributes.

During our review we also found some studies that appeared to combine elements of the RBAC and ABAC models without clearly categorizing their methodology as a hybrid. The developers of the [90] H-Plane Framework, which follows the terminology of the ABAC model, also apply several attributes in a way that is almost identical to some aspects of RBAC. In their publication they also pointed out the importance of the IoT in this domain. In 2019, Alnefaie, S. et al. [16] after reviewing the possible alternatives for access control they thought ABAC was much better suited to the needs of healthcare in combination with edge computing, but also suggested modifying the infrastructure of the methodology to bring the point of evaluation closer to the edge and place more emphasis on the identity of the IoT device itself. Tasali Q. et al. [108] extended this concept by covering not only medical data, but also the authorization process for real-time communication between IoT devices.

The solution proposed by the developers of the mHealth application [85] is also quite similar to ours, the main differences being that its policy engine is deployed as part of the cloud services and the engine is an implementation of the NIST NGAC framework, with the evaluation process based on traversing a Neo4j graph database. The infrastructure and principle designed by Ray, I et al. [91] also have similar features, with policy enforcement based on the XACML format.

To summarize the state-of-the-art based on these sources:

- A modern solution should either extend the traditional access control ABAC model or develop a custom hybrid solution to meet the needs of the domain;
- Heterogeneous storage should be taken into account and the sensitive documents must be transformed before they enter the public cloud;
- The IoT raises brand new challenges. The security solution must be able to han-

dle the different capabilities and requirements of these tools when evaluating and converting the healthcare data.

It is clear that the approach of our research group is only one of many proposals that seek to resolve the security issue of EHR. Our goal was to combine the best ideas and elements of the domain - combining RBAC and ABAC, establishing the enforcement points as a middle layers between the various actors of the infrastructure - and also to improve and extend them, to provide support for every database and application that uses FHIR, and to provide users and developers with a trusted, verified solution to the security problem.

Azeez and Van der Vyver [21] collected the various approaches towards e-health security, and from their work, it is clear how isolated and partial the majority of these solutions are. My approach, instead of focusing on a smaller detail of the e-health data path, aims to be applicable at any given point of the infrastructure—in the cloud, between the cloud and the user, on the edge, in the user applications, etc.—to provide the best and most precise access control solution possible while having as minimal a latency impact on the data path as possible.

3.3 Thesis I/1: Formal Definitions and Requirements of Telemedicine Access Control

I explored the complex requirements of modern telemedicine applications in terms of access control. I defined a taxonomy to formalize the different types of access control policies and the TAPE requirements necessary to ensure that the implementation of the defined policies can guarantee a balance between data privacy compliance and responsiveness at any point in the telemedicine infrastructure.

Publications related to this thesis: [J1],[C6],[F9],[F10]

One of the starting points of my research in this area was to determine if it would be possible to develop a solution that could act as a dynamic, portable PEP, having the capability to be deployed to any point along a complex, heterogeneous data path, with the same tools, definitions, and configurations. I defined the basic requirements that this solution had to meet as the TAPE requirements, which are as follows:

- **Transparency**: The access control solution should have as little of an effect as feasible on the data path throughput and performance of telemedicine applications.

- **Adaptability**: To safeguard sensitive data, the field of telemedicine requires stringent, very specific policies. ABAC and RBAC alone are insufficient because interoperability and interchangeability requirements require a significantly more dynamic and refined approach. The developed solution must also facilitate the formulation and evaluation of the most specific requirements.

- **Portability**: The developed solution must be deployable anywhere in the infrastructure. Edge computing's greatest asset is its ability to provide functionality even when the cloud is unavailable. This also implies that it must be able to operate between the edge and the cloud, between the edge and the endpoints, in the cloud, and in certain scenarios, even on the endpoints if they have the required resources.

- **Efficiency**: Since many elements of the infrastructure lack sufficient memory and CPU capacity to execute the more complex transformations and data analysis, the developed implementation should spare them from more demanding operations.

I have defined the telemedicine access control principles based on the following taxonomy and requirements. The patient is the primary owner, the physician who wrote the document or assisted in its creation is the secondary, while other practitioners and relatives can have access to it to some extent. The system must also handle indirect access when the applicant, as a member of a group, tries to access the file. These respective types of access must be identified based on a combination of user roles, role groups and the attributes of the FHIR documents.

In some cases, contextual information is also required to determine the degree of access. For example, while a general practitioner should be able to access patient records at any time (logging the exact time and nature of such access), a nurse or assistant should not allowed to exceed the prescribed office hours. For some especially sensitive information, other contextual information such as the physical location of the requester, the ID of the device from which the request originates, should also be used in the evaluation, and expanding this set compared to a simple role definition is enough to justify a separate category.

A key requirement in the field of healthcare is that access does not mean full access to every element of the given document. In many cases it is strictly forbidden to grant access to such information from which a third party might be able to reconstruct very sensitive events and elements. For example, if one receives a list of a patient's medicines from a certain period of time, it is easy to infer vital information that would otherwise be prohibited for that particular third party. The evaluation process in a healthcare environment should be able to determine access at a very fine granularity, essentially at the field-by-field level, and to mark or even remove certain fields that should not be available at that security level. This is also the reason why the standard security solutions of several large cloud providers and databases fails, as they can only provide this functionality by including lambda functions, trigger functions, and the like.

The last requirement is also the most unique and difficult aspect of healthcare security. The break-the-glass case requires an access control model that provides immediate access to key patient information to ensure the receiving of the necessary, possibly life-saving care. This is essentially what happens in an emergency, when life-saving surgery is required and neither the patient nor the doctor recording and processing their health data is available to grant access. In a break-the-glass situation usually only a few records are required, but in that case it is important to use very complex transformations. Only vital information should be accessed, while every other element of the document must be either removed or encrypted. Without the effective implementation of break-the-glass, no healthcare security system can be used in real-life situations.

Based on this, I have defined the four categories of access control rules that follow:

- **Role Evaluation**: The policy has to determine whether based on the user's role or roles in the system, partial or full access should be provided;
- **Contextual Evaluation**: The policy has to determine whether the combination of the user's role, various attributes and contextual information form the basis for partial or full access;
- **Contextual Modification**: Aside from providing access, the policy should also transform the data, removing or altering specific fields;
- **Break-the-Glass**: A specific requirement of a healthcare application. In the case of an emergency, the policy should provide immediate access, while also encrypting or removing sensitive information.

With the formal definitions being:

- $\mathbb{F} := \{f_1, ..., f_k\}$ marks the telemedicine record in question, where each f_x is a valid key-value pair of the record.
  For example:
  $\mathbb{F} := \{('subject','PAT/1'), ('systolic\_bloodpress', 120), ...\}$
- $\mathbb{UR} := \{r_1, ..., r_l\}$ where $\mathbb{UR} \subseteq \mathbb{F}$ is a subset containing the key-value pairs describing various primary or secondary owners of the record
  For example:
  $\mathbb{UR} := \{('subject','PAT/1'), ('practitioner','PR/A013'), ...\}$
- $\mathbb{EX} := \{e_1, ..., e_m\}$ marks the external context of the system at the time of the policy evaluation as key-value pairs

For example:
$\mathbb{EX} := \{('datetime','2020-09-12T12:20:33'), ('ip\_addr','223.134.22.1'), ...\}$

- $\mathbb{CX} := \{c_1, ..., c_k\}$ is a set of conditional functions, which take an atomic value as an argument and transform it to a boolean value. Each function is represented as an (op, val) pair where op is a conditional operation, $op \in \{<, >, \leq, \geq, =\}$ and $c_i(n) := n\ op_i\ val_i$
  For example:
  $c_1 := (>, 12), x := 5 \implies c_1(x) := 5 > 12 \implies c_1(x) :=$ false
  $c_2 := (=, 'bloodpressure'), x := 'bodyweight' \implies$
  $c_2(x) :=$ 'bodyweight' = 'bloodpressure' $\implies c_2(x) :=$ false
- **P**(n) is a function describing a policy to be enforced by a PEP engine, where $f_x \in \mathbb{F}$ and $\mathbf{P}(f_x) = Allow|Modify|Deny$ produces the decision regarding the evaluated key and $\mathbf{P}(\mathbb{F}) := \{\mathbf{P}(f_1), ..., \mathbf{P}(f_k)\}$

Formal Definition of the Role Evaluation Policy

P(n) describes a Role Evaluation policy, if $\mathbb{UR} \neq \emptyset$ and for a given user identifier $\exists key(key, id) \in \mathbb{UR}$, then $\mathbf{P}(n) := \forall f_i \in \mathbb{F}\ \mathbf{P}(f_i) := Allow$, else $\mathbf{P}(n) := \forall f_i \in \mathbb{F}\ \mathbf{P}(f_i) := Deny$

Formal Definition of the Contextual Evaluation Policy

P(n) describes a Contextual Evaluation policy if $\mathbb{G} := \mathbb{F} \cup \mathbb{EX},\ \mathbb{CE} := \{ce_1, ..., ce_x\}$ is a set of contextual conditions where $0 \leq i \leq x,\ ce_i := (key_i, c_i),\ c_i \in \mathbb{CX}\ and\ \exists x : (key_i, value_i) \in \mathbb{G}$ and $\mathbf{P}(n) := \forall f_i \in \mathbb{F}\ \mathbf{P}(f_i) := Allow,$ if $\forall x : (key_i, c_i) \in \mathbb{CE} : \exists(key_i, value_i) \in \mathbb{G}$ and $c_i(value_i) := true,$ else $\forall f_i \in \mathbb{F}\ \mathbf{P}(f_i) := Deny$

Formal Definition of the Contextual Modification Policy

P(n) describes a Contextual Modification policy if similarly to the Contextual Evaluation policy, $\mathbb{G} := \mathbb{F} \cup \mathbb{EX},\ \mathbb{CE} := \{ce_1, ..., ce_x\}$ is a set of contextual conditions where $0 \leq i \leq x,\ ce_i := (key_i, c_i),\ c_i \in \mathbb{CX}\ and\ \exists x :\ (key_i, value_i) \in \mathbb{G}$ but there is also a **FX** mapping, which **FX** $: \mathbb{CE} \implies \mathbb{F}' \subseteq \mathbb{F}$, with $\cap_{0 \leq i \leq x}\ \mathbf{FX}(ce_i) := \emptyset$. $If 0 \leq i \leq x\ ce_x(g_x) := false$ then
$\forall f'_i \in \mathbf{FX}(ce_i)\ \mathbf{P}(f'_i) := Deny,\ else\ \forall f'_i \in \mathbf{FX}(ce_i)\ \mathbf{P}(f'_i) := Allow$

Formal Definition of the Break-the-Glass Policy

P(n) describes a Break-the-Glass policy if **P**(n) satisfies the requirements of the Contextual Modification with the further addition of a **TX** mapping, identifying the attributes which have to be encrypted or modified **TX** $: \mathbb{CE} \implies \mathbb{F}'' \subseteq \mathbb{F}$, with $\cap_{0 \leq i \leq x}$

$\mathbf{TX}(ce_i) := \emptyset$ and $\cup_{0 \leq i \leq x} \mathbf{FX}(ce_i) \cap \cup_{0 \leq i \leq x} \mathbf{TX}(ce_i) := \emptyset. \ If 0 \leq i \leq x \ ce_x(g_x) := false$ then $\forall f_i'' \in \mathbf{FX}(ce_i) \ \mathbf{P}(f_i'') := Deny, \ else \ \forall f_i'' \in \mathbf{FX}(ce_i) \ \mathbf{P}(f_i'') := Modify$

3.4 Thesis I/2: Formal Definitions and Challenges of Access Control Beyond the Telemedicine Cloud

I proposed special categories of edge instances beyond the cloud that represent a special category of modern telemedicine infrastructure from the perspective of access management. I formally defined a categorization of these into storage and processing edges. I then discuss the increasingly prevalent smart device based peer-to-peer networks as an extreme case of edge solutions, and analyse their potential and stability to function as stand-alone edge networks.

Publications related to this thesis: [J1],[J2],[C5],[C7],[C8],[F10],[F11]

The transformation of the healthcare landscape is increasingly influenced by the expanding disciplines of edge and fog computing [19, 29, 35, 114]. The pervasiveness of e-health applications necessitates a shift from centralized to distributed computing paradigms.

Edge computing is the paradigm of bringing computation closer to the data source, which may be IoT devices, sensors, or other data-generating terminals. In contrast to the traditional cloud-centric paradigm, where data is sent to centralized data centers for processing, this transition involves sending data directly to the cloud. Edge computing addresses several challenges posed by bandwidth limitations, latency, and the shear volume of data generated by modern devices by bringing computation closer to the data source.

Fog computing, which frequently accompanies edge computing, seeks to introduce intelligence to the local level. Fog computing operates between edge devices and the central cloud, offering a more distributed approach to data, storage, and application services. While edge computing concentrates on immediate, real-time actions close to data sources, fog computing operates between these edge devices and the central cloud.

These computing paradigms provide numerous advantages in the context of e-health and telemedicine. They can facilitate real-time patient monitoring, improve the responsiveness of medical applications, and protect the privacy and security of patient data. Imagine a scenario in which the vital signs of a patient are monitored remotely. Instead of transmitting all data to a central server, preliminary analysis can be performed at the periphery, near the patient. Only pertinent information, possibly indicative of an anomaly, would be transmitted to the cloud or central server for further analysis. This not only reduces the burden on central resources, but also ensures timely intervention when necessary.

3.4.1 Processing Edge

The infrastructure component that I refer to as the processing edge represents a pivotal evolution in the architecture of edge computing. It is capable of imitating a variety of cloud-typical functions. Two of its most distinguishing characteristics are its advanced access control and robust data storage capabilities.

The processing edge functions as an intermediary element in the e-health infrastructure, ensuring that data is not only effectively stored but also accessible in a controlled manner. Its architecture is intended to span the distance between the cloud's vast storage and processing capabilities and the periphery devices' immediacy and responsiveness.

Consider a telemedicine application that requires a patient's data spanning multiple years for a comprehensive analysis. While real-time processing of imminent data from sensors and devices is possible at the periphery, historical data may reside in the cloud. With its cloud-like capabilities, the processing edge can seamlessly integrate these datasets, ensuring that the entire medical history of the patient is available for analysis without significant delays.

In addition, the enhanced access control mechanisms at the processing edge are crucial for guaranteeing data privacy and security, which are of paramount importance in the healthcare industry. It provides granular control over who can access the data, under what conditions, and for what purpose, thereby protecting the data path from unauthorized access and potential intrusions.

3.4.2 Caching Edge

The caching edge is a distinct portion of the edge in terms of storage and computational capacity. It may not possess the same computational prowess as the processing edge, but its strength rests in its enhanced storage capabilities, distinguishing it as a layer that is more sophisticated than basic edge endpoints but less complex than the processing edge.

In situations where continuous access to the cloud or processing edge is impractical or momentarily lost, the true value of the caching edge becomes apparent. Consider a scenario in which remote healthcare interventions are carried out in regions with intermittent or no connectivity. Wearable devices or other medical equipment may not always have the luxury of transmitting vital patient data in real-time to centralized servers or even the processing periphery, despite the fact that they generate such data.

In these instances, the caching edge functions as a safe haven for these data. By providing extensive offline storage, it ensures that no data is lost even if connectivity is compromised. Once the devices regain access to more robust networks or the processing edge, the stored data can be synchronized to ensure that every piece of data is recorded and available for subsequent analysis.

Moreover, when immediate processing is not essential but data retention is, the offline storage of the caching edge becomes invaluable. Real-time analysis may not be required, for instance, if a healthcare provider is monitoring non-critical parameters of a patient over an extended period. However, in order to provide comprehensive care, it is necessary to store this data for days or even weeks. With offline storage capabilities, the caching edge becomes the primary repository for such data, preserving it until it is processed or analyzed.

Additionally, the offline capabilities of the caching edge play a crucial role in disaster recovery scenarios. In the event of unanticipated disruptions or system malfunctions, data may be compromised. Nonetheless, data consistency is maintained because the caching edge functions as a temporary storage space. Once systems are restored, cached data can be retrieved and integrated back into the primary flow, minimizing data loss and ensuring uninterrupted patient care.

With its enhanced offline storage capabilities, the caching edge serves as a safety net in the e-health infrastructure. Whether it's to ensure data retention in connectivity-challenged areas, function as a buffer during non-critical monitoring, or play a crucial role in calamity recovery, the caching edge is essential for bolstering the telemedicine data path.

3.4.3 Formal Definition

Let us denote the entire e-health cloud-based system as $\mathcal{E}$. This system is a tuple comprising various components, such as devices, edge nodes, storage units, processing units, and communication links.

$$\mathcal{E} = (D, N, S, P, L)$$

Where:

- D represents the set of all devices or endpoints in the system.

$$D = \{d_1, d_2, \ldots, d_n\}$$

- N represents the set of all edge nodes in the system.

$$N = \{n_1, n_2, \ldots, n_m\}$$

- S represents the storage units available at different layers, including the caching and processing edge.
$$S = \{s_1, s_2, \ldots, s_o\}$$

- P represents the set of processing units or capabilities at different layers.
$$P = \{p_1, p_2, \ldots, p_p\}$$

- L represents the set of communication links connecting various components.
$$L = \{l_1, l_2, \ldots, l_q\}$$

The caching edge, denoted as $\mathcal{C}$, is a subset of the e-health system $\mathcal{E}$ focusing primarily on storage capabilities, especially during offline scenarios.

$$\mathcal{C} = (D_c, N_c, S_c, P_c, L_c)$$

Where:

- $D_c \subseteq D$ is the set of devices that interact directly with the caching edge.
- $N_c \subseteq N$ is the set of edge nodes specifically designated for caching purposes.
- $S_c \subseteq S$ is the predominant component, representing the enhanced storage capabilities of the caching edge.
- $P_c \subseteq P$ signifies limited processing capabilities, mostly to manage the storage and retrieval of data.
- $L_c \subseteq L$ represents communication links pertinent to the caching edge.

The processing edge, denoted as $\mathcal{P}_{\mathcal{E}}$, is another subset of $\mathcal{E}$, characterized by its processing prowess and ability to mimic cloud functionalities.

$$\mathcal{P}_{\mathcal{E}} = (D_p, N_p, S_p, P_p, L_p)$$

Where:

- $D_p \subseteq D$ is the set of devices directly interacting with the processing edge.
- $N_p \subseteq N$ is the set of edge nodes designated for processing tasks.
- $S_p \subseteq S$ represents storage units at the processing edge, more advanced than the caching edge but not as vast as the cloud.

- $P_p \subseteq P$ is the predominant component, symbolizing the significant processing capabilities of the processing edge.
- $L_p \subseteq L$ denotes communication links associated with the processing edge.

1. **Storage Capabilities**: The cloud has the highest storage capability, followed by the processing edge and then the caching edge.

$$S_{\text{cloud}} > S_{\mathcal{P}_{\mathcal{E}}} > S_{\mathcal{C}}$$

2. **Processing Power**: The cloud possesses the maximum processing power, followed closely by the processing edge. The caching edge has the least processing capability.

$$P_{\text{cloud}} > P_{\mathcal{P}_{\mathcal{E}}} > P_{\mathcal{C}}$$

3. **Access Control Capabilities**: The cloud and the processing edge possess advanced access control mechanisms, allowing for granular permissions and access restrictions. The caching edge has limited access control capabilities and can only grant or deny access based on its current storage state.

$$AC_{\text{cloud}} = AC_{\mathcal{P}_{\mathcal{E}}} > AC_{\mathcal{C}}$$

   Where AC denotes access control capabilities.

4. **Data Retrieval Rule**: If a device queries data on the e-health edge:

   (a) The caching edge will first check its storage. If the data is available, it will provide it; if not, and it cannot evaluate the access, it will deny the request.

   (b) If the data is not available at the caching edge, or if the caching edge cannot evaluate access rights, the request is forwarded to the processing edge.

   (c) The processing edge will check its storage and access control mechanisms. If the data is available and access is permitted, it will provide the data; otherwise, the request is directed to the cloud.

Consider a scenario where a healthcare provider wants to retrieve a patient's medical history. The request is initiated from a device d_2. The medical history comprises recent data (stored at the caching edge), intermediate data (stored at the processing edge), and older data (stored in the cloud).

1. The device d_2 sends a query to the nearest node n_3 of the caching edge.

2. n_3 checks its storage. It finds the recent data but doesn't have the entire medical history. Let's hypothesize that this particular request requires more complex access control solutions than simple role or contextual evalution. In this case, given its limited access control abilities, the edge node sends the available recent data to d_2 and forwards the query to the processing edge node n_4 for the documents requiring more complex evaluations.

3. n_4 at the processing edge checks its storage. It locates the intermediate data and, after evaluating its advanced access control mechanisms, determines that d_2 has permission to access the data. It sends the intermediate data to d_2.

4. For the older data, n_4 forwards the request to the cloud. The cloud evaluates the request, accesses the older data, and after verifying the access rights, sends the data to d_2.

In this example, the device d_2 successfully retrieves the entire medical history of the patient, navigating through the different layers of the e-health cloud-based system. The process showcases the layered storage, processing, and access control mechanisms in place across the caching edge, processing edge, and the cloud.

3.4.4 Smartphone-based Peer-to-Peer Edge

Distributed computing over the edge as part of various smart systems is becoming a popular research topic [47]. Research into algorithms that are suitable to such environments often involves actual deployments, because realistic conditions are non-trivial to model, yet they are crucial for finding an optimally efficient and robust solution. Still, this severely limits the possibilities of exploratory research.

One important domain is smartphone applications that can form a part of many smart systems such as smart city or e-health solutions [116]. In this domain, it is important to fully understand the capabilities and limitations of the devices and their network access as well. This includes battery charging patterns, network availability (churn) and network attributes (for example, NAT type).

Our team started to develop the smartphone app Stunner in 2013 to collect data concerning the NAT properties of smartphones using the STUN protocol [23], as well as many other attributes such as battery level and network availability. Since then, we have collected a large trace involving millions of individual measurements. Recently, we also updated the application to collect data concerning direct peer-to-peer capabilities based on a basic WebRTC implementation.

There have been many data collection campaigns targeting smartphones. This included the famous Mobile Data Challenge (MDC) [67], which aimed to collect large amounts of data from smartphones for various research studies, including sensory data, cell towers, calls, etc. and ran between 2009 and 2011, resulting in the largest and most widely known dataset yet. After this, the most prominent project to achieve similar results was the Device Analyzer Experiment. started in 2011 by the University of Cambridge, aiming to not only record similar attributes to the MDC, but also system-level information such as phone types, OS versions, energy and charging [31, 115]. This trace has been used, for example, to determine the most energy consuming Android APIs [70] or to reconstruct the states of battery levels on the monitored smartphones [48]. Our dataset is unique in that, apart from being five years long, it contains all the necessary attributes to simulate *decentralized* applications.

Another set of projects are concerned with measuring the network (e.g., detecting NAT boxes) as opposed to collecting a full trace from the devices, which is our main goal. For instance, in 2014 a study was initiated to analyze the deployment rate of carrier-grade NATs that can hide entire areas behind a single public IP address [93]. The measurement was based on **Netalyzr**, as well as on crawls of BitTorrent DHT tables to detect possible leaked internal addresses due to hairpin NAT traversal. In another study across Europe, an application called **NAT Revelio** was developed [75]. Yet another data collection campaign attempted to collect traceroute sessions from smartphones using the custom **TraceboxAndroid** application [112]. The application detects the exact number of middleboxes and NAT translations encountered between the device and a specified test target. In a similar two-week campaign, the **Netpicular** application was deployed [117]. Also, a mobile application called **Mobil Tracebox** was deployed to carry out traceroute measurements [125]. This campaign ran for an entire year. A summary of these NAT studies can be found in Table 3.1.

While our NAT measurements were simply based on STUN server feedback [74], thus underestimating the complexity of the network, our P2P measurements indicated that our NAT type data is a good basis for predicting connection success, thus measuring the capabilities of a peer-to-peer network.

The results of our research team was fourfold:

1. our application Stunner had been collecting data for a much longer time than any of these applications, which allowed us to observe historic trends;

2. with an updated version of the application, we measured direct P2P connections allowing us to collect NAT traversal statistics;

3. we collected a wide range of properties simultaneously, including NAT type, battery level, network availability, and so on, to be able to fully model decentralized protocols; and

4. we made our trace publicly available at `http://www.inf.u-szeged.hu/stunner`.

Table 3.1: *Comparison between various NAT measurement campaigns*

Source	Collected Attributes	Length	Public	Tools
[93]	local, external and public IP addresses	2014-2016	No	Netalyzr
[75]	external IP, mapped port, traceroute results, UPnP query results	2016 May and August	No	NAT Revelio
[125]	traceroute results	2016 Feb - 2017 Feb	No	Mobile Tracebox
[117]	traceroute results, number of detected middleboxes	2011 Jan., 2 weeks	No	Netpiculet
[112]	traceroute results, number of detected middleboxes	2014 May - Sep	No	TraceboxAndroid

Data collection methodology

The functionality of our Android app Stunner was to provide the user with information about the current network environment of the phone: private and public IP, NAT type, MAC address, and some other network related details [23]. At the same time, the app collects data about the phone and logs it to our servers. The app was launched in April 2014, when it was simply made public without much advertising. Since then, at any point in time we had a user base of a few hundred to a few thousand users, and over 40 million measurements have been collected from all over the world.

In the original version measurements were triggered either by the user (when the app is used) or by specific events that signal the change of some of the properties we measure: battery charging status, network availability. There was periodic measurement as well every 10 minutes, if no other events occurred.

The version used for the peer-to-peer stability measurements was completely redesigned. This was necessary because Android has become very hostile to background processes when the phone is not on a charger, in an effort to save energy. For this reason, we collected data only when the phone is on a charger. This, however, was not a real issue, because for decentralized applications these are the most useful intervals, when it is much cheaper to communicate and to perform computing tasks in the background. Android event handlers have also became more restricted, so we could use them only under limited circumstances or on early Androids. The events raised by connecting to a charger or a network can still be caught by the Android job scheduler, but the timing of these events was not very reliable.

For this reason, instead of relying on event handlers, the new version checked the state of the phone every minute, and if there was a change in any important locally available networking parameter or in charging availability, it performed a full measurement. A measurement was still triggered if the user explicitly requests one, and it was also triggered by an incoming P2P measurement request. Also, if there was no measurement for at least 10 minutes, a full measurement is performed.

P2P connection measurements are also a new feature in the latest version that are performed every time a measurement is carried out. They are based on the WebRTC protocol [15], with Firebase as a signaling server [81], and a STUN server [74]. We built and measured only direct connections, the TURN protocol for relaying was not used. Every node that was online (had network access and was on a charger) attempted to connect to a peer. To do this, the node sent a request to the Firebase server after collecting its own network data. The server attempted to find a random online peer and managed the information exchange using the Session Description Protocol (SDP) to help create a two-way P2P connection over UDP. If the two-way channel was successfully opened then a tiny data massage is exchanged. The channel was always closed at the end of the measurement. One connection was allowed at a time, every additional offer was rejected by default.

Measurements

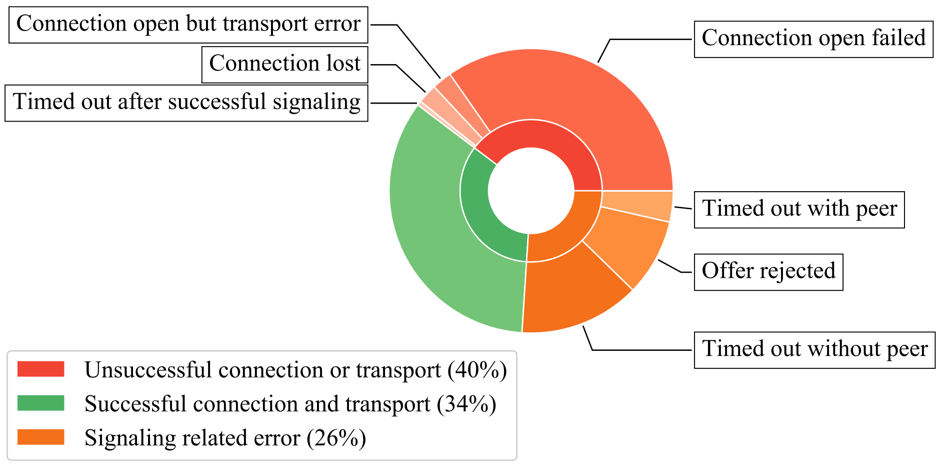


Figure 3.3: *Proportions of the possible outcomes of P2P connection attempts.*

Figure 3.3 shows the proportions of the outcomes of 63184 P2P connection attempts. Out of all the attempts, 34% was completed successfully. While analyzing the results, we attempted to investigate the various reasons why some of the sessions

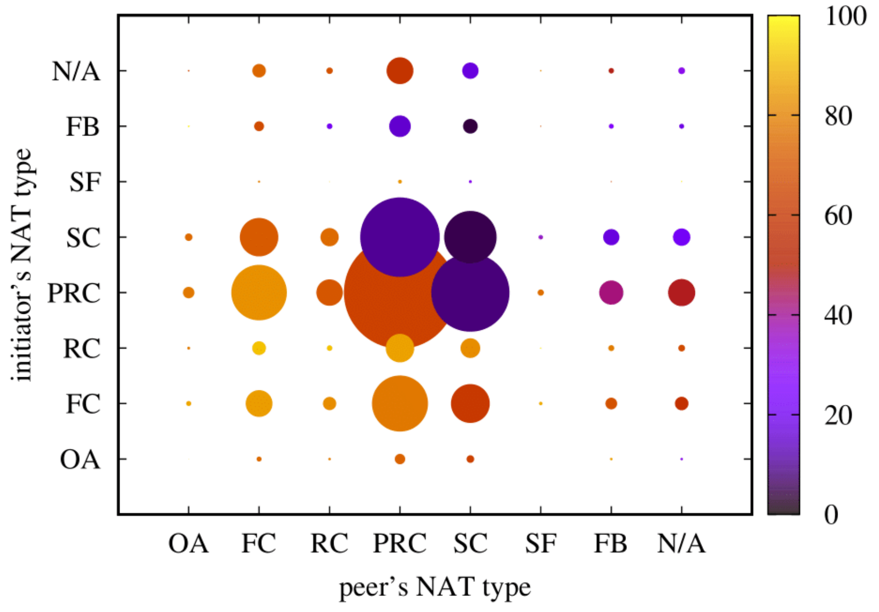


Figure 3.4: *Statistics over successful connections as a function of NAT type. The area of a disk is proportional to its observed frequency, the color signifies the success rate. Examined NAT types: OA - Open Access, FC - Full Cone, RC - Restricted Cone, PRC - Port Restricted Cone, SC - Symmetric Cone, SF - Symmetric UDP Firewall, FB - Firewall blocks, N/A-missing type*

failed. First, *signaling related error* means that the SDP data exchange via the signaling server failed. This can happen, if the server contacts a possible peer but the peer replies with a reject message (offer rejected), or it does not reply in time (timed out with peer), or we cannot see proof in the trace that any peer was actually contacted (timed out without peer). Note that a peer rejects a connection if it has an ongoing connection attempt of its own.

If the signaling phase succeeds, we have a pair of nodes ready to connect. The most frequent error in these scenarios is failing to open the channel, most likely due to incompatible NAT types. After the channel is open, transporting the test message is still not guaranteed to succeed (transport error). Participant nodes may disconnect with an open connection (connection lost). In some rare cases a timeout also occurred after successful signaling, that is, the WebRTC call did not return in time.

Figure 3.4 shows statistics over successful connections as a function of NAT type. The signaling related errors are not included here. Note that NAT type discovery is an independent process executed in parallel with the P2P connection test. Therefore, there are some cases where the NAT type information is missing but the signaling process is completed nevertheless.

The dynamics of the NAT distribution over the years are shown in Figure 3.5

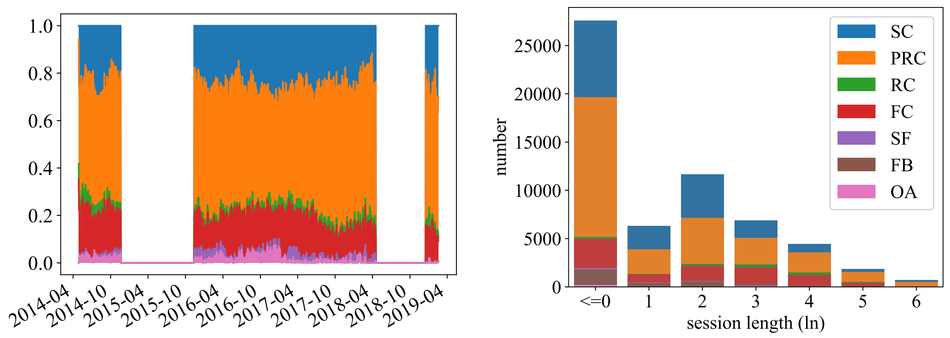


Figure 3.5: *(1) NAT distribution per day over 5 years (2) Session length distribution*

(left). The distribution is based on continuous sessions of online users. These continuous sessions of homogeneous network conditions were determined based on the measurement records. A session has a start time, a duration, and a NAT type. The distribution is calculated based on the number of aggregated milliseconds of session durations falling on the given day. The distribution of online time per day is near 8% almost every time. Recall, that here the online state is meant to imply that the phone is on a charger.

The plot has gaps because in 2015 the data collector server was down, when the project was temporarily neglected. In addition, the first version of our P2P connection measurement implementation caused lots of downtime in 2018. Also, some of the STUN servers that were initially wired in to the clients disappeared over the years. As a result, the *Firewall blocked* NAT type is not reliable, so we exclude that category from the figure. Note that the distribution is surprisingly stable over the years.

The session length distributions are presented in Figure 3.5 (right). Session length is in minutes, the bins for the histogram are defined on a logarithmic scale. Sessions shorter than one minute are not always measured accurately due to our one minute period of observation, so we grouped such sessions in one bin ($<= 0$).

Figure 3.6 contains stacked bar charts illustrating the distribution of different NAT types in the 6 continents and in the networks of the top 10 most represented providers in 4 different years. The most common NAT type is the Port Restricted Cone except in Africa where the Symmetric Cone has a relatively larger share. According to the chart the rarest NAT type is Open Access everywhere. Interestingly, the NAT type distribution is very different among the different providers, unlike in the case of the distributions based on geographic location.

Discussion

The development and significance of distributed computation at the periphery, particularly in smartphone applications, cannot be denied. Our investigation via the

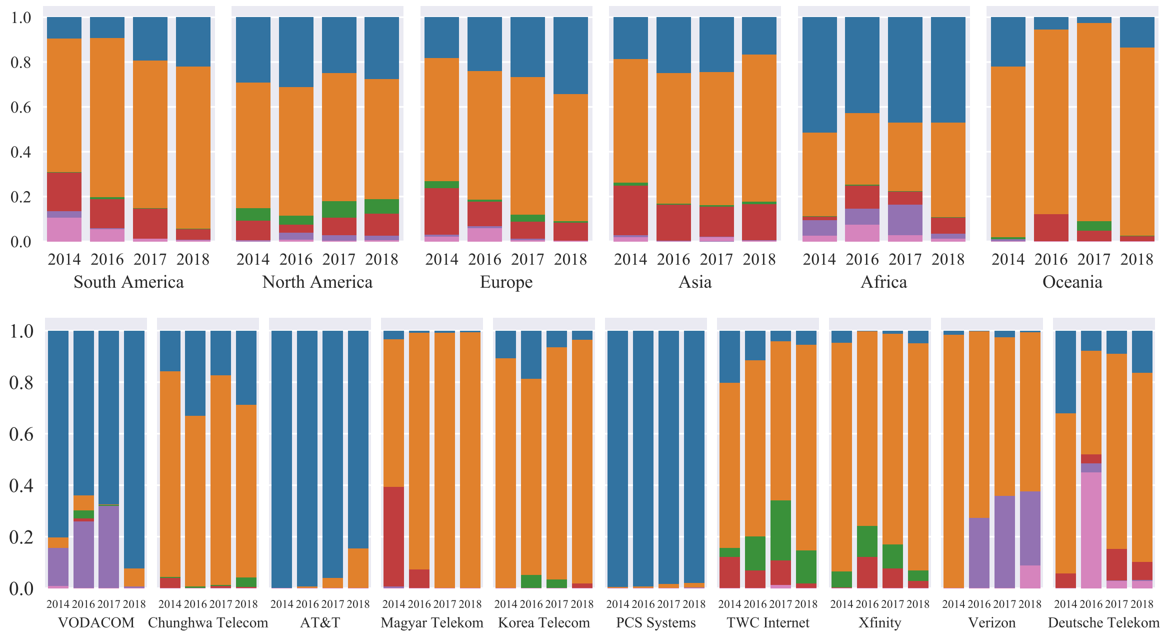


Figure 3.6: *NAT type distribution by continent in 4 different years (top) and NAT type distribution by the top 10 providers in 4 different years (bottom). Colors represent types as defined in Figure 3.5.*

Stunner application yielded an exhaustive data set regarding the NAT properties and direct peer-to-peer capabilities of mobile devices. With data spanning multiple years, we were able to observe historical trends, providing us with a unique perspective. The results of our P2P connection attempts, particularly the 34% success rate, demonstrate the complexities and difficulties inherent to such environments. Discrepancies in connection success, coupled with variations in NAT type distributions across regions and service providers, emphasize the complexities of devising robust and efficient algorithms for such networks.

Nevertheless, given the complexity and variance of our results, it is evident that deploying an access control solution on smartphone-based peer-to-peer networks is not simple. Diverse NAT types, diverse connection outcomes, and regional variations in network characteristics necessitate a more nuanced approach. Despite the fact that our research has provided a firm foundation and comprehension of the terrain, additional exploration and possibly simulations are essential. We can only assure the successful deployment and evaluation of an access control solution on smartphone-based peer-to-peer networks with this level of preparation.

3.5 Thesis I/3: Policy Enforcement Implementations for the Processing and the Caching Edge

I presented the implementation of the proposed access control solution, then I will set up test environments to represent both edge types, and sample rules to validate the effectiveness of the implementation under increasing data volumes. During the measurements, I examined the resource requirements of the nodes performing the evaluation, as well as the latencies measured for the data retrieval processes. I present the measured latencies which confirmed, that for reasonable amounts of data at the various edge types my policy categories met the requirements established in the previous theses.

Publications related to this thesis: [J1],[J2]

After defining the requirements and investigating the environments in which the solution had to be implemented, the next step was to construct test configurations, model the various edge types, and evaluate the effectiveness and usability of the proposed framework using a set of test policies.

The goal of my research was to create a solution that is able to support several different storage providers and, at the same time, make the exact implementation of access control transparent to the end users. While the use of such heterogeneous backends is recommended in various use cases, the field of telemedicine is a prime example of how the strengths of this model can be brought to bear. To test the concept, we chose a promising new solution called Open Policy Agent [7], available in Go and WebAssembly, which could play every role in the enforcement process (PEP, PDP, PIP, and PAP). In its standalone form, OPA is able to act as an expansion of the processing edge, while its WebAssembly runtime has the opportunity to run completely offline and be integrated into the applications themselves or the caching edge. OPA also permits us to store the information necessary for decision-making in JSON format and define the various policies in its own scripting language, Rego, which can later be accessed via a well-defined REST interface with an HTTP POST request containing the contextual information to be filtered or evaluated (in our case, the medical records).

3.5.1 Policy Test Set

Using OPA, I defined a test set of policies based on the taxonomy introduced in Chapter 3.3. I created two policies for each of the four categories - one simpler and one more complex, the latter containing more operations or more resource-intensive operations, or both. All algorithms run on multiple Observations received as part of the input, but only returned those in their original or modified state which were allowed by the policy.

The two policies of the Role Evaluation category included in the Appendix in Figures A.1 and A.2 both focus on the identity of the practitioner. However, while *role_simple* only checks to see that the specified role identifier matches the practitioner's identifier in the document, *role_complex* checks the care teams responsible for the patient and only grants access if the requester is a member of those teams.

The main difference between the policies of the Contextual Evaluation category, shown in algorithms A.3 and **??**, is the nature of the contextual attribute.

In *context_simple* we check a high-level attribute, the status of the Observation and an external attribute, the hour. Here *context_complex* requires the PEP iterating through the component array of each Observation, finding each medical value based on the defined LOINC identification code, and checking to see if the exact value is greater than the threshold.

A key aspect to be evaluated was the efficiency of the PEP when iterating and handling arrays, since in the current version of the OPA, the developers noted in the official documentation [6] that the performance of such an evaluation engine is the most efficient when it works with objects, and weakest when it must iterate through non-indexed arrays.

The majority of the documents in the FHIR standard use the array structure very often, and if the performance of array operations is significantly worse than in any other case, this would provide a strong argument against adapting our concept. The modification operation in a PEP node is very demanding in itself, since the engine treats every variable as a constant due to their non-imperative behavior. For this reason, if we need to modify or redefine a particular field, we must first create a copy of the original object without the value, then create a new value, and finally create the response by adding the new value to the filtered object copy. The Contextual Modification policies are shown in algorithms A.4 and A.5. Here *modif_simple* simply removes the patient data from the Observation and the encapsulating shell, while *modif_complex* must create a new component array for each Observation that does not contain urls pointing to sensitive patient documents.

The Break-the-Glass Policies shown in algorithms A.6 and A.7, are the most complex. These policies are expected to be fast, accurate, and effective, because they are most commonly used in emergency scenarios when a doctor or nurse needs to access limited patient information to provide the necessary care.

In these scenarios, the system must remove or encrypt everything else that goes beyond the most necessary attributes, and with two policies we tested how resource consumption varies when we wish to encrypt a single attribute that is deeply embedded in the document and requires filtering in *break_simple* and in *break_complex* when we wish to encrypt every identifier in the document that could later be used to identify users in the system.

3.5.2 Results from the Processing Edge

To capture the core characteristics and capabilities of the processing edge in its entirety, I meticulously designed and assembled a compact, multi-node test environment. This environment is a miniature version of a vast network, designed to imitate the distinctive characteristics of the processing edge without requiring extensive resources or infrastructure. At the core of this test environment is a WiFi-connected, well-structured local network. This network is purposefully tiny, functioning as a microcosm of the processing edge that is manageable and readily observable. The incorporation of a database into this network allows us to simulate the robust data storage capabilities intrinsic to the processing edge, allowing us to observe data accumulation, retrieval, and integration processes in real-time and on a manageable, yet representative scale.

The presence of an intermediary node is required because it depicts the ”intermediary element” that the processing edge exemplifies. This node functions as a bridge, transmitting data between the database (similar to a cloud storage or the cache of the edge) and the user application, simulating the seamless integration of current and historical data, as seen in applications such as telemedicine. By running the user application on a separate node, it is possible to closely monitor the responsiveness and immediacy typical of peripheral devices in real-world situations. This configuration guarantees the user receives timely and accurate information, allowing us to evaluate the performance and efficacy of data retrieval and processing.

Our compact test environment aims to depict the delicate equilibrium maintained by the processing edge between the vastness of the cloud and the immediacy of end users. This equilibrium is reflected in the testbed’s design, which necessitates only periodic interactions with the cloud as opposed to a continuous connection. This design choice not only conserves resources but also mirrors the operational efficacy of the processing edge, which can manage immense quantities of data without being permanently connected to the cloud. By distinguishing agents for storage, processing, and access control within the testbed, we can hone in on the specifics of how each function operates. This division enables us to independently modify, optimize, and refine each component, ensuring that every aspect of the processing edge is finely refined.

In this environment, I chose a Squid proxy-like [95] implementation for the OPA, deploying it on a separate node, connecting it to the intermediary node and configured it to forward each request from the user applications towards the database, but upon receiving the response containing FHIR structures from the storage, send it to the PEP for filtering and (if necessary) transforming before forwarding it to the end user. This provides a necessary middle layer, unlike most solutions that existed in the domain when we conducted our research. In this way, policy enforcement can take place outside the cloud, which allows the use of a heterogeneous storage solution (provided that it uses the FHIR standard as the format of the stored documents), but it also relieves the burden on the end systems. This approach is not only better optimized in terms of efficiency and capacity, as the elements of the processing cloud are very likely to have the required capabilities, but it also ensures that sensitive information never reaches the end users without prior assessment and filtering. It should be added that with such a proxy, developers are also able to log in detail the various operations on the telemedicine records in order to comply with the GDPR.

The components of the test environment were the following:

- A desktop PC running Windows 10 on an AMD Ryzen 5 processor at 3.59 GHz speed and 16 GB DDR4 memory ran the client application on a Kingston SSDNow V300 SSD with 120 GB capacity and 450 MB/s reading speed, acting as the controller node of the environment;

- A laptop running Windows 10 on an Intel i5 processor at 2.49 GHz speed with 8 GB DDR4 and a 120 GB SSD with a reading speed of 423 MB/s memory hosted the MongoDB v3.2.1 [5]-based backend along with a lightweight REST API that handled the requests and query parameters;

- A secondary laptop with similar attributes hosted the Squid proxy written in NodeJS 10.14;

- An iMac running macOS Catalina with an Intel i5 processor at 2.9 GHz speed and 20 GB DDR3 memory hosted the OPA v0.23.2 runtime with a hard disk of size 1 TB and reading speed of 210 MB/s.

The database was loaded with over 500,000 different FHIR Observation documents, based on properties from the MIMIC3 database [61], involving 200 patients, 30 doctors and 12 nursing teams, all signed with a different time stamp between 2015 and 2020. The template and structure of these Observations were taken from one of our industry projects to simulate the size and complexity of healthcare documents in a real system. The measurements were performed by a monitoring host that issued the restarts and reinitializations of each element of the architecture between measurements. During the experiment, each component was configured so that its output was logged in separate files that were collected and evaluated by the monitor at the end of a round. Each rule ran on 8 different input sizes: 10, 20, 50, 100, 200, 500, 1000 and 2000 data sets. The PEP engine received exactly the same amount of data in each round, but of course the size of the output varied from case to case, depending on the selected policy and the content of the input.

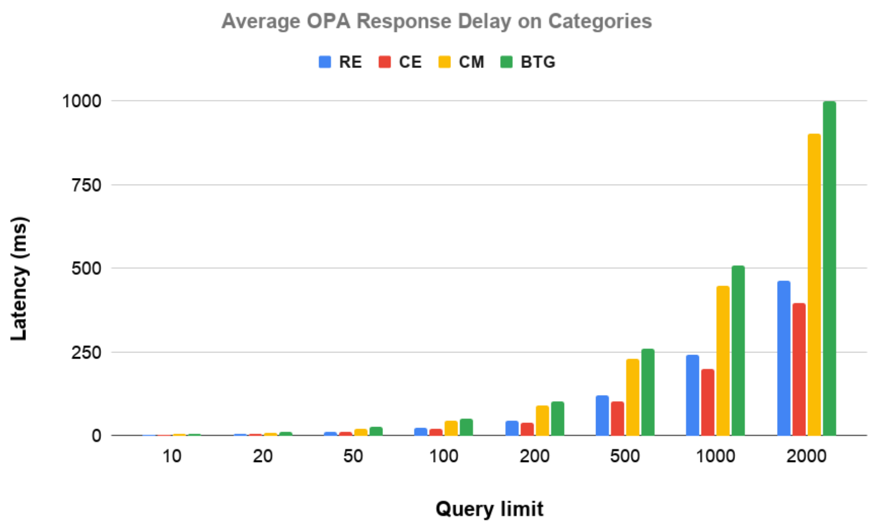


Figure 3.7: *Average Delay on PEP by Categories*

Performance between Categories

After evaluating the average performance of the various categories, we observed several interesting trends, compared to what we originally expected. The most notable of these is the relatively faster evaluation time of the Contextual Evaluation policies compared to the Role Evaluation category, as seen in Figure 3.7.

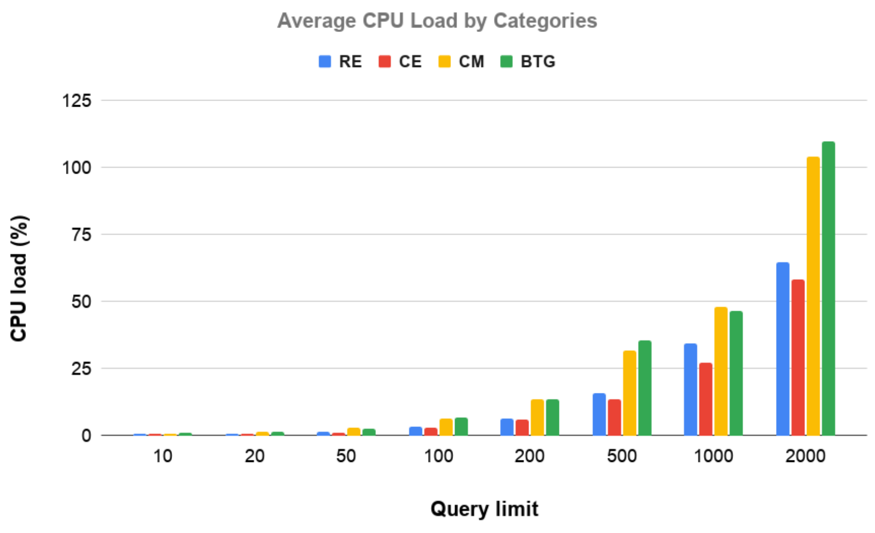


Figure 3.8: *Average CPU Load of OPA by Categories*

While the CE policies are more complex in nature, it seems that if the contextual information sets the result as true or false, the evaluation is significantly quicker than the cases when an internal examination of the input documents is required.

We observed a similar trend with the average CPU load of the categories, shown in Figure 3.8 with the Contextual Evaluation policies demanding slightly less percentage of the CPU time compared to the Role Evaluation policies, while the Break-the-Glass policies remain the most demanding ones. However, the overall difference between the first two and latter two categories is not as big as on the response delay. Another key observation is that while the size of the input was below 1000 documents (which is already an unrealistically large query size for a real-life application), not even the Break-the-Glass policies required more than 50% of the CPU.

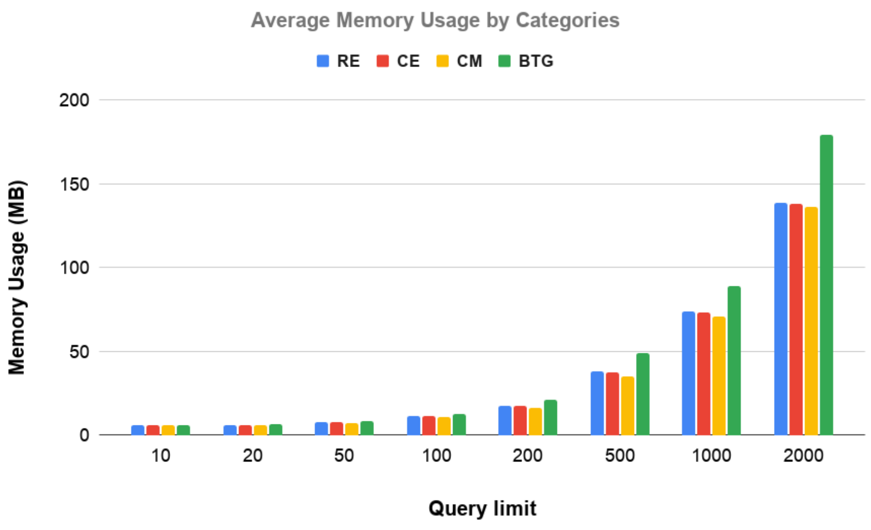


Figure 3.9: *Average Memory Usage of OPA by Categories*

The memory usage of the categories, shown in Figure 3.9 on the other hand, while still showing the trends of the previous figures and requiring only manageable amount of memory when evaluation smaller inputs (not even Break-the-Glass policies demanding more than 50-60 MB while the input size is around 50 documents), this demand shows a sudden jump after the input size reaches 1000 documents, with even the Role Evaluation policies requiring around 100 - 150 MB to evaluate inputs between sizes 1000 and 2000.

The measurements were taken with the PEP solution restarted and reinitialized between each measurement, since, as we have shown in our previous paper, OPA employs a very lazy approach towards garbage collecting, cleaning the memory only when it is required by the system or an especially large evaluation. This fact makes these sizes even more alarming for a real-life scenario, since the size of OPA in the memory can grow significantly during a series of evaluations.

Based on these results, while the CPU load and the response delay seem to be manageable requirements, the memory demand, combined with the experienced lazy garbage collecting process of OPA might requires a custom build or external process that manages and frees the memory after the evaluations are finished to optimize this aspect of the PEP nodes.

Performance in Categories

We ran each policy with each size at least 30-50 different times to collect the raw data for the statistics shown in the tables below, and these group the policies belonging to the same category. For each policy, we calculated from the collected data sets the mean value of CPU load, memory usage and response delay on the PEP (OPA) node.

Although the question of whether to use the same constant inputs for each evaluation, or use HTTP(S) requests that simulate a real-world application is a complex element for this phase of our research, we decided to use dynamic inputs to get more precise, realistic results.

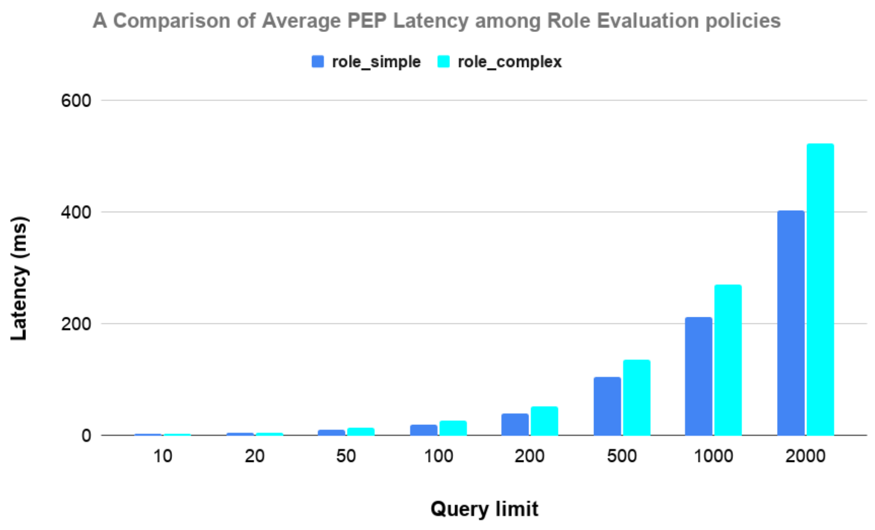


Figure 3.10: *Average PEP Latency in Role Evaluation category*

A comparison between Role Evaluation policies is shown in Figure 3.10. While, as we have assumed, the complexity of *role_complex* induces a higher latency, the total difference between the two policies is not very significant. Even with an input of 2000 records the PEP was able to filter out the restricted ones in half a second.

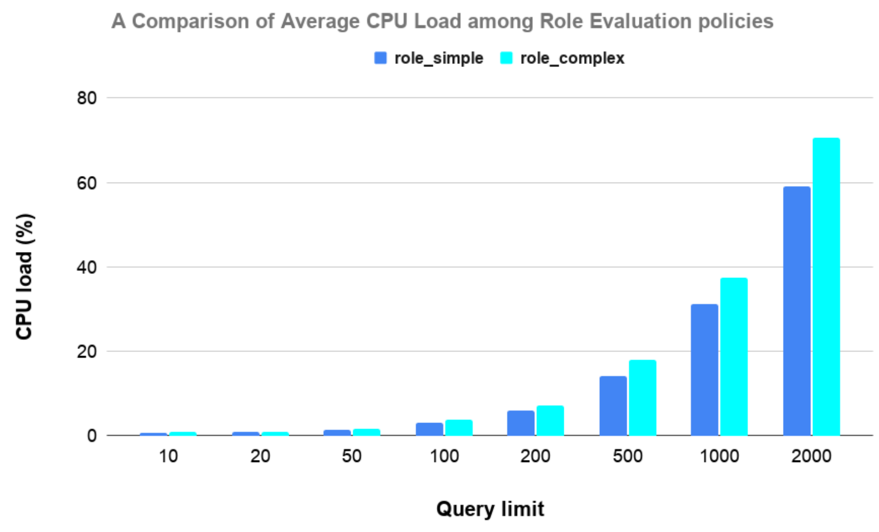


Figure 3.11: *Average CPU load in Role Evaluation category*

The effects on CPU and memory, as shown in figures 3.11 and 3.12, are somewhat more demanding - when 2000 documents are sent, 70% of the processor is required to evaluate the policy and about 140 MB of the memory - a clear indication of how costly it is to perform subqueries in isolated structures, such as the list of careteams and their members.

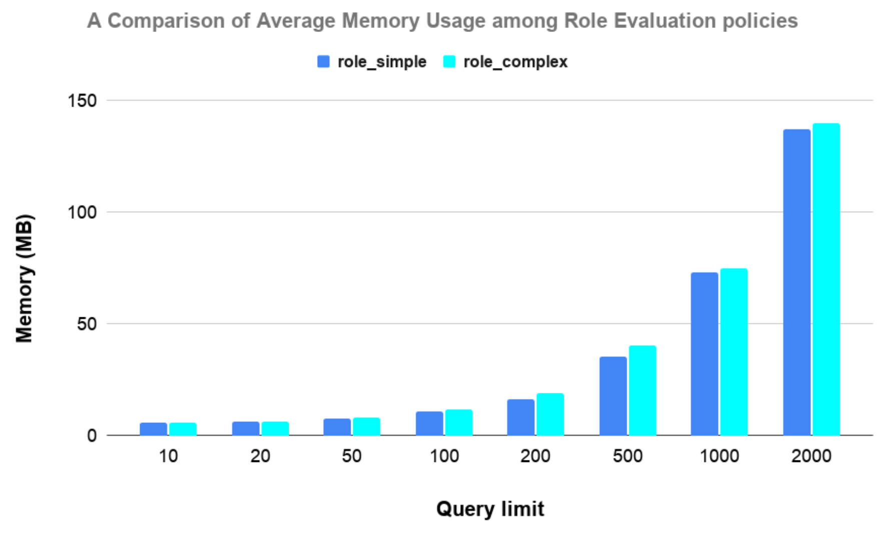


Figure 3.12: *Average Memory usage in Role Evaluation category*

Based on these results, it is evident that the proposed solution is capable of handling more complex Role Evaluation policies without difficulty, but it is advisable to store the teams, groups, institutions in indexed objects rather than in arrays.

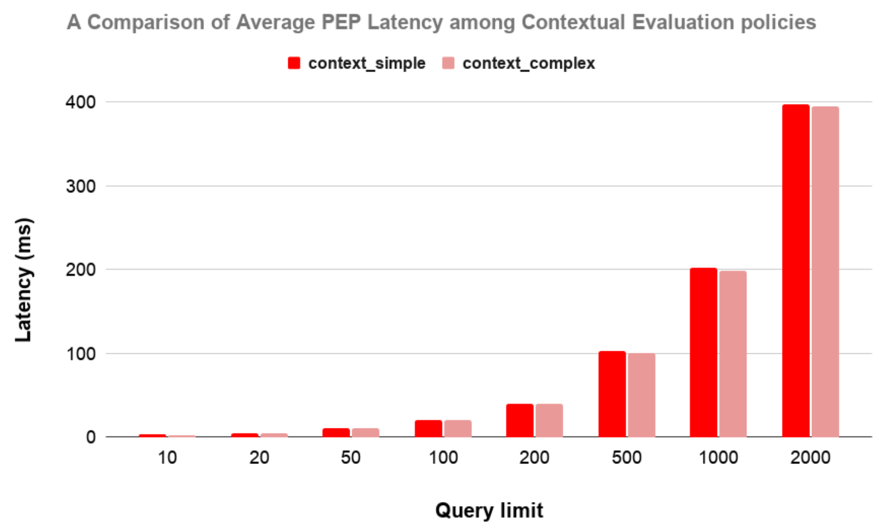


Figure 3.13: *Average PEP Latency in Contextual Evaluation category*

A comparison of the two Context Evaluation policies also produced some interesting results, which are presented in figures 3.13, 3.14 and 3.15. Our main objective here was to determine what kind of contextual evaluation is more demanding, and on the basis of the data it is clear that array-based evaluations are generally more complex, but in small evaluations they are actually cheaper than collecting and comparing external information such as dates.

This calls into question some notable architectural aspects of the infrastructure, such as whether this context information should be collected and forwarded by the proxy as part of the input data. Seeing that in some cases it is possible on a larger scale that the PEP deployment can handle traffic from end nodes in different time zones, it might be a good idea to omit such internal queries as a general design pattern of the policies.

Moreover, it is interesting to note that after some minor differences, besides inputs with more than 20 documents, the metric values start to converge, since after a certain point in the measurement set a significant portion of the documents is rejected during the evaluation of context1 due to their inactive status, thus the value is set false, before the engine starts evaluating the relational conditionals.

Our interpretation of these results can be summarized as follows: It is clear that policies with contextual evaluation may be as effective as simple Role Evaluation policies,

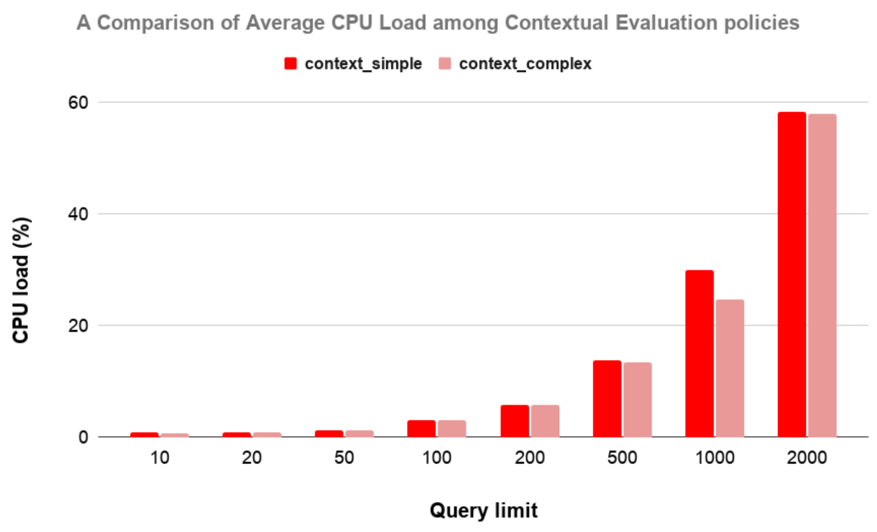


Figure 3.14: *Average CPU load in Contextual Evaluation category*

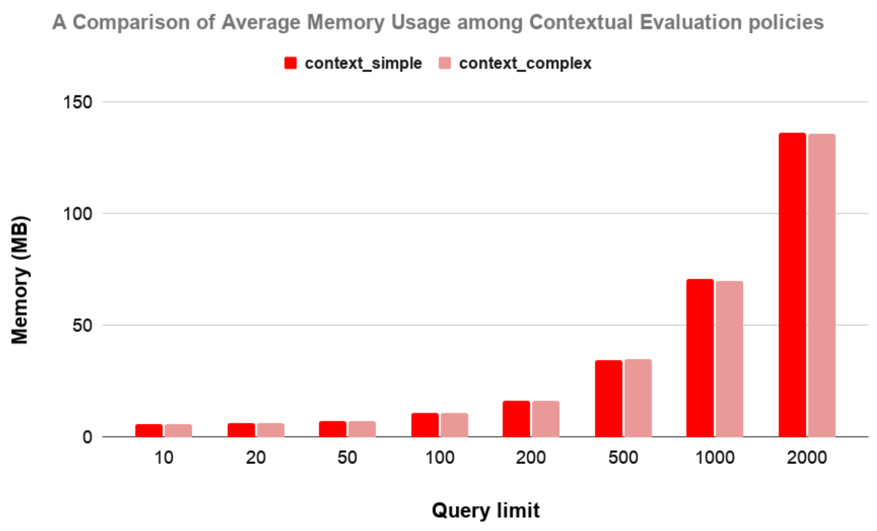


Figure 3.15: *Average Memory usage in Contextual Evaluation category*

but where possible, contextual information must be provided as part of the input, rather than being queried on the PEP node.

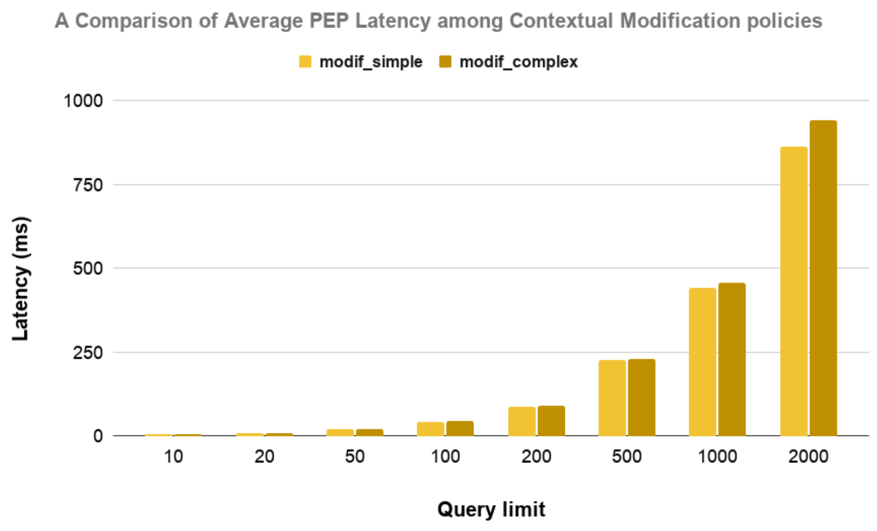


Figure 3.16: *Average PEP Latency in Contextual Modification category*

The results of the Contextual Modification policies are included in figures 3.16, 3.17 and 3.18. These results showcased another important aspect of the evaluation engine that could be one of the pillars of the design patterns for defining the policies. It was expected that *modif_complex* would be the more complex of the two.

Instead of this expected result, the measurements clearly indicate that neither is significantly more demanding. In some cases *modif_complex* consumes more CPU, and it has slightly more latency than *modif_simple* due to the demanding array copy and filter mechanisms, while *modif_simple* requires slightly more memory.

Modification policies are much more demanding than simple access evaluations. Nevertheless, they can be implemented effectively if we take into account the increased costs. We also wish to investigate the possible patterns and anti-patterns in order to write more effective policies for this type.

Based on the results of our previous evaluations, we assume that Break-the-Glass policies will be the most resource-intensive portion of our evaluation set, and the results (see figures 3.19, 3.20 and 3.21) were as we thought they would be, but again with some minor differences from our original expectations.

While the CPU load and memory usage of the two policies are almost identical, it actually takes a little longer to evaluate *break_simple* than *break_complex*, although based on the sheer number of operations (especially the number of encryption operations) in *break_complex*, it should have been a much more expensive policy compared

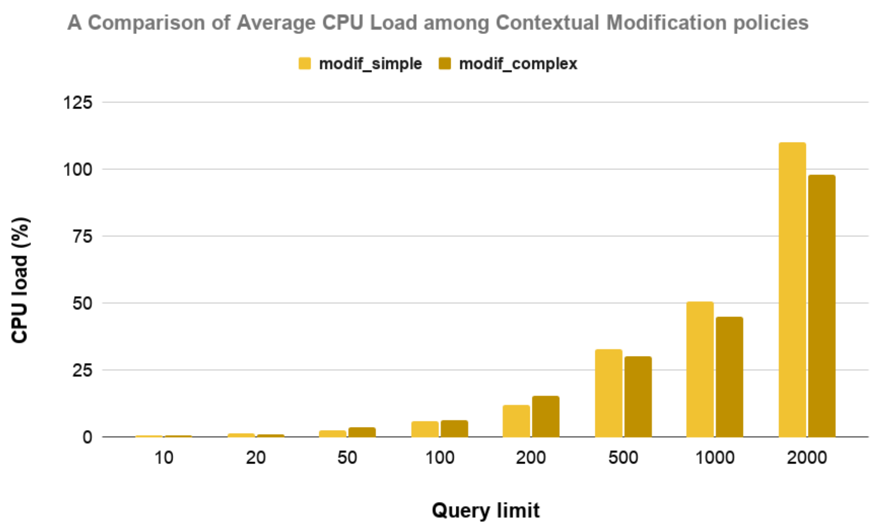


Figure 3.17: *Average CPU load in Contextual Modification category*

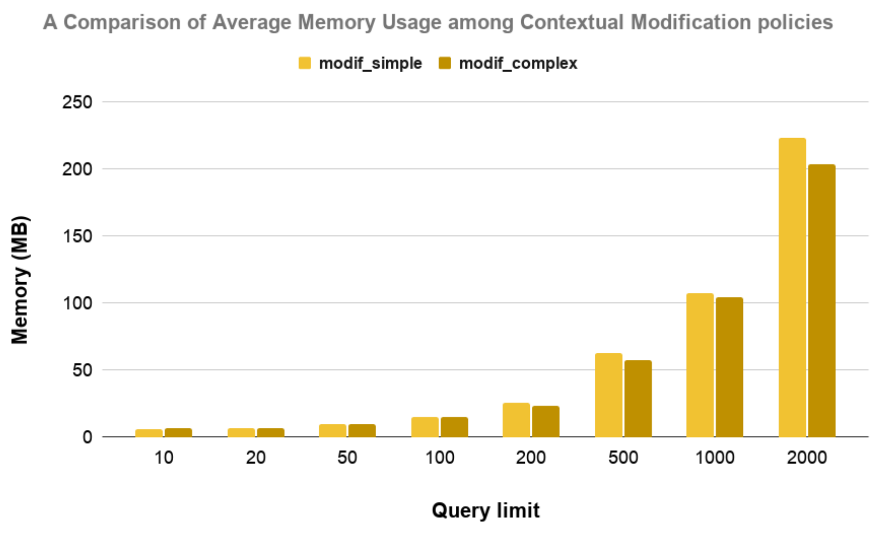


Figure 3.18: *Average Memory usage in Contextual Modification category*

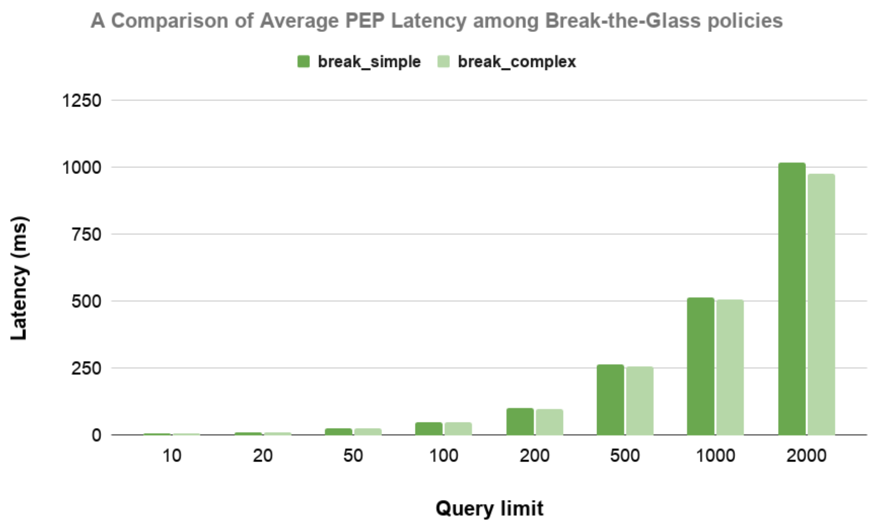


Figure 3.19: *Average PEP Latency in Break-the-Glass category*

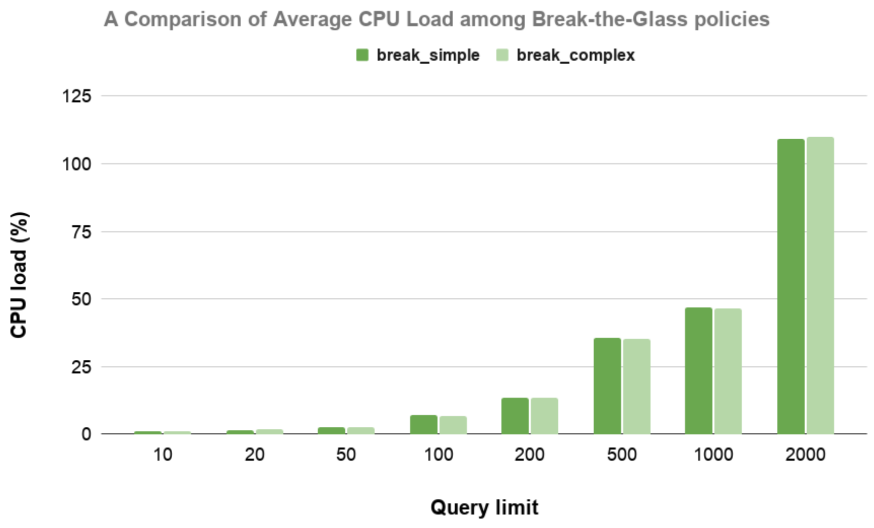


Figure 3.20: *Average CPU load in Break-the-Glass category*

to *break_simple*. The answer lies in the nature of operations that the policy executes: It filters an array, then creates a new array to store an encrypted value, and then concatenates two arrays to be embedded in an object. Apparently, these array operations, with the emphasis on the concatenation operation, which is unique in our evaluation set for this policy, is just as resource-demanding as its pair.

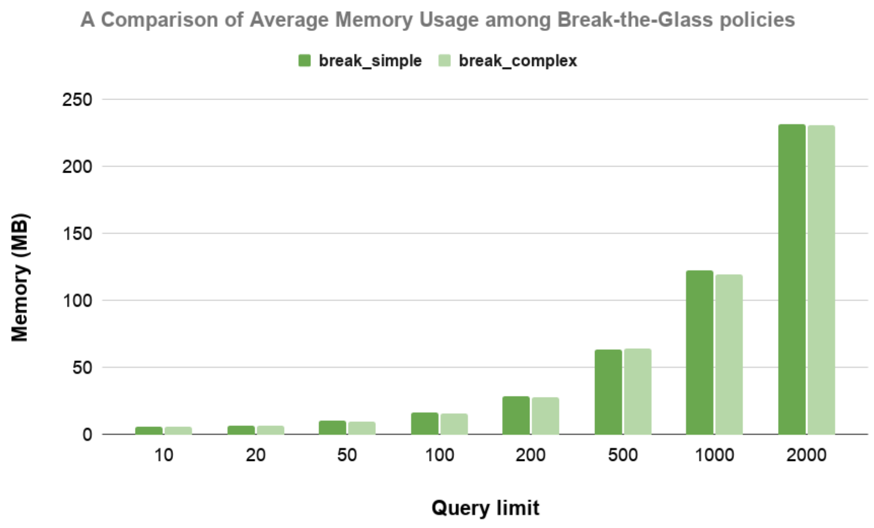


Figure 3.21: *Average Memory usage in Break-the-Glass category*

Just as we expected, the Break-the-Glass policies are the most demanding ones that can be evaluated on the PEP node, but even with the increased cost they can achieve the expected results. While the essence of these policies is to transform and encrypt the data, it is important to avoid array operations as much as possible, as they only further increase the cost when potentially cheaper workarounds might be available.

Performance in Infrastructure

It is interesting to see how the latency of the PEP node affects the latency of the entire infrastructure. From each policy pair we took the one with the greater response delay and compared it with the system latency in figures 3.22, 3.23, 3.24, and 3.25.

Based on these results, we may conclude that when the data set is increased, the increase in system-wide latency, PEP latency and the relative latency of the two components are all nonlinear.

The difference between the complexity of the various policy categories, on the other hand, is not always as clear as we initially assumed. The results of *role_complex* and *context_simple*, for example, are almost identical, and the care team identifying *role_complex* even turns out to be somewhat more demanding than context1, which

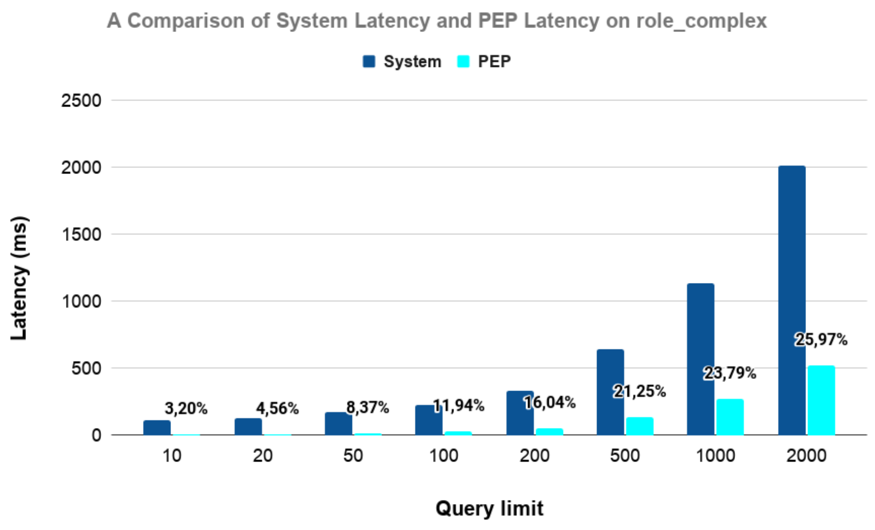


Figure 3.22: *System Latency and PEP Latency on role_complex*

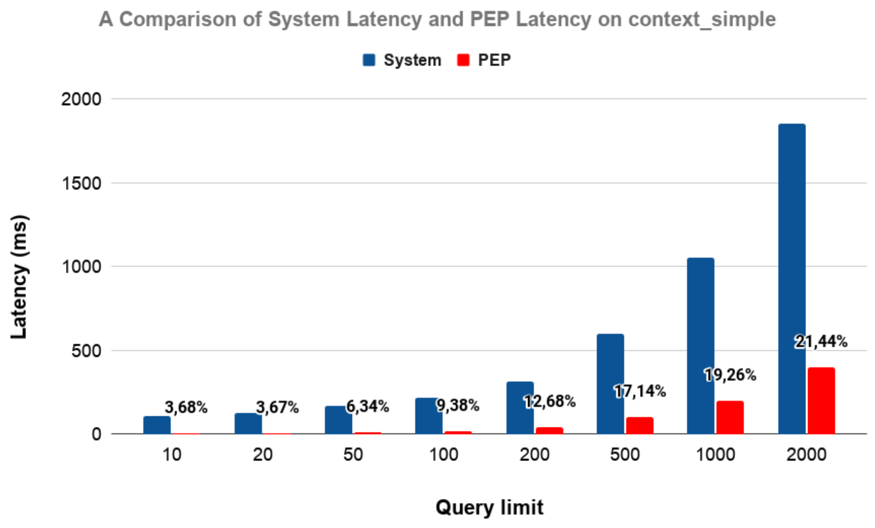


Figure 3.23: *System Latency and PEP Latency on context1*

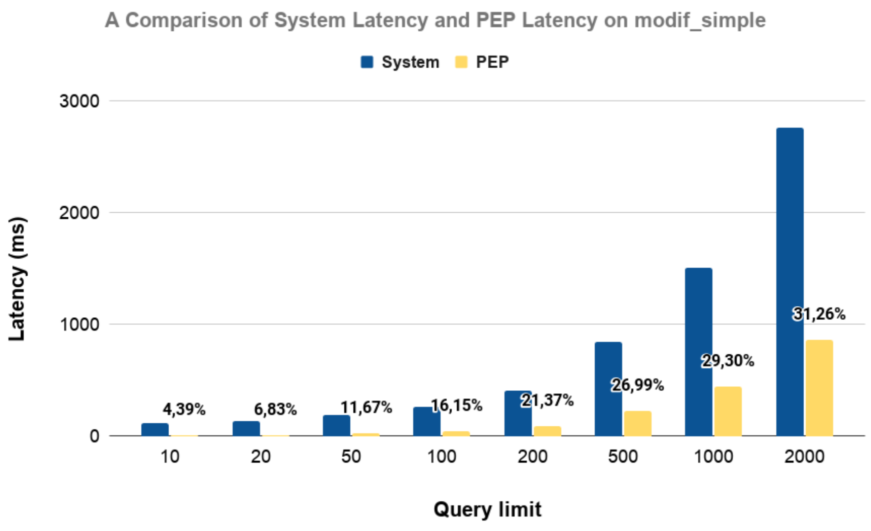


Figure 3.24: *System Latency and PEP Latency on modif_simple*

has to iterate and filter the contents of an embedded array, and with an input of size 2000 on *context_simple*, the PEP only provides the 21.4370% of the full system latency and 25.9672% on *role_complex*.

However, most of our expectations were confirmed by the results we obtained. Although it is clear that the identification of good practices, patterns and anti-patterns is necessary in the next phase of our research to further optimize the use of the PEP, the relative complexity and cost of the different categories were as expected. There is an overall latency and efficiency of the prototype infrastructure - a barely noticeable increase when we consider that the majority of healthcare applications, including our client application in its unmodified state. This requests 100 or 200 documents in a single operation, and we found the PEP (and OPA as its implementation) to be a very effective component of our security solution.

Discussion

The meticulous evaluation of the simulation of the processing edge has yielded a wealth of insights, particularly concerning the performance of policies within distinct categories. The unexpected speed with which Contextual Evaluation (CE) policies operated in comparison to Role Evaluation policies stood out dramatically. This suggests that when contextual information determines the outcome directly, the evaluation becomes significantly quicker, obviating the need for an exhaustive internal examination of input parameters. In addition, the observed average CPU usage reflected these findings. Break-the-Glass policies emerged as the most resource-

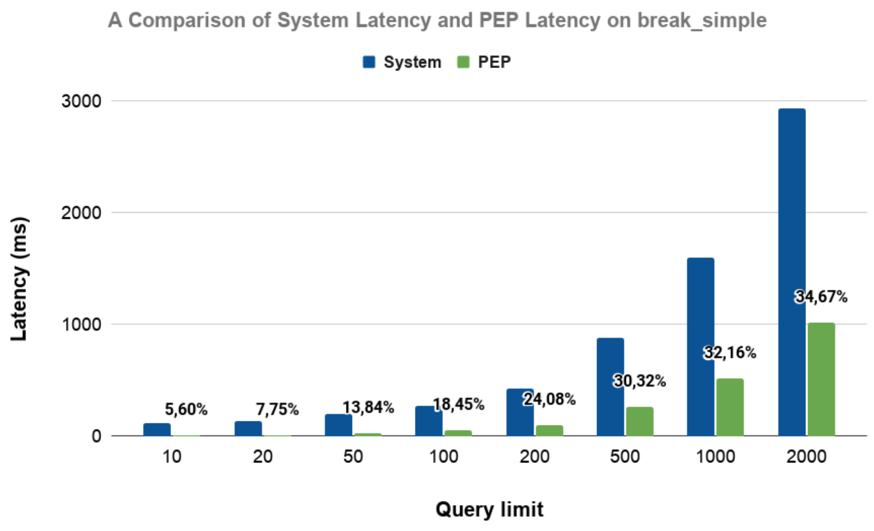


Figure 3.25: *System Latency and PEP Latency on break_simple*

intensive, while CE policies were marginally more efficient. However, the performance disparity between these categories was not as pronounced as with response time.

As we dug deeper, our data analysis, which included multiple runs for each policy, confirmed that the solution could manage complex Role Evaluation policies with ease. Nonetheless, the results support storing entities such as teams and institutions as indexed objects as opposed to arrays. Moreover, because array-based evaluations proved to be more difficult, we were compelled to reevaluate the architectural framework, particularly in terms of how context information is provided. In situations where the PEP may serve end nodes in different time zones, omitting internal queries appears to be a prudent design decision.

Moreover, our evaluations revealed that after a certain threshold, the metric values tend to converge, indicating the efficacy of contextual evaluations for policies. To improve this efficacy, it is recommended that the necessary contextual information be included explicitly in the input. Intriguingly, our assumptions about the complexity of particular policies were debunked. The performance differences between these policies were negligible, and each had distinct resource requirements. These nuances highlight the need for a more nuanced comprehension of policy categorization and its effects on performance.

In conclusion, while the results primarily supported our initial hypotheses, they also highlighted potential improvement areas. During the upcoming investigation phase, it will be crucial to identify best practices and patterns. Despite these obstacles, the overall efficacy and latency of our prototype infrastructure meet the needs of the vast majority of healthcare applications. Consequently, the PEP, with OPA as its foundation, has proved to be an indispensable component of our security solution in the processing edge, exhibiting great promise for future deployments.

3.5.3 Results from the Caching Edge

The caching edge, with its emphasis on offline storage capabilities, is exemplified precisely in environments where nodes may occasionally be offline but still operate user applications. Such situations highlight the significance of the caching edge because they frequently characterize situations in which data storage is essential, even when immediate processing or data transmission is not feasible. Here, the caching edge becomes indispensable, preserving data until it can be synchronized or analyzed later. Nodes executing user applications in potentially offline scenarios encapsulate the substance of the caching edge: they serve as both the data collection point and impermanent storage, bridging the distance between data acquisition and eventual processing.

Given the proliferation of devices and platforms, it is essential to comprehend how the caching interface functions in different environments. Web browsers have evolved substantially and now offer a variety of features, including service workers that enable web applications to function offline. By evaluating the caching edge across multiple browsers, we can gain insight into compatibility, performance variations, and possible optimization opportunities. In addition, the proliferation of hybrid smartphone applications – those developed using web technologies but encased in a native application container using frameworks, using tools such as Cordova [25] or the popular framework Ionic [124], integrating the Angular framework and Cordova – adds an additional layer of complexity. These applications frequently utilize device-specific features while striving to deliver a consistent user experience across platforms. Measuring the efficacy of the caching edge within these hybrid applications can cast light on how effectively they cache data, particularly when compared to native applications or traditional web applications.

To gain a deeper understanding of these offline use cases, particularly with regard to the average user workload, I undertook an exhaustive benchmarking exercise. By evaluating our industrial progressive web applications (PWAs), some of which were beginning their clinical trials at the time these experiments took place, we aimed to ascertain their offline capabilities and storage needs. The results of this benchmarking effort are presented in Table 3.2. In conclusion, by grounding our research in real-world applications and encompassing multiple platforms – ranging from various web browsers to hybrid smartphone applications – we intend to develop a comprehensive understanding of the caching edge. This multifaceted investigation not only highlights its significance in the telemedicine domain, but also provides actionable insights for its optimal implementation and use.

Table 3.2: *Summary of various telemedicine applications and daily data traffic*

Project Name	Average Patients per Doctor	Maximum Daily Uploads per Patient
FOG	100	3
CAPD	10	4
SZIVE	100	5
SPIRO	25	3
METSZI	60	8
INZULIN	60	8
PERIFERB	23	5
STRESSZ	80	2

While the MIMIC database contains, among other records, 27,854,055 laboratory measurements and 17,527,935 care values collected from 46,520 patients, it is clear from the summary that in a general, real-world application covering a single subset of telemedicine, the amount of data to be processed by a single practitioner is much more manageable, increasing the feasibility of offline use cases. Based on this, I defined the abstract use case for evaluating the offline access control solution as follows:

1. A single practitioner handles 50-100 patients in an application.
2. A single patient generates 3-8 measurements on average during a single day
3. A practitioner requires access to 75 - 500 documents during the course of a single day.
4. All of these applications utilize paging methodologies, listing only 100-200 documents at a time.

What I wanted to check was whether evaluating these policies in an application on a set of documents with 100-500 entries is feasible and efficient for users. I ran my measurements multiple times in the following environments:

- PWA application on an iPad Air 2 tablet
- PWA application on a Samsung Galaxy S6 Edge smartphone
- Chrome 87.0.4280.88, Microsoft Edge 87.0.664.66 and Firefox 83.0 on a desktop PC with Ryzen 5 3600 CPU and 16 GB DDR4 RAM

I integrated the WebAssembly runtime of OPA in a custom developed PWA which handles Observation documents, and included the two most critical policy categories the requirements of which must be met by the caching edge - Role Evaluation and Contextual Evaluation - in their config. In the following evaluations, role1 is the same as role_simple, role2 is the the exact copy of role_complex and context1 and context2 are likewise context_simple and context_complex.

Performance in WebAssembly Runtime

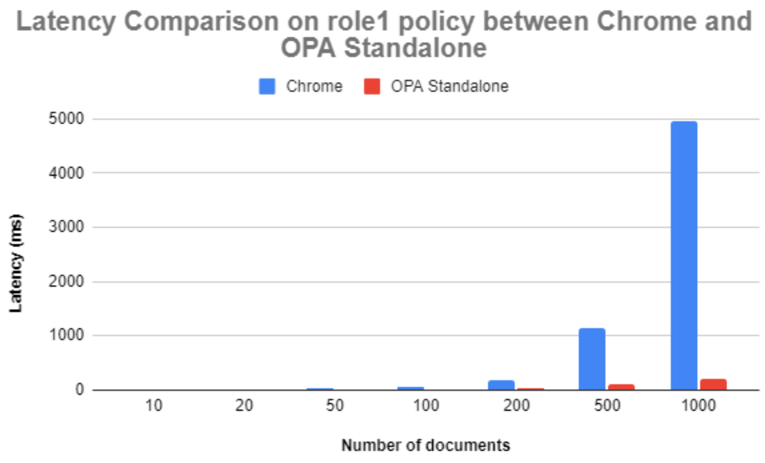


Figure 3.26: *Latency (ms) comparison between standalone OPA server and WebAssembly runtime in Chrome*

On the basis of the evaluations I was able to establish the following results. First and foremost, it became clear that the latency of WebAssembly-based policy enforcement in the application was significantly slower than with the standalone OPA server, as shown in Figure 3.26 especially after the document query size reached 500 entries. In some cases, as with 1000 entries on role1, the evaluation in WebAssembly proved to be over 10 times slower than the standalone version. Also, after the query limit exceeded 500 documents, the latency of the WebAssembly evaluations began to converge to a linear increase, instead of the nonlinear pattern we discovered in the standalone OPA deployment.

However, as long as I stayed below 500 documents, the latency was below 1 second even in the worst case scenario, as can be seen in Figure 3.27. This proves that if we combine the evaluation process with the paging mechanisms of the applications, they can solve access control in of line situations. However, the policy role2 proved to be the slowest due to the breadth-first search algorithm in the array structure

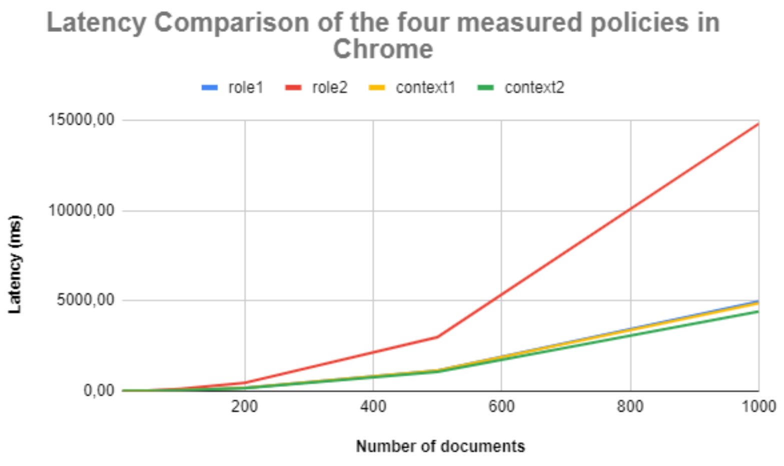


Figure 3.27: *Latency (ms) comparison between the four policies in Chrome browser*

containing the connections between care teams and handlers, which could lead to marking these types of policies as anti-patterns in the WebAssembly runtime.

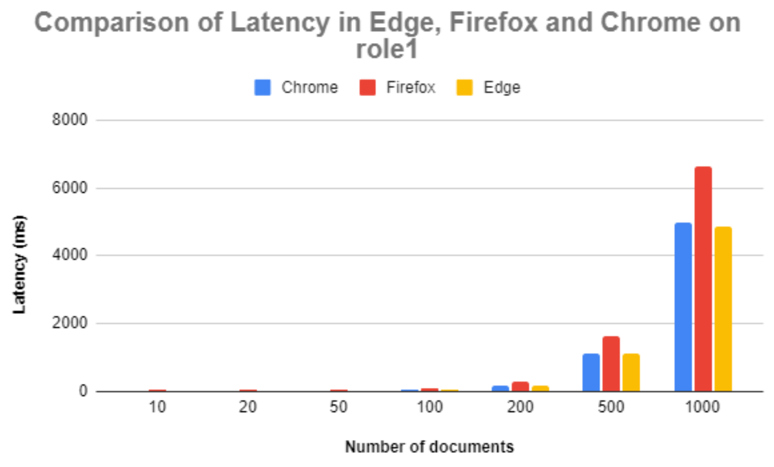


Figure 3.28: *Latency (ms) comparison on role1 between browsers on the desktop PC test environment*

In addition, latency can improve or worsen depending on the exact operating environment, as shown in Figure 3.28. On the desktop PC, the Edge and Chrome-based evaluations were very similar, but Firefox's results proved to be different, less efficient, due to a possible difference in the handling of heap memory, which warrants further measurements and a deeper investigation of the possible optimizations in this browser.

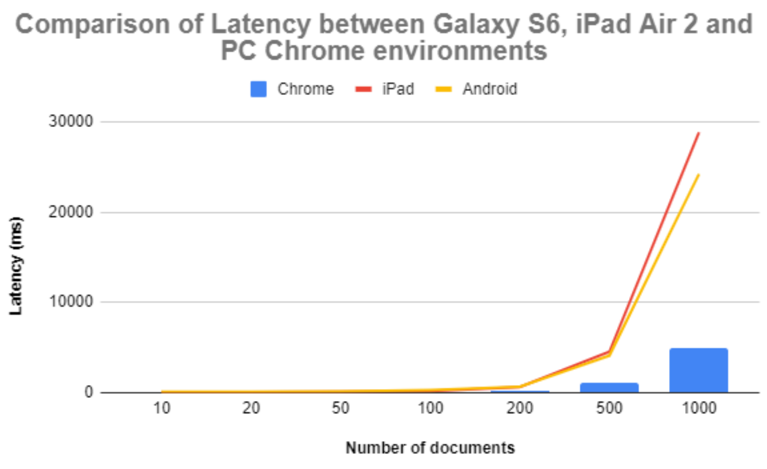


Figure 3.29: *Latency (ms) comparison on role1 between Chrome on desktop PC and mobile devices*

Before the measurements, I assumed that due to CPU and memory constraints, latency would be highest on mobile devices. This assumption was confirmed by the results shown in Figure 3.29. even though the process was slightly more efficient on Samsung Galaxy S6. With the paging mechanism, the process can still work efficiently with a load of 100-300 documents, but even in these cases the latency is higher than on PC with the same dataset.

Finally, I was also able to confirm a conjecture from our previous work, where the results suggested that the policy category Contextual Evaluation might be more efficient than the category Role Evaluation, as shown in Fig. 3.30. In the current set, I provided the contextual attributes externally to the OPA runtime, and I observed significantly better results in the category that we originally assumed to be more complex and challenging.

The atomic check of whether the status of a document is active or not was naturally faster than the breadth-first search in an array containing the links between care teams and handlers This leads to better latency with the increase in document volume.

Discussion

My exhaustive investigation into the potential Caching Edge nodes yielded a wealth of insights, especially concerning the nuances of policy enforcement and their effects on latency. The difference in latency between WebAssembly-based policy enforcement within the application and the isolated OPA server is a noteworthy observation. Once the document query size exceeds 500 entries, the WebAssembly implementa-

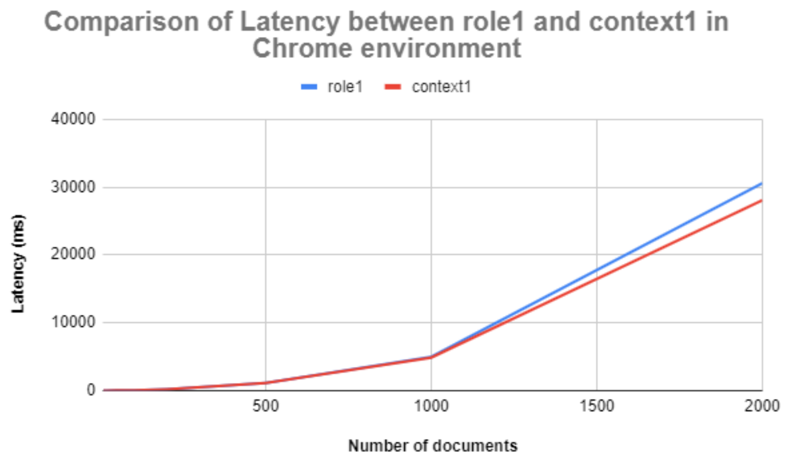


Figure 3.30: *Latency (ms) comparison between role1 and context1 on desktop PC Chrome environment*

tion experiences a significant increase in latency, sometimes exceeding tenfold. Notably, after 500 documents, the growth pattern of latency changes from nonlinear in standalone OPA to linear in WebAssembly. However, for document sizes less than 500 bytes, the latency is still manageable, even in more demanding circumstances. This highlights the potential viability of coupling evaluation processes with application paging mechanisms to effectively address inactive access control.

My evaluations also revealed that the operating environment influences latency. Firefox lagged behind Edge and Chrome in desktop performance evaluations, presumably due to its unique heap memory management strategy at the time of the experiments. The initial hypothesis, which predicted increased latency on mobile devices due to their CPU and memory limitations, was confirmed. While integrating paging mechanisms does provide some respite, mobile latencies for identical datasets are inherently higher than their desktop counterparts.

Lastly, my evaluations confirmed a prior hypothesis from the Processing Edge, which suggested that the Contextual Evaluation policy category may be more efficient than the Role Evaluation policy category. Externally supplied contextual attributes to the OPA runtime also improved performance in the Caching Edge. Evidently, discrete tests, like determining the status of a document, are more time-efficient than complex breadth-first queries in data arrays.

In conclusion, the in-depth examination of evaluations in Caching Edge environments highlights the complex interaction between policy types, enforcement mechanisms, and operating environments. While some patterns emerge as optimal, others present difficulties, highlighting the need for ongoing refinement and optimization in the swiftly evolving landscape of telemedicine.

3.6 Thesis I/3.1: An Open Source Patient Flow Simulation Tool for Validation of Results

I developed a simulation tool using open source libraries and tools that can simulate the distribution of patients and their waiting times in a hospital ward. The developed simulation tool can be used to validate the extent to which the increased waiting times due to the access control implementation from the previous theses might slow down the process based on the amount of handled data, and the impact this can have on the telemedicine data path, the patient flow churn and waiting times at crucial parts of the care process.

Publications related to this thesis: [J3],[F12]

3.6.1 Motivation

As can be read in the NEJM Catalyst short article [12], patient flow technically defines the total time frame that patients spend in and move through the healthcare system from arrival to discharge. In general, we want this time to be as minimal as possible, apart from the time required for the actual examination, diagnosis, and care processes, without compromising patient and provider quality and satisfaction. Improving the flow is essential as it can reduce the workload of medical staff and patient waiting times, but otherwise overcrowding can occur, patient health can deteriorate, while readmission and mortality rates can increase. [12, 52]. Improving patient flow was also an area of research in the 1990s. The World Health Organization (WHO) published a study using patient flow analysis (PFA), which helps researchers examine staff utilization, key patient flow characteristics, resource and financial needs, and emerging problems [8]. Another approach has been variability analysis, which involves dividing variables into groups and then determining how to measure them (e.g., severity of illness can be described as the deviation from a perfectly healthy state). The next step is to reduce or even eliminate any variability that is artificial, as it usually arises from dysfunctional processes. This should already lead to an improvement in patient flow. Further progress can be expected if natural variability is also measured and optimally managed. [72]. Our approach was to create a patient flow simulation framework that could account for different variables to calculate and measure potential patient flow. With this tool we hoped to gain the opportunity to generate validational parameters for our access control framework, by measuring the exact impact its latencies might have on the throughput of various departments based on the patient throughput and data quantities. To do this, we collected information on commonly used patient flow measures and the variables that may affect patient numbers in the Emergency Department (ED).

As the first step, our research group performed an in-depth literature review of the patient flow and analysis domain to understand the exact requirements and current state of the technology, which our custom solution had to meet.

According to the literature, different patient flow patterns occur under different circumstances. Kang and Park [63] studied the hourly visit pattern and found a bimodal distribution: the peak flow was from 10:00 to 11:00 and from 20:00 to 21:00. The lowest number of visits was between 02:00 and 08:00. In one Hungarian hospital, patient volumes increase from 8:00 and peak around 12:00. Late night hours are the least visited times, but the workload for staff is fairly constant [113]. When daily visit patterns were the focus, Hitzek et. al. [52] found that the peak in patient numbers occurred on weekends (starting on Fridays, with the highest numbers on Saturdays), holidays, and school vacations. The authors suggest that the explanation may be that people tend to engage in risky activities at these times. Varga et al. [113] also examined the difference between patient numbers on weekdays and weekends: They found similar trends, except that weekend nights were slightly more demanding.

There are also seasonal patterns of visits: Hitzek et. al [52] found the highest numbers of patients in spring and the lowest in fall. In contrast, Won, Hwang, Roh, and Chung [119] found the highest number of asthma patients in the fall, especially in September and October, and the lowest from June to August. They also found that more patients visit the ED in spring from year to year.

Linked to seasonality, but with more focus on the actual temperature Otsuki, Murakami, Fujino, Matsumura and Eguchi [84] found that during cold winters less non-urgent patients visited the ED, suggesting that people are less active in the cold weather. In contrast the warmer summer weather raised the patient numbers.

Heat waves can also impact visits to ED. Schramm et. al [101] published a study of the likely impact of a June 25-30, 2021 heat wave, affecting 10 regions of the U.S. that contain 4% of the population but accounted for 15% of heat-related ED visits. From May to June, there were 3,504 heat-related cases at the ED, 79% of which occurred during the heat wave. The peak was on June 28, when 1,038 patients arrived. In comparison, 2 years earlier on the same day, 9 patients had heat-related problems at the ED.

The usual measures of patient flow are bed occupancy rate (it is also suggested to consider the number of outgoing and incoming patients) [50, 64], transfer time (i.e., the time to prepare the bed for a new patient), and patient transfer (how many patients had to be transferred, how much time and phone calls were required to transfer, etc.). Other ED related measures may include: the time a patient spends in the department from admission to discharge, the actual time it takes to discharge a patient and/or refer them to another department, how many patients were treated in a given time interval, the wait time to see a physician or receive treatment, the number of ambulances transferred to another ED, etc. [50].

For example, Varga et. al. [113] measured how much time elapsed before medical care was initiated between different triage levels. The results showed 3.6 ± 5.8 minutes at the first triage level, 7.0 ± 11.8 and 23.2 ± 26.1 minutes at the second and third triage levels, and 37.8 ± 38.3 and 44.2 ± 43.5 minutes at the fourth and fifth triage levels.

Patient flow analysis has also been used as the basis for many research projects using genetic algorithms and in some cases, machine learning, to solve or optimize scheduling issues at various parts of the hospital process.

Yousefi et al. [121] conducted an evaluation of 38 simulation-based optimization experiments for the ED, published between 2007 and 2019. They have given a bibliographic foundation on the topics discussed, compiled data on the methodologies and tools used, and identified significant trends in the area of simulation-based optimization. They have stated that future research should concentrate on improving the effectiveness of multi-objective optimization problems by reducing their time and labor requirements.

In their study, Yang-Kuei Lin and Yin-Yi Chou [71] examined the difficulty of allocating a set of surgical procedures to many multipurpose operating rooms. They have suggested a redesigned mathematical model and four simple heuristics that ensure the efficient discovery of viable solutions to the examined issue. In addition, they provided four local search processes that may greatly enhance a given solution and used a hybrid genetic algorithm (HGA) that combines initial solutions, local search procedures, and an elite search technique to the examined issue.

El-Bouri et al. [41] conducted a literature study on the use of Artificial Intelligence (AI) to hospital patient scheduling. They addressed the many AI strategies described in the literature, such as rule-based systems, decision trees, artificial neural networks, and evolutionary algorithms. In addition, they have examined the many sorts of patient scheduling challenges that have been investigated, including surgery scheduling, appointment scheduling, and emergency department scheduling.

Seunghoon Lee and Young Hoon Lee [68] have suggested using reinforcement learning (RL) to schedule emergency department (ED) patients. They have developed a mathematical model and a Markov decision process (MDP). Then, they developed an RL algorithm based on deep Q-networks (DQN) to identify the ideal scheduling strategy for patients. In the provided cases, they have shown that deep RL outperforms dispatching rules in terms of reducing the weighted waiting time of patients and the penalty score for emergency patients.

Haya Salaha and Sharan Srinivas [97] investigated the usage of a hybrid artificial intelligence system to solve the issue of hospital patient scheduling. To enhance patient scheduling, they have presented a mix of genetic algorithms and an Artificial Neural Network (ANN). They have shown that their hybrid approach can find supe-

rior schedules than either Generic Algorithm (GA) or ANN alone, and it has been applied to actual hospital data.
An unfortunate circumstance presented our team with an opportunity, while developing this simulation tool, namely the COVID-19 epidemic. While various impacts and metrics of the pandemic itself were still being researched and evaluated by various research teams, another very active area was focused on preparing existing systems to work better and more efficiently in the event of another pandemic.
One need that most research teams agree on was the need for modeling and simulation tools. Currie et. al [37] in their work emphasized the importance of simulations to reduce the impact and severity of the epidemic COVID. They identified the following decision areas as appropriate for optimizing their effectiveness through simulations: the selection of quarantine and isolation strategies, the development of social distancing rules, the construction of lockdown release scenarios, the appropriate method for test distribution and transport, the identification of the most critical demographic groups for vaccine distribution, and the appropriate expansion and allocation of hospital resources.
Similar comments were made by Dieckmann et al. [39], whose work focused on the resources needed for effective simulation and how they can be used. In their view, simulations should focus on three main areas: educating workers about the epidemic, optimising the process of care at the system level, and assessing the needs and mental health workload of health care workers.
Improving hospital systems and patient flow to provide faster patient treatment, efficient resource allocation, and the development of techniques to avoid future infections lies at the junction of the two fields of study. Tavakoli et al.[109] recently published their results on a simulation methodology similar to ours. Although the model and triage levels are much simpler than they should be to prove accurate in simulations of Hungarian hospitals, the metrics and principles established can serve as a model for similar simulations. Terning et al.[110] had similar elements in mind, and although the simulation from their published work is still relatively rudimentary, the formulas and conditions used to evaluate their results provide a very good basis for initial validation of a similar simulation.
One of these key parameters, perhaps the easiest to follow in simulations, is to avoid overcrowding, i.e., to avoid the kind of patient flow where many patients are waiting in an area at the same time. Dinh et al. [40] specifically focused on this importance in their work, attempting to establish principles and rules to avoid unnecessary hospitalizations during an epidemic and to reduce the length of stay in the hospital. In their brief review, Janbabai et al. [56] focused on protecting hospital staff in addition to patients, focusing on preoperative, intraoperative, and postoperative processes within the patient flow. Of course, other approaches have been explored in addition to simulation-based patient flow study and analysis. For example, Arnaud

et al, [18] have attempted to use machine learning based on patient flow metrics to determine how to optimise the number of hospital beds and expedite the triage process, to name a few examples.
To investigate and design the appropriate scenarios, our team also relied on the work of Prof. Jose L. Jimenez & Dr. Zhe Peng [60] who, based on various peer-reviewed research, developed an easy-to-use tool to measure the likelihood of COVID infection in different environments based on the size and type of the area in question, as well as the number, behaviour, and condition of the people in it. Based on these results, and taking into account the fact that patients and staff wear masks in the hospital and hospitals use various distancing measures and restrictions, including a strong emphasis on ventilation, our team calculated that the probability of infection for a number of 10 to 20 patients in the area was only 4.39 % after one hour, and even after six hours it only increased to 5.96 %. This means that one of the most important aspects of optimising patient flow for COVID prevention is to keep the number of patients in a given range around or below 10 while trying to speed up the flow itself to avoid congestion.

3.6.2 Methodology

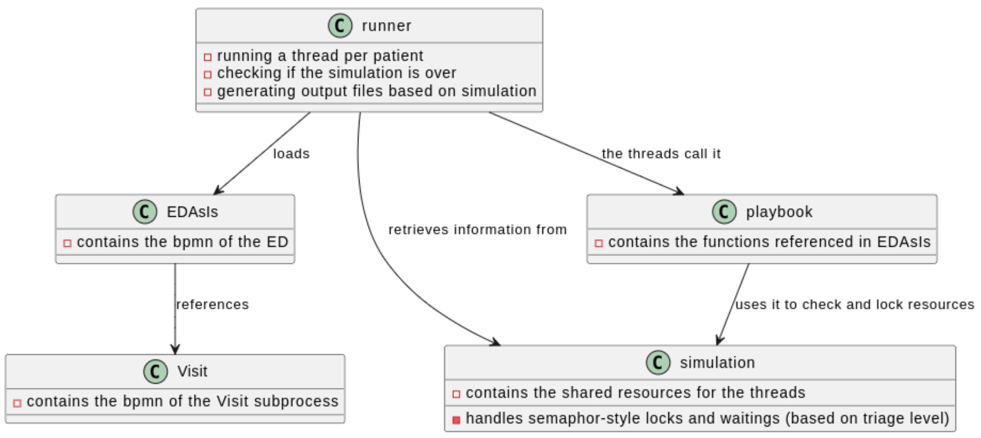
Introduction of the ED

During the early phases of our research, we used the ED model of Leva and Sulis [69], as it proved to be the model most similar to the Hungarian ones based on comparisons between this model and our team's experience and knowledge of the structure and functioning of the ED. Differences include minor changes in terms of which station is served by which staff member, and the introduction of an additional fifth triage level as mandated in the Hungarian system. The model includes 7 different lanes and a sub-process for handling the complex visit process if required by the patient's condition. The first lane is the registration process, where patients are admitted to the hospital and treated according to the severity of their condition upon arrival. They are then admitted to the triage lane where they await initial assessment. For less urgent cases, they may voluntarily leave the process at this stage if they wait too long.
After leaving triage, the triage nurse may decide to refer the patient to an internal clinic; otherwise, the visit process takes place, where the nurse or physician takes a history, takes blood, performs a radiology referral, and then decides the patient's fate based on the results. The outcome of the process may be referral to an internal clinic, admission to the emergency department, referral to an outside facility, discharge, or in a small percentage of cases, death of the patient.

Modeling and Simulation Tools

To create the hospital simulation, we chose the open-source Camunda Modeler [1], which allows us to create arbitrary processes in a parameterizable, editable, and executable format. The output of Camunda modelling is the BPMN (Business Process Model and Notation) file [2], a text file based on the XML standard that is displayed by Camunda-compatible tools and execution environments with a visual representation.
However, Camunda and BPMN modelling do not always prove suitable. In our various healthcare projects, the issue of clarity and complexity of modelling has often arisen in similar cases, especially when some clinicians and researchers wanted to model and describe processes in a way that was transparent to them, but the Camunda elements were considered too broad and complex. Our goal, therefore, was not only to accurately model the model we created, but also to make it understandable to researchers outside of IT and be able to create similarly simple processes, leaving the more complex parts to scripts and programmers running in the background.

Figure 3.31: *Flowchart of the simulation framework*


runner
running a thread per patient
checking if the simulation is over
generating output files based on simulation
loads
the threads call it
retrieves information from
EDAsIs
contains the bpmn of the ED
playbook
contains the functions referenced in EDAsIs
references
uses it to check and lock resources
Visit
contains the bpmn of the Visit subprocess
simulation
contains the shared resources for the threads
handles semaphor-style locks and waitings (based on triage level)

While developing the simulation, I considered using the official Camunda simulation tools and Visual Paradigm, among others. However, we ultimately decided to create our own simulation environment using open source tools to ensure that the simulation settings, configurations, and types of metrics collected were customizable for us. Our custom simulation is based on the Python library SpiffWorkflow [14], which can process and run bpmn models created with Camunda, among many other inputs. Scalability and robustness were key elements of the hospital workload modelling operating environment. The principle is based on the idea that each patient is a parallel running SpiffWorkflow thread sharing common resources for which we implemented waiting, handover and reservation using semaphores. The measurement and logging of wait and turnaround times in the system is differential, with each thread regularly logging its timestamps as it arrives at and departs from the stations. Our approach was initially based on the naive assumption that the bottleneck is the availability of staff in the ED, and that if the required physician, nurse, or nursing staff is available to perform the task, then the space and equipment are available as well.

Figure 3.31 illustrates the components of the framework and their precise relationships. The model defining the ED is provided in two bpmn files: Visit.bpmn for the visit subprocess and EDAsIs.bpmn for the ED architecture, which references Visit in the correct place. The framework's starting point is runner.py, which specifies how many patients must be allowed into the system for the simulation, what stop condition must be satisfied to terminate the simulation, and also manages the extraction of the various metrics at the conclusion of the simulation (the latter activity is expected to be handled by a separate module in a future version). The runner.py parses the contents of the bpmn files and utilizes them to generate runner threads for each patient that will execute the steps specified in the bpmn files. The simulations employ playbook.py to execute the simulation of each step and simulation.py to indicate when a shared resource (e.g., doctor, nurse) is required, lock it using semaphore, or set a triage level-based queue if there are no available instances of that resource.

Modeling

The following section presents the model and elements of the Camunda workflow based on the combination of the Leva and Sulis paper with elements from the Hungarian hospital system. Table A.1 in the Appendix shows the content of the first two lanes, registration and triage, the first stations that are the same for every patient in the hospital. The elements of the simulation script are handled either as events, where patients must acquire a shared resource and then perform some processing before proceeding, or as end states, which, when reached, terminate the patient's simulation thread. The required resource in the simulation is a member of the ED staff: Generic Nurse (GN), Hospital Employee (HE), Specialized Nurse (SN), Doctor (DR) and Generic Operator (GO).
Note that due to the deterministic nature of the simulation, these stations and steps model how a patient is admitted to the hospital and then treated, with the environment assigning almost the entire pathway to the patient at the beginning of the simulation with all the important attributes. Our research team had two main reasons for this: On the one hand, the methodology gives researchers who might use our tools in the future the ability to analyze and debug the expected runtime of the sim-

ulation without having to wait for the entire simulation to run. On the other hand, it also gives us the ability to manually enter patients into tables in order of arrival with their severity, and even to examine specific cases in minute detail using the tools. The modeling of these pathways raised a serious research question at the beginning, as the international literature and the original source of this model indicated that the necessary personnel for these pathways are the general nurses, and in order to keep the simulation accurate, we decided to stick with this version. However, in Hungarian hospitals it is much more common to have at least one physician present during these phases. In our further research, collecting more specific information from Hungarian hospitals, including those we have conducted research with, we want to test different modifications of this trace and see how they might change some of the results and trends we have obtained during our previous research. The next major step is the Visit, which is modeled as a separate subprocess. The elements of the Visit model can be seen in Table A.2. The main difference from the main lanes is the need for specialized nurses and doctors, and the many optional pathways depending on whether blood tests or radiological examinations are required.

After the Visit subprocess, the only step left in the simulation is the processing of the outcome, which is usually performed by a specialised nurse (SN). There are five possible outcomes defined both in the paper containing the basic version of this model and in the papers analysing Hungarian hospitals: Death, Hospitalisation on Ward, Discharge, Transfer to External Facility, or Transfer to Internal Clinique.
We also achieved the desired simplification in modelling. Since the model was not overly complex, we used only four elements that were visually and practically comprehensible: the start point, the end point, the event, and the decision point. These were simply augmented during design with information about which event gave the patient which additional attributes, and the decision points were then used to select exactly what criteria the patient should use to choose the direction of travel in the simulation. The entire modelling process thus consisted of a total of four elements, plus a few lines of pseudocode description for the events, which is not only simple, but also compatible and interoperable with many other modelling tools. The timer event was considered as a fifth element type, but it was ultimately ruled out due to the system's standardized time management. All wait periods are supplied to the framework as arguments or thresholds with defined values. SpiffWorkflow's scripting and customisation options are restricted in this domain, and the timer event would be conducted in real time regardless of the simulation's time format.

3.6.3 Simulation

Our goal was to study how an emergency department ideally operates and how unexpected events can occur, using this operating environment and the modelled emergency department with the number of patients arriving, the severity of their cases, the probabilities and rates for each branch from real data. Or in the case of an epidemic, how to optimise turnaround and wait times (since similar studies in many cases have only looked at similar models in terms of staff time or budget): Is it clearly a good idea to increase staff and the number of rooms and equipment needed to perform each activity?

Scenarios

To be able to create different situations and scenarios to analyze how small changes in patient flow, staffing, or processing time of the different stages affect the simulation throughput and metrics, we first created a baseline scenario based on real data from the ED of Somogyi Kaposi Mór Practicing Hospital [113] to estimate the rates of patient arrival and distribution between the five triage levels (i.e., the urgency of each case) to model. According to their data from 2015 statistics, the ED sees approximately 90 patients per day. In terms of triage levels, 0.67% of patients had triage level 1, 1.24% had triage level 2, 23.35% had triage level 3, 40.17% had triage level 4, and 34.54% had triage level 5. Triage levels 1 and 2 require immediate treatment, level 3 can tolerate waiting times up to 30 minutes, level 4 up to 60 minutes while level 5 even up to 2 hours.
As for the fate of the incoming patients after treatment: 20.9% were hospitalised, 2.7% voluntarily discharged, 1.5% were referred for triage, 0.4% were transferred to another inpatient facility, 0.4% died and 73.5% were discharged to their home.
The baseline scenario was based on the work of Leva and Sulis and was run with 3 doctors, 2 generic nurses, 3 specialist nurses, 2 clinical staff and 4 generic operators, with a 20% chance of a new patient arriving every minute - this resulted in the most even distribution, the element of the simulation to handle increasing or decreasing patient arrival density at given times is currently being tested and will be included in a next pilot phase. The turnaround times at each station, which depend on the triage level, follow the one-to-one model of Leva and Sulis, considering triage level 3 as the dividing line between urgent and less urgent cases. For all other scenarios, these original distributions and proportions were shifted through a type of exacerbated bias. In some cases, we increased the severity of incoming patient cases, in others we reduced the number of emergency department staff, and in still other scenarios, to approximate the impact of COVID, we used estimations of the need for and duration of decontamination to increase wait and turnaround times at each station in the simulation. The type and number of staff in the emergency department was based on the paper by Leva and Sulis. This base scenario is referenced as SC0. Another scenario, SC3, showcases a test scenario which tried to simulate the estimated patient load of and increased waiting times (due to disinfection and other procedures) of a pandemic scenario.

- **SC0**: This is the basis of comparison made by merging of the Somogyi Hospital and the Italian sample. Patients are rarely admitted for urgent triage 1 or 2. The time spent at each station follows the original pattern drawn from the papers. **Specification**: 3 doctors, 3 generic nurses, 2 specialized nurses, 2 clinical staff, 4 generic operators; regular processing times; triage distribution: l1-0.00673, l2-0.01241, l3-0.23359, l4-0.40175, l5-0.34549; 20% patient arrival chance. **Expectation**: Patients with lower triage levels have to wait longer at common stations (registration, triage), where congestion and waiting times increase, but the time spent in the system remains within acceptable limits.

- **SC3**: An epidemic-inspired scenario. With only urgent patients coming to the hospital (less urgent cases are not even admitted), the number of staff in the Emergency Department has been increased, but also the minimum waiting and turnaround times due to disinfection procedures. **Specification**: 6 doctors, 4 generic nurses, 6 specialized nurses, 8 clinical staff, 4 generic operators; processing times are increased with a few minutes to simulate disinfection; triage distribution: l1-0.1873, l2-0.2241, l3-0.13359, l4-0.00417, l5-0.00345 ; 20% patient arrival chance. **Expectation**: Barely any patients from lower triage levels, but significantly increased times for higher levels.

3.6.4 Results

Each scenario was run with a load of 90 patients (the daily average based on Somogyi Hospital data) and was intended to fill a 7-8 hour shift in the emergency department.

SC0 - Base Scenario

At SC0 we immediately noticed some interesting differences compared to our first hypothesis. As seen in Figure 3.32, the pre-visit phase had the longest combined times (i.e., waiting and execution combined), while other elements of the first two lanes, such as registration and urgency assessment, were relatively short.

Similarly, the maximum number of patients either waited or were studied at the same stage of their simulation. As shown in Figure 3.33, the longest queues were in radiology, diagnosis, and outcomes management after the visit.

A smoothed trend using the LOESS method can be seen in Figure 3.34, which shows how the number of patients at a given station has changed over time in the simulation. This also justifies that while the registration and triage processes began to congest early in the simulation, they never experienced as high a number of waiting patients as outcome management, radiology, and diagnosis establishment.

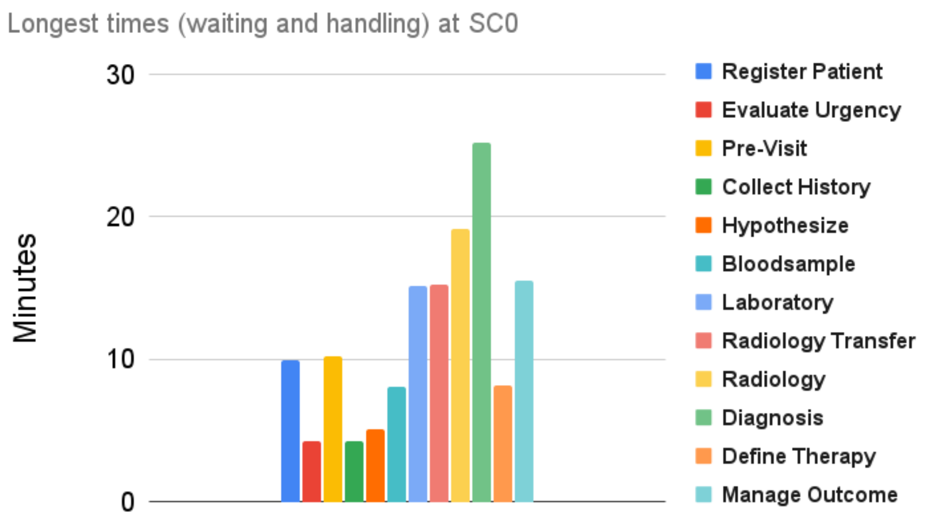


Figure 3.32: *Longest combined times in SC0*

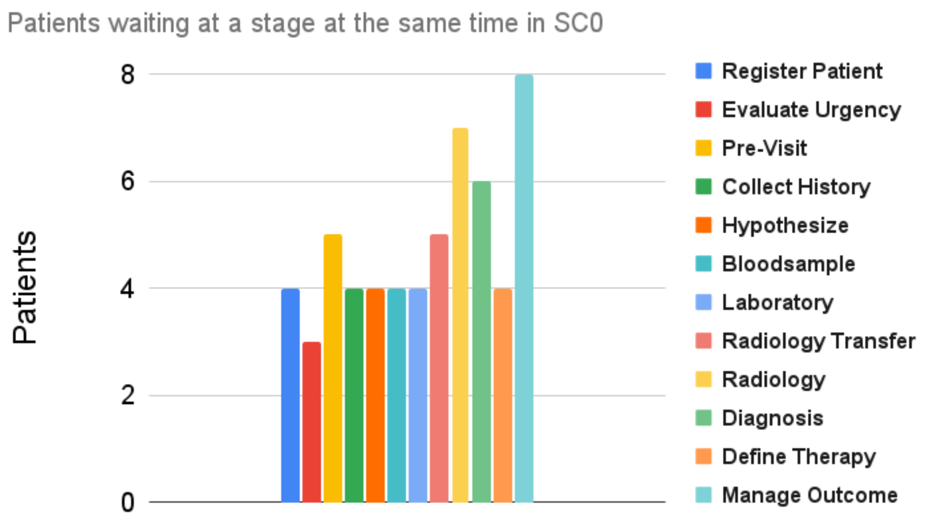


Figure 3.33: *Maximum number of waiting patients in SC0*

As for the comparison between the various triage levels, these values can be seen on Figures 3.35 and 3.36.

Based on the distributions and probabilities of the Hungarian hospital, not a single triage level 1 patient was admitted to ED during the simulated day. Triage stage 2, of course, had the shortest average time, while others had significantly longer times, with a bottleneck in the outcome management phase after the visit process.

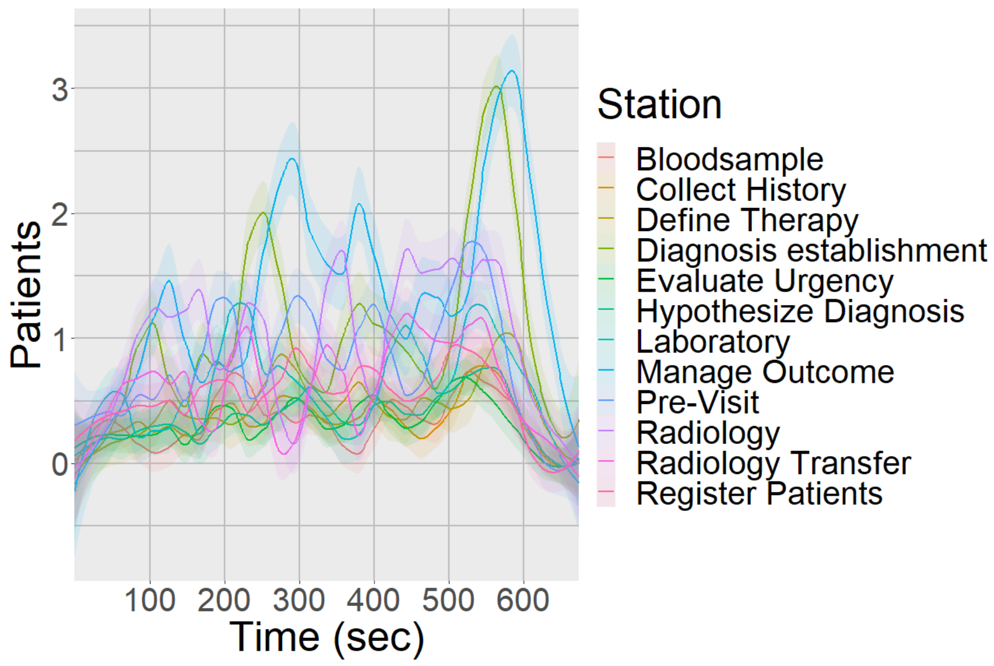


Figure 3.34: *Patient number trends in SC0*

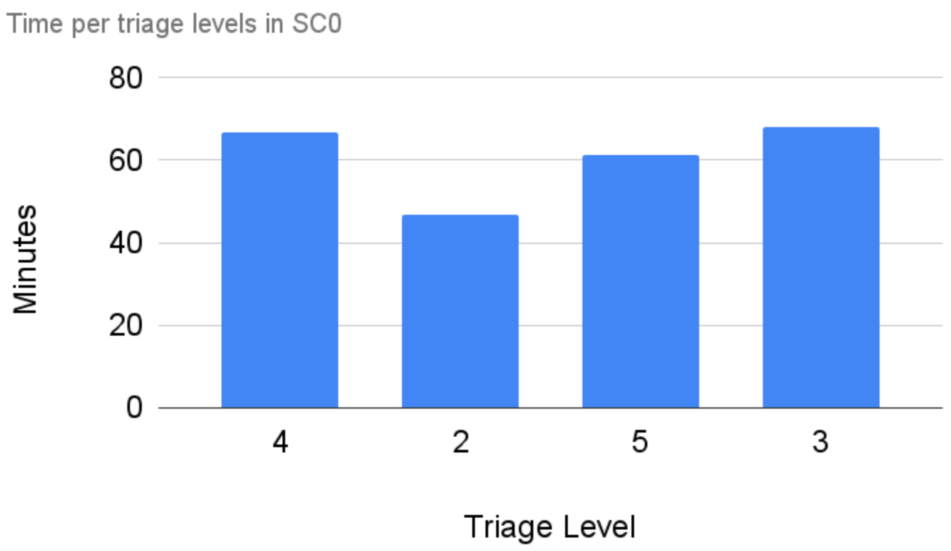


Figure 3.35: *Average times spent in the simulation per triage level in SC0*

SC3 - Pandemic Scenario

Scenario 3 was the most critical simulation and the most important for our further research, as we attempted to create a patient flow whose points and biases reflected

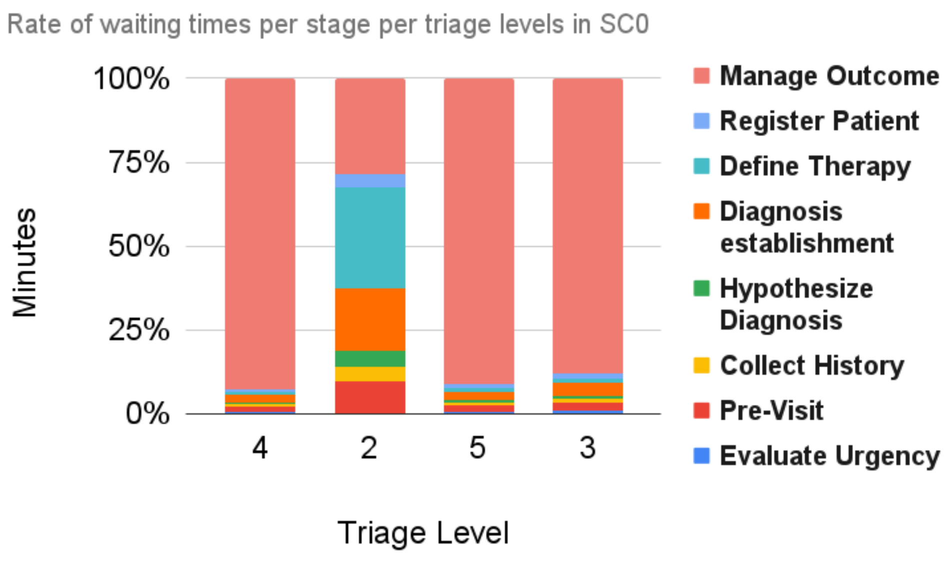


Figure 3.36: *Distribution of times spent at the various stations per triage level in SC0*

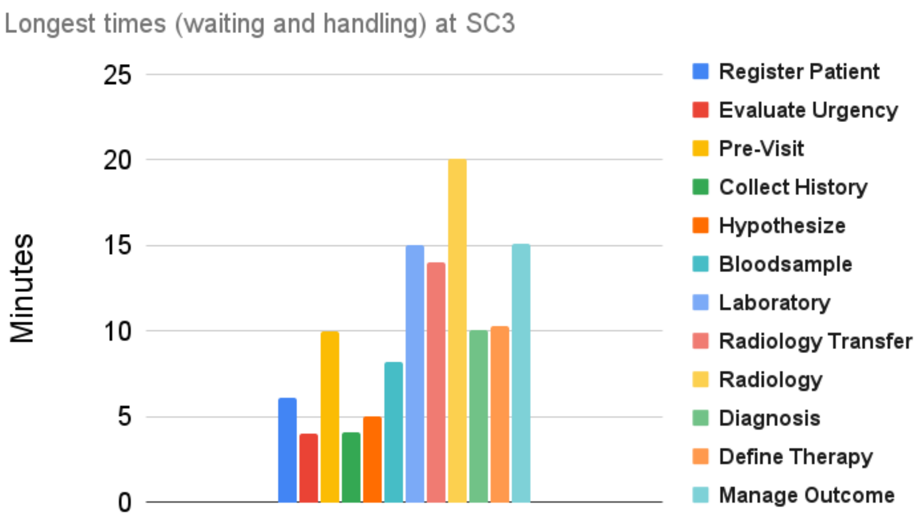


Figure 3.37: *Longest combined times in SC3*

the characteristics of an actual epidemic compared to the base case. In this case, the emergency department was visited only by urgent patients, typically with a triage level of 3 or higher, and the flow was slowed by the fact that although the number of staff was increased, the passage through the phases was much slower because of mandatory decontamination. Figure 3.37 shows the primary consequence: average wait and turnaround times per station are significantly longer than for the original cases (especially considering that the majority of cases here required urgent care). The number of patients waiting at the same time has also increased significantly, as

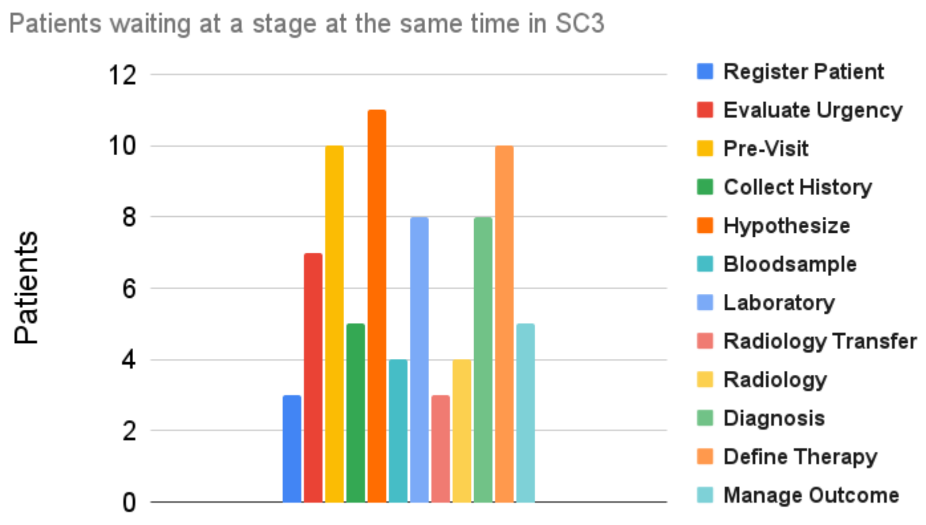


Figure 3.38: *Maximum number of waiting patients in SC3*

can be seen in Figure 3.38. In addition to the bottlenecks defined so far, the one that stands out is the hypothesis_diagnosis, the step in the visit process where the patient is first seen by a physician in our model rather than by various nurses and generic operators. This increase can also be seen in the weighted trends in Figure 3.39, which are not only much higher than the results in the previous scenarios, but the peak is not a breakout point, but an extended phase that takes up a significant portion of the simulation runtime. In other words, the congestion problem started much earlier, and as the later phases slowed, the number of patients in the backlog did not start to decline as much as in the earlier cases, even though there should have been more staff available and the patients would have warranted a faster process due to the high triage levels.

Figures 3.40 and 3.41 also confirm that, as expected, almost exclusively patients with a triage level of 3 or more were admitted to the emergency department. On average, levels 1 and 2 were completed within an hour. As for the time distribution, it is interesting to note that it is quite similar for the three triage levels, with a significant proportion still managing the outcome of the visit in each case.

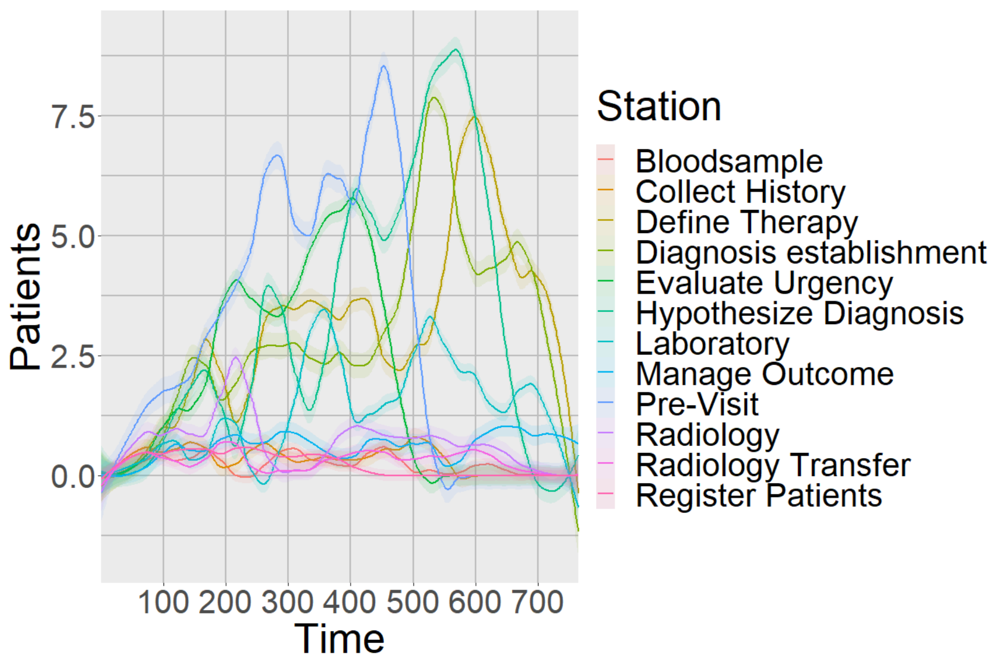


Figure 3.39: *Patient number trends in SC3*

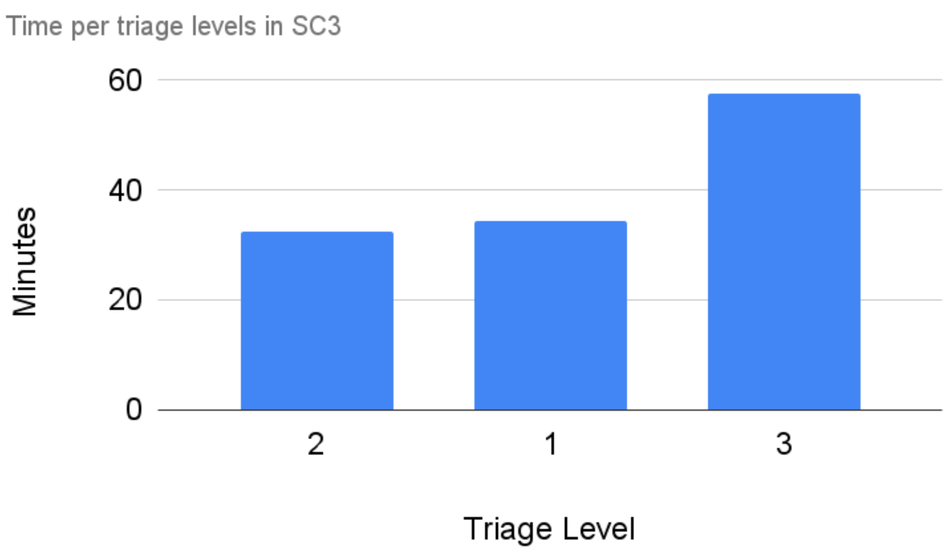


Figure 3.40: *Average times spent in the simulation per triage level in SC3*

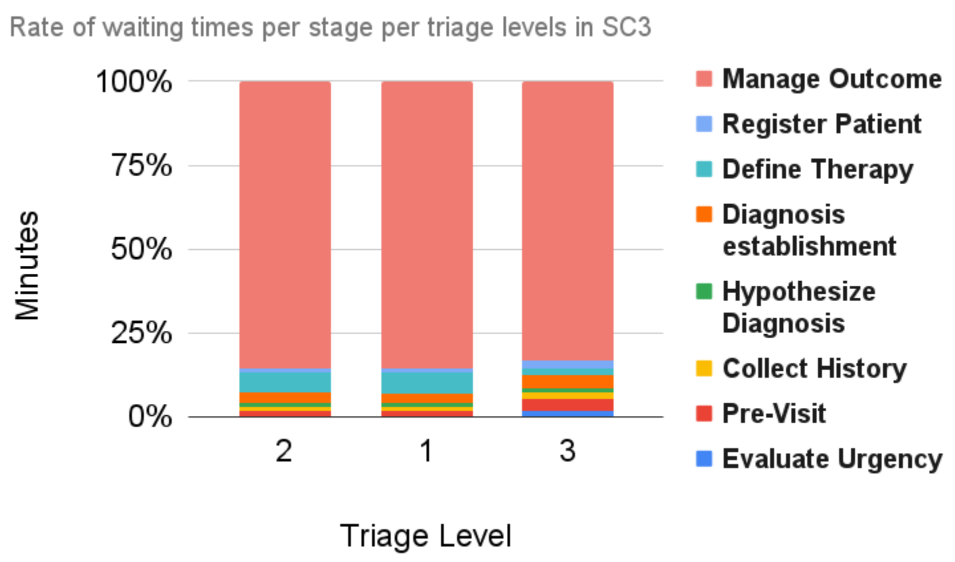


Figure 3.41: *Distribution of times spent at the various stations per triage level in SC3*

3.6.5 Discussion

In reviewing the simulation results, we found the following. First, the expectations for each scenario were either met or deviations occurred that can be interpreted based on the simulation run. Thus, despite the fact that the modelling toolbox itself has been simplified in line with our original objective, the model functions with the same accuracy. For example, with input from Hungarian and Italian hospital sources, the results of Scenario0 meet all specifications, from patient waiting times through triage level prioritisation to the maximum number of patients waiting at any one location. For the additional scenarios, the biases also yielded the expected results, so we can say that ***the reduced modelling toolkit, supported by scripts based on the SpiffWorkflow library, met our expectations and can be used for further simulations.***

Examination of the trend plots also confirmed our theory, that in a scenario such as SC3, overcrowding becomes much more continuous due to the time gained from decontamination, with the peak typically occurring in the middle of the Emergency Department simulation, typically at points where throughput was already more critical. In both scenarios, it has been shown that the most problematic phases in the patient flow are determining therapy and waiting for the visit to be evaluated. If the goal is to comply with COVID recommendations and reduce potential infection rates, these might be the stages of patient flow where it is worth either reducing the time spent in the waiting room or, ***if this is not possible, providing patients with more***

separate, well-ventilated waiting rooms where they can wait for results without risking an extended stay that could reduce the effectiveness of infection prevention.

Moreover, unlike many commercial solutions such as Visual Paradigm [11], Simcad Pro Health Simulation Software [9], or Simul8 [10], our solution is a significant improvement in that it provides both free modeling and model execution, the source code of the modules used can be modified freely, as can the simulation's exact elements and output. In addition, the simplified modelling toolset and the BPMN file format do not restrict the usage of the created model, so if a research team has access to alternative simulation systems, the generated model may be utilized as-is or with minor modification. And its usage in its current form, maybe with minor enhancements, enables it to be utilized in conjunction with other simulation and assessment tools or by other processes. For instance, the data may be automatically merged with the hospital's measurements, which can then be run through Jimenez and Peng's tool [60] to create an accurate picture of the possibility of COVID spreading in a particular department or institution. Since the output is customizable, it may be used to study a broad variety of optimisation tasks, answering the demand mentioned by Yousefi et al [121]. The output and Python-based framework will presumably be of great value for reinforcement learning.

The patient flow within the healthcare system is a crucial factor that can impact both patient outcomes and the effectiveness of healthcare delivery. In this study, a comprehensive simulation framework was implemented to determine the effects of various scenarios on patient flow, with a particular emphasis on the Emergency Department (ED). The primary objective of this investigation was to create a tool that can determine the impact of the latencies introduced by our access control solution from the previous theses on overall patient throughput and data volumes.

Our method distinguishes itself from commercial solutions by providing free modeling and execution. Due to the adaptability of the BPMN file format and the adaptability of our solution, the generated model can be seamlessly integrated into alternative simulation systems. Its compatibility with other assessment tools provides opportunities for future research and practical applications, such as predicting the spread of infectious diseases in a hospital setting.

In conclusion, our simulation framework has proved to be a valuable instrument for comprehending the dynamics of patient flow in various scenarios. The results not only validate the delays introduced by our access control framework, but they also provide a road map for optimizing patient flow in healthcare contexts. As healthcare systems around the world contend with the dual challenges of delivering efficient care and managing infections, our tools will be indispensable for molding the future of patient care.

The latest version of the developed tool is available at:

https://github.com/szaboz/ActaPatientFlow

3.7 Chapter conclusion

Proper authorization management in the domain of telemedicine and e-health presents a number of challenges that classical authorization management methods are insufficient to cover due to the countless contextual factors affecting the level of access, the dynamic scope of authorization for roles, and the lack of industry-wide agreed procedures. The problem is especially severe on the periphery of the telemedicine data path, where the various computing and storage capacities of different types of nodes must also be taken into consideration in order to satisfy the requirements so that there can be no significant delay in the duration of the services. During this chapter, I defined a taxonomy by which I distinguished the four policy categories that together can encompass the access control requirements that arise along the e-health data path. Based on their capabilities and abilities, I have separated two types of edge nodes: processing edge, which can almost simulate cloud performance in terms of local data requirements, and caching edge, which has lesser computing but still robust storage capacity. As a member of a research team, I have also examined the stability of smartphone-based peer-to-peer networks, but I have come to the conclusion that further testing and simulations are required to determine whether they can implement a local e-health infrastructure as well. I defined two example rules for each policy category, one more simple and one more complex, and then developed two test environments, one for the processing and one for the caching edge. With the aid of the Open Policy Agent (OPA), I implemented my proposed access control solution based on the Policy Enforcement Point (PEP) principle, whose performance I evaluated in both test environments with an ever-increasing volume of documents. Although some of my hypotheses regarding the relative complexity of the rights management categories were disproved, the results showed that the latency caused by my solution accounted for less than 35% of the total runtime on the processing edge, even with the largest amount of queried data, while the proportion was less than 30% on smaller amounts; while on the caching edge the delays were much more significant, but as long as only a few hundred documents were handled, the method proved to be effective on both desktop PCs, smartphones, and tablets. To measure the exact effects of the delays caused, I have also been involved in the development of an open-source patient flow simulator that makes it possible to construct scenarios, parameters, and thresholds that enable more accurate validation of these access control implementations. Although the results of my years of research and experimentation can still only cover a portion of the domain, I believe my solutions and findings provide a great basis for both practical solutions to the problem and further research on the domain. It is also worth noting that, due to the e-health developments of the team I am part of, a significant portion of these results were, are, or will be used in applications and real-life scenarios. The author of this PhD thesis is responsible for the following contributions presented in this chapter:

I / 1. I explored the complex requirements of modern telemedicine applications in terms of access control. I defined a taxonomy to formalize the different types of access control policies and the TAPE requirements necessary to ensure that the implementation of the defined policies can guarantee a balance between data privacy compliance and responsiveness at any point in the telemedicine infrastructure.

I / 2. I proposed special categories of edge instances beyond the cloud that represent a special category of modern telemedicine infrastructure from the perspective of access management. I formally defined a categorization of these into storage and processing edges. I then discuss the increasingly prevalent smart device based peer-to-peer networks as an extreme case of edge solutions, and analyse their potential and stability to function as stand-alone edge networks.

I / 3. I presented the implementation of the proposed access control solution, then I will set up test environments to represent both edge types, and sample rules to validate the effectiveness of the implementation under increasing data volumes. During the measurements, I examined the resource requirements of the nodes performing the evaluation, as well as the latencies measured for the data retrieval processes. I present the measured latencies which confirmed, that for reasonable amounts of data at the various edge types my policy categories met the requirements established in the previous theses.

I / 3.1. I developed a simulation tool using open source libraries and tools that can simulate the distribution of patients and their waiting times in a hospital ward. The developed simulation tool can be used to validate the extent to which the increased waiting times due to the access control implementation might slow down the process, and the impact this can have on the patient flow churn and waiting times at crucial parts of the care process.

Chapter 4

Thesis Group II: Potential Data Leak Detection of Progressive Web Applications with Large Language Models

In the subsequent group of theses, I focused on a crucial component of the telemedicine infrastructure, namely the front-end applications at the end of the data path. I defined a taxonomy that defines and ranks sensitive data in applications based on the impact of data leaks, and another taxonomy to determine the protection level of applications, in such a way that the defined categories can be transferred to LLMs using prompt engineering techniques. Then, I presented my results in classifying the elements of a variable-name dictionary and then in detecting sensitive data from open-source front-end applications via the GPT-3.5 and GPT-4 APIs, with which I have validated the hypothesis that through the complex knowledge of the LLMs at the level of the current GPT models, machine learning in the classical sense can in some cases already be derived by passing the necessary knowledge in the form of well-written prompts to the LLMs. Lastly, I discussed my results in static code-based detection of potential application vulnerabilities by evaluating the application protection level classification and then combining it with the results of the sensitivity detection.

Publications related to this thesis: [J4]

4.1 Introduction

In recent years, the development of Large Language Models (LLMs) has accelerated substantially, and in the past year, both their popularity and adoption have attained new heights due to popular services such as ChatGPT [38] and its competitors. Numerous research groups are currently investigating the precise impact of the use of these models in various fields and the extent to which they can be used for process optimization and decision support improvements, with a particular focus on the IT sector and programming, in addition to education, healthcare, administration, and business. While more research is focused on the goodness and optimality of the code generated by Generative Pre-Trained Transformers (GPTs), as long as adequate prompting has been used to avoid so-called ”hallucinations”, ie. estimated but nonexistent libraries and function calls [57] - the limitations of the models remain unclear. There is a similar challenge with regard to interpretability, or the depth to which a large language model can comprehend the function and purpose of the code in order to test, debug, or even enhance it with minimal developer intervention.

This kind of AI-assisted debugging and code interpretation also seems to be a particularly interesting area because, with the right semantic and logical knowledge, it is likely to be able to detect potential sources of errors and problems in the code, which could only be discovered through extensive testing. Trends in recent years suggest that testing itself is becoming an area where AI-enabled tools are becoming increasingly popular [54]. The significance of aggregating and making the code base more transparent and manageable for large, complex code bases in order to effectively manage configurations such as vulnerability analysis and security configuration enforcement is not a new concept. Heydon et al. [51] introduced Miró, a set of visual description languages, in 1990 to make security configurations of software systems more straightforward and transparent. A similar motivation led Giordano and Polese [49] to introduce the Vicoms framework, which allowed developers to administer fine-grained, role-based access control settings for their Java applications in a manner that was distinct from the source code but still readily comprehended. However, the implementation and utilization of such an instrument, as well as its testing, are invariably subject to the aforementioned code interpretation.

The architecture of modern web frameworks, such as Angular [4], makes it significantly more challenging to build a DOM structure in line with the call graph and dependency graph for single-page applications due to elements such as modular nesting and partial navigation of components. Even the very first version of Angular was challenging from an analytical standpoint. Misu et al. [53], for instance, created FANTASIA, an AST-based utility for analyzing AngularJS MVCs, to aid developers in detecting inconsistencies in static source code. While innovations such as high modularity, the introduction of typology, and object-oriented paradigms, which are highly valuable for debugging and design using TypeScript, have helped developers, the increased modularity, more complex call and dependency graphs, and difficult-to-understand navigations have made it very difficult for both static and manual analysis, similar to other emerging modern front-end frameworks.

In the past, our research team has investigated the issues of access management and data security [107] in the design of healthcare applications, with an emphasis on optimizing the access control implementation with the goal of obtaining the necessary information from the databases as quickly and efficiently as possible, without any improper use, and ensuring that only authorized entities have access to the information. However, there were limitations at the end of the data path for the front-end applications, as quantifying the security of the applications or web applications located there would be required to evaluate their security, while this information is necessary to evaluate the security of the entire data path.

With the outstanding contextual interpretation capabilities of the GPT-3.5 and GPT-4 models [99] demonstrated, we decided to open in this direction and assess, with appropriate prompting and subtasking, how much deep and accurate information can be extracted from Angular-based web applications by simply examining the source code. Another branch of our research team has obtained promising results with GPT-assisted static code analysis of Angular applications [**?**], achieving better results on the problem set under investigation than with BERT, using only prompting and no additional training of models. Consequently, we investigated whether analogous advances could be made in a problem domain where a deeper understanding of the source code and the data handled by the program is required beyond semantic analysis. It is important to note that, although we chose Angular, the methodology presented in this paper should be easily adaptable to other frameworks to solve similar problems, as the basic pillars of the idea—mapping the code base, breaking down the vulnerability detection into steps and prompts, and then aggregating the partial results—are essentially programming language-agnostic steps that can be equally applicable to arbitrary backend and frontend frameworks.

The ”Improper Isolation of Compartmentalization” vulnerability, identified as CWE-653 in the Common Weakness Enumeration (CWE) database [76], is prevalent in frontend applications. This vulnerability for web applications means that certain critical processes or operations are implemented in the code with improper isolation; consequently, if the same piece of code is equally responsible for handling critical, sensitive data and less relevant data, there is a risk of significant data leakage or unauthorized use due to developer inattention or malicious user activity, as defined in CWE members of the CWE-200 (”Exposure of Sensitive Information”) category, with members that are easy to encounter in front-end applications such as CWE-203 (”Observable Discrepancy”), CWE-359 (”Exposure of Private Personal Information to an Unauthorized Actor”) and CWE-497 (”Exposure of Sensitive System Information to an Unauthorized Control Sphere”). The foundation of our approach was the identification of sensitive data, which should be accorded a high level of importance in the source code with appropriate isolation, protection, and access control; failure to do so can indeed result in significant misuse. In the case of progressive web applications, we have assumed that such data originates from a server or database connection, through which it is passed to the web application; therefore, the key to protecting it is the proper protection of the entities that manage these operations, in the case of the Angular framework, the Service classes, which in many cases implement the classic Data Object Access (DAO) pattern.

Such Services operate as singletons within the Angular framework and are accessed and invoked via injection from Component classes responsible for each element of the user interface or from each other. This paper's research questions evolved from the question of whether the Component classes that invoke these Services have appropriate access control, which in the instance of the Angular framework is controlled via the AuthGuards, classes implementing the CanActivate interface, which, when attached to the navigation rules within the application, can prevent or redirect a user from accessing user interfaces they are not authorized to access.

We were confronted with a significant problem when investigating this issue: can sensitive data be categorized, and can we quantify how sensitive the data and information are? It is indisputable that the technological advancements of the past decade, not to mention the various scandalous violations of personal data by large technology companies against their users, have raised an entirely new set of concerns regarding how we comprehend, handle, and expect our personal data to be handled. In the past few years, numerous studies have examined issues such as how we understand the concept of sensitive data in different domains [30], how the concept of personal sensitive data has evolved in the first place, and its various degrees [22].

Data collection parsimony, which sets a metric for machine learning algorithms on how little data they require for efficient learning and operation [66], has become a dominant trend, as have categorizing and quantifying the sensitivity of data [33, 96] and the automated recognition of sensitive data [26]. In the categorization we devised, the details of which are explained in greater detail in the Methodology section, we categorized the sensitive data into three categories based on the potential damage of a disclosure. While at the bottom of the scale we can only speak of potential damage through the accumulation of a large volume of leaked data, the data at the top of the scale is vital information of its own. Although the categorization of such a method set can be arbitrary for a given development team, we wanted to create a scale for large language models that is easily defined and can be described using the chain-of-thought principle.

The primary findings of this paper are as follows:

- Defining a proprietary classification to quantify the sensitivity and immunity of the data, and evaluating it on a test set of 200 variable names using the GPT API.

- Demonstration of prompts generated using prompt engineering for the GPT API for static analysis of the code of a complex web application, taking the larger context and deeper context into account.

- Evaluation of the efficacy of the GPT-3.5 and GPT-4 models in detecting sensitive data in an application via static code analysis.
- Evaluation of the ability of the GPT-3.5 and GPT-4 models to determine the protection levels of front-end application elements.
- Evaluation of the effectiveness of the GPT-3.5 and GPT-4 models in detecting when sensitive data handling in a web application is not adequately isolated and protected, thus producing a possible vulnerability.

We begin with a literature review in the Related Works section. Then, in the Methodology section, we present our methodology, in which we attempted to identify parts of the code handling sensitive data, categorize their sensitivity, determine their protection levels, and detect when the sensitivity level and protection level are inconsistent by executing an analysis pipe supported by GPT API calls on a set of open source Angular applications randomly selected from Github. In the Results and Discussion section, following the presentation of our results, we discuss the interpretation of the errors and anomalies found in the analysis as well as the improvements obtained by optimizing our method. In the Threats to Validity section, we discuss potential issues that threaten the validity of our results. Lastly, in the Conclusion section, we discuss the future research potential of our developed method.

4.2 Related Works

4.2.1 The impact of artificial intelligence on software development

The rapid growth in the use of artificial intelligence in programming did not follow the rise in prominence of GPT, but its potential was recognized by a large number of people long before that. In their 2019 paper, Jiang et al. [58] investigated how, for instance, they could enhance the readability of program code using machine learning to suggest method names based on their root.

Another research project, [59], CURE, strengthens the category of automatic program repair (APR), which aims to automatically detect, patch, and debug errors based on static program code. It has significantly exceeded the capabilities of similar existing solutions.

In the comprehensive review by Sharma et al. [102] in addition to categories such as program interpretation and classification of code quality, the identification of vulnerabilities, which corresponds to our research area, was highlighted.

Moment et al. [80] used machine learning to detect vulnerabilities and bugs in smart contracts used on the Ethereum chain, achieving 95% accuracy in their experiments in detecting a total of 16 different types of bugs and analyzing their results to demonstrate how much faster, more efficient, and simpler to use the resulting method was in comparison to other static parsers.

Mhawish and Gupta [78] concentrated on detecting code smells, and with the various machine algorithms evaluated and validated, they obtained over 90% accuracy in detection rate, which has tremendous potential for enhancing code quality and AI-based programming. Using artificial intelligence, Cui et al. [36] and Park and Choi [86] have studied the protection of Android-based systems in the IoT and self-driving vehicle domains, with the latter employing real-time scanning of network traffic to detect malware in the network in order to protect critical elements.

4.2.2 The code interpretation capabilities of GPT

The emergence and dissemination of GPT and other transformer-based neural networks, large language models, which have demonstrated exceptional results in code generation, code interpretation, documentation, and automated debugging, marked a turning point in this research domain.

Liu et al. [73] aimed to provide a comprehensive overview of researchers' use of ChatGPT and GPT-4 in their work. They based their analysis on a total of 194 papers found in the arXiv database. In their summary, they highlighted the potential of using the models and Reinforced Learning from Human Feedback (RLHF), as well as the numerous concerns regarding the ethical use of the models, their potential for producing detrimental content, and their potential privacy violations.

A comprehensive evaluation by Sarkar et al. [100] examined the efficacy of various LLMs for programming. They concluded that AI-assisted programming is an entirely new form of programming with its own challenges and complexities, and as part of their research, they considered, among other things, how proficiently a neophyte developer could program with the assistance of these tools.

Wei et al. [118] focused their research on the evaluation of the capabilities of LLMs, which surprised the research community as their emergence did not result from the scaling of the capabilities of lesser language models. According to them, one of the primary directions of NLP research is to determine how these emergent abilities will continue to change and evolve as the models continue to scale.

The research of Surameery and Shakor [105] has already centered on evaluating ChatGPT's ability to effectively repair flaws in response to prompts. While they believe that the effectiveness and limitations of ChatGPT will largely depend on the quality of the training data and the types of bugs to be fixed, ChatGPT will undoubtedly be a very important and relevant tool in this regard, and its strength will be bolstered by the use of other debugging and analysis tools as support.

Using predefined queries, Borji and Mohammadian [24] benchmarked the largest LLMs, such as GPT-4 and Bard, in their work. Based on their findings, the GPT-4 model appeared to be the most reliable for tasks centered on software development, code generation, and code comprehension, with great potential for usage in more complex scenarios.

4.2.3 Vulnerability detection with Large Language Models

Naturally, we are not the first to attempt to use the interpretive capacity of large language models to examine program code vulnerabilities.

In their 2021 literature review [120], Wu presented research on software vulnerability detection based on BERT and GPT. Common among the presented efforts was the segmentation of source code, followed by the extraction of features, and the fact that the introduction of specific vulnerabilities was one of the final fine-tuning steps. Thapa et al. [111] and Omar [83] attempted to detect vulnerabilities in C/C++ codebases using further training of GPT models and a similar methodology. The latter's detector, VulDetect, outperformed state-of-the-art tools on the tested datasets, which is an excellent illustration of the performance of large language models.

However, Sun et al. [104] pointed out in their own study that attempts such as the ones described above do not even attempt to exploit the ability of GPT-type models to interpret and evaluate domain-specific information and instead concentrate on vulnerabilities inferred from control and data flow analysis. However, they presented GPTScan, which searched the source code of smart contracts for logical inconsistencies with 90% accuracy.

Cheshkov et al. [32] have attempted to detect the most prevalent CWE vulnerabilities in open-source Java applications in a manner similar to ours by prompting GPT-3.5 and ChatGPT. However, their work was based on rather naive assumptions about the existing knowledge set of GPT; they only mentioned prompt engineering techniques in their conclusions, and their results were not particularly convincing, performing essentially on par with their benchmarking dummy classifier. They also assumed that the ChatGPT temperature level could be adjusted by prompts, but this has not been demonstrated to work, as it can be overridden by the system prompt used by ChatGPT alone, which is responsible for configuring the model's responses towards a more human-like interaction feel.

Feng and Chen [42] attempted a similar experiment with ChatGPT, although they have improved their approach by using chain-of-thought and few-shot example prompt engineering techniques, showing promising results in creating an Android Bug Replay automation tool supported by ChatGPT.

A comparison of the mentioned works with our own approach is given in the

Appendix, in Table B.1.

All of these studies demonstrate the potential of artificial intelligence in programming, specifically for detecting vulnerabilities and enhancing the quality and security of code.

4.3 Thesis II/1: Formal Categorization of Sensitivity and Protection Levels

I defined a taxonomy of sensitive data in front-end applications, which categorises them into three different categories based on the level of damage caused by their leakage and unauthorised access. I then presented another taxonomy, which divides the protection levels of the application components into categories such that they are representative of the protection required for the sensitivity categories.

Publications related to this thesis: [J4]

Our methodology is predicated on the notion that in Angular web applications, sensitive data is centralized in Services, which are accessed and utilized as singletons by Angular framework classes via their methods. To detect sensitivity levels and vulnerabilities based on static code, we had to first detect the elements that appear to be sensitive in the Component classes using the context handling capabilities of the GPTs and identify which of these elements are handled by Services.

To classify the sensitivity level of the data, we considered the attempts to categorize the data [33, 96] mentioned in the Introduction, but the relevant metric was the damage to the user in the event of a leak if unauthorized persons gained access to the data. Based on this, we divided the sensitive data into three levels. Motivated by the need to provide a categorization whose levels build upon one another, the classification of data follows a logical reasoning that allows us to debug the thought process of the large language model under consideration and improve or validate it in accordance with the chain-of-thought principle.

The categories are defined as follows::

- **Level 1 (Low Sensitivity)**: The accumulation and compilation of large quantities of data at this level is required to infer confidential information and create abuse opportunities. Sensitive information includes, among other things, a user's behavior history, a list of websites visited, products and topics of interest on a website, and search history.

- **Level 2 (Medium Sensitivity)**: By obtaining data at this level, it is possible to retrieve and compile potentially exploitable information. Solutions such as two-factor authentication, regular email and SMS notifications, and exhaustive logging in affected applications can mitigate the effects of a potential breach. This is the largest and most extensive group. It includes data such as, usernames, passwords, private records, political views, sexual orientation, IP address, physical location, and hectic schedules are examples of sensitive information.

- **Level 3 (High Sensitivity)**: The data at this level is sensitive in and of itself, and its acquisition or disclosure can have severe legal repercussions and cause extensive harm. Examples include health information, medical history, social security number, driver's license, and credit card information.

A formal definition of the categories is as follows:

- C marks the superset of all sensitivity levels. $C = \{L_1, L_2, L_3\}$
- E marks the various types of data, while D is the set of sensitive data $D = \{e \in E : \text{ if } e \in D_{L1} \text{ or } e \in D_{L2} \text{ or } e \in D_{L3}\}$
- O is the superset of the operations which can leverage sensitive data O = { A, R, I }
- For the operation A on E: $e \in D_{L1}$ if $A(e) = e' \in D_{L2}$
- For the operation R on E: $e \in D_{L2}$ if $R(e) = e' \in D_{L3}$
- For the operation I on E: $e \in D_{L3}$ if $I(e) =$ data leak

As stated in the Introduction, this categorization, which is based on a rationale for the level of data sensitivity for large language models, is anticipated to function well in arbitrary applications but may not be ideal for all development teams. In certain instances, it may require some refinement, perhaps by providing more specific examples for each level in the prompts. However, we anticipate that such fine-tuning will not be a challenge for GPT's capacity if our method proves to be functional and can efficiently detect and classify various sensitive data with appropriate justification.

Additionally, we have attempted to define the application's protection scale in a manner compatible with the three-level sensitivity scale. While the scale is based on the naive assumption that the data path to the frontend application is also equipped with various privilege management and unauthorized access prevention solutions, such as two-factor authentication, frontend applications are theoretically capable of integrating solutions that can prevent or at least reduce the likelihood of unauthorized access even if server-side and data path protection capabilities are insufficient.

The scale presented is as follows:

- **PL0**: The component has no protection at all.

- **PL1**: Bare-bones authentication: the application checks whether the user trying to access the resource is logged in to the application.
  - U represents a user.
  - S_U represents the session of user U.
  - $PL1(U)$ is satisfied if and only if S_U is valid.
- **PL2**: RBAC [46]: In addition to being logged in, the user has different roles that define their scope of privileges within the application.
  - R_U represents the set of roles assigned to user U.
  - P_R represents the set of privileges associated with a role R.
  - $PL2(U,R)$ is satisfied if and only if $PL1(U)$ is satisfied and R_U has the necessary roles to access the resource.
- **PL3**: ABAC [122]: In addition to the login and possible role scopes, other attributes such as time, physical location, or type/id of device used are checked before granting access.
  - T represents the time of access.
  - L represents the physical location of the user.
  - D represents the type or ID of the device used.
  - A_U represents the set of attributes associated with user U which could include elements from T, L, and D.
  - $PL3(U,R,A)$ is satisfied if and only if $PL2(U,R)$ is satisfied and A_U meets the required attributes for access.

In our experiments, we attempted to identify instances in which the AI-based analysis of the source code produced pairings in which these two values did not match, i.e., in accordance with the definition of CWE-653 failure, a Service dealing with sensitive data could be accessed and called by a Component with a protection level lower than the sensitivity level of the data due to a lack of proper isolation.

4.4 Thesis II/2: Sensitivity Analysis of Web Application Data and Components

I developed a GPT-enhanced methodology that uses the categorization defined in the previous thesis to detect sensitive data in front-end applications and, based on this data, to tag the code elements that focus on operations on such data. I validated the categorisation using an artificial intelligence-based evaluation that classifies the elements of a

200-word collection of variable names into one of the defined categories. I followed this by analyzing a total of 292 components from open source applications to detect sensitive data and based on that, identify the services handling sensitive information in the application.

Publications related to this thesis: [J4]

4.4.1 Variable Dictionary Evaluation

To validate the categories developed and the analysis capabilities of the GPT-3.5 and GPT-4 APIs, which at the time of writing were the most efficient GPT models available and promptable via API, we compiled a dictionary of 200 variable names, equally divided among the Non-Sensitive, Low Sensitivity, Medium Sensitivity, and High Sensitivity categories. In preparing the dictionary, we intentionally made it difficult for the models to achieve perfect accuracy: some variant names had deliberate misspellings, others were in foreign languages such as Hungarian, German, or Italian, and others contained a combination of a more sensitive word and a less sensitive ending.

The GPT-3.5 API incorrectly classified 45.5% of the dictionary, with the majority of cases being trivial. In contrast, GPT-4 only made errors in 23.5% of the cases, and a deeper examination of the errors revealed that it only failed on the explicitly difficult terms, and in more than half of the cases, it correctly identified the errors as sensitive data but incorrectly categorized the exact category. Although both models were used throughout our investigations, preliminary results suggested that GPT-4 might be able to produce more accurate results by analyzing codes, whereas GPT-3.5 appeared more prone to failure in all but the most straightforward cases.

Our research team collected thousands of public Angular projects from GitHub using a crawler algorithm in order to conduct various source code analyses. We minified the TypeScript source files of the projects by labeling, removing whitespace characters, and removing line breaks, and then randomly sampled 12 of the largest, most complex, and largest projects that include 292 Components with mixed complexity and sensitivity. If the methodology proves effective on such large and complex codebases, we anticipate that fine-tuning the prompts may be sufficient to handle potential anomalies, while the preparation of the codebase, the validity of our methodology, the outlined steps, and the ability of the GPT models to interpret static code will remain unchanged while being able to handle specific cases. The random selection was modified with the constraint that the 12 selected projects must contain non-English terms, variable names, and class names to support the hypothesis, confirmed during the dictionary evaluations, that GPT can effectively detect sensitive information regardless of its language, given the appropriate context. The following evaluation concerned the effectiveness of the detection of sensitive Services from these test projects. The results of GPT-4 and GPT-3.5 are shown in Table 4.2. From the results, it is clear that although GPT-3.5 marked significantly more data as sensitive in the source codes tested, it did so incorrectly in many more cases compared to GPT-4. Incorrect Detections counts cases where the detection was incorrect; perhaps a function or the Service itself was marked as sensitive data instead of a variable or object; Duplicate Detection counts cases where the same sensitive data was marked more than once within the same file; and Insufficient Detection counts cases where the sensitive data was correctly detected but the justification for the sensitivity was incomplete or incorrect. Next, the Services that served as the source or target for the sensitive data and objects were identified. The results can be seen in Table 4.1, and although GPT-4 was clearly more efficient here, it made a significant number of errors in consistently detecting sensitive Services. In the table, Undetected Sensitive Services is the number of Services that, although using sensitive data, were not detected, Incorrect Sensitivity Levels is the number of Services that were not assigned a correct sensitivity level even once, and Missing Information is the number of Services that were detected, for which some key information, such as the exact name of the Service, was not detected but its presence was, while the Insufficient Detection Count is the number of Services that were successfully detected but not as many times as they occurred in the Components of the application.

4.4.2 Sensitivity Detection of the Test Suite

Table 4.1: *Results of sensitive Service identification by the two GPT models*

	GPT-4	**GPT-3.5**
Detected Sensitive Services	37	28
Undetected Sensitive Services	17	26
Incorrect Sensitivity Levels	2	17
Insufficient Detection Count	23	24
Missing Information	3	5

A very common example during the analysis of errors was the violation of the SOLID principle [77] defined by Robert C. Martin and interpreted in programming languages based on an object-oriented paradigm: instead of being responsible for managing a single well-defined resource, Services often managed a multitude of similar operations. For example, it was a common phenomenon in selected projects that all database operations, regardless of the tables or collections manipulated, were performed by a single Service, which could then make a single ill-parameterized or insufficiently tested function call to attempt to retrieve data that a significant proportion of users would not have had access to on a relatively unprotected interface. Another part of our research team has demonstrated, through the development of a library [65], how a more precise, resource-based allocation can have additional productivity benefits. Such a lack of resource isolation is in fact a direct practical implementation of the CWE-653 vulnerability, as illustrated by the fact that the Insufficient Detection Count for GPT-4 and GPT-3.5 is almost identical.

Table 4.2: *Results of sensitive data detection by the two GPT models*

	GPT-4	**GPT-3.5**
Sensitive elements	363	375
Incorrect detection	7	128
Duplicate detection	25	23
Insufficient detection	0	29

The three most common types of bad development practices that occurred most frequently during the evaluation and led to errors were:

- **Injection Circumvention**: Although Angular formally documents the exact role and intended use of Services, a recurring problem in debugging has been that developers have circumvented the development pattern of injecting and then invoking methods. There were cases where Services saved sensitive data in localStorage and the Components of the application later read it from there, and one case where Services used EventEmitters to pass sensitive values. This resulted in Missing Information cases in which the presence of Services was detected but not their precise identities because they were not used in a conventional manner.

- **Monolith Service**: The most common issue was the lack of isolation mentioned above, a textbook example of the CWE-653 vulnerability, where Services are not organized around a resource but around a type of operation. For example, there is an entity called DbService, which is responsible for the database operations of registration, login, shopping, commenting, and private messaging at the same time. This flaw led to very high Insufficient Detection Count values, where the Service was identified either as sensitive or not, depending on which of the many resources it was currently managing.

- **Delegated Responsibility**: This issue refers to instances in which a Service was initially used correctly, but then the data it handles was stored and transmitted in the application differently, such as by passing parameters between parent and child components or by creating a separate Component or other object that acted as a session database.

These error possibilities, while to some extent mitigatable, have also led to more severe issues later in the evaluation process.

4.5 Thesis II/3: Protection Level Analysis and Vulnerability Detection for Web Applications

I defined a GPT-supported static source code analysis pipeline that uses the protection level categorisation to identify the protection level of components in a frontend application and then uses the results from the protection level and sensitivity level evaluations to detect vulnerabilities where components are not protected or whose protection level is insufficient for the sensitivity of the data they handle, in line with the CWE-653 type software vulnerability.

Publications related to this thesis: [J4]

4.5.1 Evaluation of the Protection Level Prompt

As far as the detection of the Components' protection level is concerned, the GPT's task here was more of an aggregation operation than an operation requiring deep interpretation. The results of the JSON Mapping and Protection Level Discovery steps for the total of 292 Components examined are shown in Table 4.3.

Table 4.3: *Results of sensitive Service identification by the two GPT models*

	GPT-4	**GPT-3.5**
Components with Incorrect Service Detection	0	23
Components with Incorrect AuthGuard Detection	14	96
Components with Incorrect Protection Level	4	112

In addition to the results presented in the table, the majority of Incorrect Auth-Guard Detection issues for GPT-4 were caused by the nesting of parent and offspring modules within an application, which made it difficult to detect AuthGuards accurately. During the investigations, the maximum level of AuthGuard was detected in the majority of cases, resulting in only four cases of inadequate protection level out of 292 investigated cases. When evaluating the results of GPT-3.5, we had to confirm the findings of Cheshkov et al. [32] because, despite employing prompt engineering techniques and setting the temperature of the models to 0, we encountered difficult-to-explain problems. In the case of Incorrect Service Detection, errors occurred where interfaces and elements not acting as Services, such as elements of the Angular Material library or the Observable interface, were marked as Services; during AuthGuard Detection, despite a temperature of 0, hallucinations occurred and generated Auth-Guards that did not exist in the application and either left their associated protection level at 0 or changed it to 1 or 2, causing 38% of the Components to have incorrect protection levels.

Thus, although we used the GPT-3.5 models for completeness in the rest of the evaluation, it was guaranteed that the aggregate results of Vulnerability Detection would not be adequate for errors of this magnitude, and the errors it made would be difficult to correct even with prompt engineering.

4.5.2 Definition of the Detection Pipeline

With the evaluations of the individual prompts complete, it is now time I introduce the pipeline which integrates their results to complete the evaluation of the projects from the test set. The pipeline's prompts were developed using the principles of prompt engineering [79], a relatively new discipline, through multiple cycles of experimentation; the used prompts are included in the Appendices of this paper. Criteria included plain wording, the inclusion of rules prohibiting the reappearance of anomalies discovered during initial testing, and the inclusion of examples pertinent to the expected response format. To accomplish this, we used larger, more comprehensive prompts, which allowed us to avoid both format errors and the inclusion of uptake prompt engineering techniques such as ”few shot example”, where each prompt was provided with at least one sample input and a corresponding sample output, and chain-of-thought, where we provided the thought and logic flow in the prompts in addition to simple rules, which helped to provide the correct deductions and avoid various errors. Although this significantly increased the size of the prompts and we were initially concerned about exceeding the 8096 token limit of GPT-4 (in the case of GPT-3.5, we were not at risk of exceeding it due to the 16,384 token limit of gpt-3.5-turbo-16k), a survey conducted prior to the actual evaluations to measure the maximum size of the prompts and input files dispelled such concerns. The results are shown in Table 4.4, and the prompts and precise methodology are described in the sections that follow.

Table 4.4: *The token counts of the various prompt parts used in the GPT API requests*

Prompt Type	Token Count
Senstivity Detection Prompt (Appendix B.2.1)	905 tokens
Feature Extraction Prompt (Appendix B.2.2,B.2.3)	1564 tokens
AuthGuard and Protection Level Evaluation Prompt (Appendix B.2.4)	2354 tokens
Average token count of the largest files from the largest projects	3560 tokens
Average token count of the largest mapped json	1145 tokens

In addition, it is important to note that, at the time of writing, GPT-4 has already begun to implement its extended model with 32,000 tokens, which will permit the delivery of significantly larger code segments or even more complex prompts than those under consideration, thereby mitigating this risk. [13]

The complete analysis process consists of the following steps:

1. **Minifying Code Base**: The .ts source files of Angular applications have been minified, removing line breaks and whitespace characters.

2. **Sensitive Element Detection**: The minified Component files were passed one by one to the GPT API, embedded in the Sensitivity Detection Prompt included in the Appendix B.2.1. The analysis first identified the sensitive data, explaining how it appears in the application, how it is used, and then an aggregation script used this information to select the Services that were involved in read or write operations on sensitive data in the project. For each occurrence of the tagged Services, the sensitivity level of the data they handle is also determined.

3. **JSON Mapping of Project Files**: The minified Component files were passed one by one to the GPT API, embedded in the Feature Extraction Prompt (Appendix B.2.2,B.2.3). As a result of this step, a JSON file is created for each project, in which the components of the application are reduced to JSON objects, containing only the information needed for the current (or planned future) tests, such as the parent-child component relationships, the injected Services, and their used data members and operations.

4. **Protection Level Discovery**: The JSONs resulting from the previous step, along with the minified Router configurations and AuthGuards, were passed one by one to the GPT API embedded in the AuthGuard and Protection Level Evaluation Prompt (Appendix B.2.4), which added the protection level corresponding to our scale. A Python script then unified these protection levels, assigning the child components the protection level of their parent components.

5. **Vulnerability Detection**: By aggregating the results of the Protection Level Discovery and Sensitive Element Detection steps in an xlsx file, the vulnerabilities from the tested projects are detected, the cases where a Component using a sensitive Service does not have a high enough protection level.

The results were then checked manually. More detailed explanations and outputs of the steps are presented in the following subsections.

JSON Mapping of Project Files and Protection Level Discovery

The purpose of this step is to reduce the source code of the projects under investigation to the most relevant elements for the investigations, thereby reducing the number of tokens used in subsequent operations and enhancing the quality of the evaluation by passing only the most pertinent information from the files to the subsequent prompts. The step is essentially equivalent to the code reduction/feature extraction phase [83, 104, 111], which has been referenced numerous times in the literature, with the output format described in the Appendix, in Table B.2.

During the JSON Mapping of Project Files phase, the Guarded, Guards, and GuardLevel fields continue to be populated with empty values. During the Protection Level Discovery step, the already mapped json files are passed along with the AuthGuards and routing configurations to the GPT API in order to determine the protection levels based on the interpretation of the AuthGuards functionality and mappings.

Our goal with the format was also to ensure that if we were to take our research beyond security risk analysis into other directions with static analysis in the future, the methodology could be used for further studies and experiments and ultimately form the basis of a different type of schematic representation of the codebase, for example, a visual representation including dependencies, possible navigations, and security configurations, similar to the visual description languages [49, 51] mentioned in the Introduction chapter.

Code 4.5.2 is an example result of the json mapping operation.

```
{
    "type": "Component",
    "selector": "profile-comp",
    "name": "ProfileComponent",
    "file": "profile.component.ts",
    "path": ["/profile"],
    "guarded": 0,
    "guards": [],
    "guardLevel": 0,
    "navigations": ['/dashboard'],
    "injections": {
        'OAuthService':
                {'Functions': ['checkAuth'],
                'Variables': [],
                'Asynchrons': ['token']},
        'LoggerService':
                {'Function': ['logInfo', 'logError'],
                Variable': ['log'],
                Asynchron': []}
    },
    "parents": [],
    "children": []
}
```

Sensitive Element Detection

As input, the step also receives the component's minified source code, which is embedded in the corresponding prompt. The intent was for the GPT API to mark in the received source code the variables and objects that it identified as sensitive, classify them according to the sensitivity level identified, and provide explanations for tracing back the findings in the event of a problem. The output format, shown in the Appendix, in Table B.3, which this time used csv format to simplify manual validation, provided a way to do this by using the Type field, where GPT could write a short, few-word category identifying exactly what type of data the information marked as sensitive was (e.g. Confidential User Data, Company Information, Financial Information), and the Function field, where a much more detailed description could be provided explaining the reasons for marking the information as sensitive (e.g. "This is the device data collected for payment processing", "Array of Materia objects containing educational records").

Vulnerability Detection

A simple aggregating Python script that read the json files containing the protection letters and the csv file containing the list of sensitive Services identified the vulnerabilities based on these files. In the output, it enumerated all of the application's Components, their protection level, the sensitive services detected, and the maximum level of sensitivity among them. If a Component's protection level was lower than its sensitivity level, it was marked as potentially vulnerable. The results were then manually inspected.

4.5.3 Evaluation of the Vulnerability Detection Results

The Vulnerability Detection step was performed based on the aggregation of the two sub-results, and the success of this step is shown in Table 4.5.

Table 4.5: *Results of the Vulnerabitily Detection.*

	GPT-4	**GPT-3.5**
Detected Vulnerability	40	17
False Vulnerability	0	8
Undetected Vulnerability	49	80
Undetected High Sensitivity Vulnerability	10	16
Undetected Medium Sensitivity Vulnerability	31	56
Undetected Low Sensitivity Vulnerability	8	8

As can be seen, our assumptions about GPT-3.5 have been confirmed, accumulating the poor results of the previous steps and resulting in a total of only 1% of vulnerabilities successfully detected. However, our initial naive assumption about the identification of sensitive Services had very serious consequences for GPT-4 in the first round. Of the total number of errors that should have been identified, only less than 45% were successfully detected, and 20% of the errors (i.e., cases where a vulnerability should have been detected but was not) were in the High Sensitivity category, meaning that attempts to retrieve or manipulate highly sensitive health, financial, or other personal data could theoretically be made through the frontend from code in an interface that was not nearly strong enough to handle such data. A significant part of the problem was identified in the RQ1 investigation as the Monolith Service, where Services did not have one well-defined task and role but rather developers crammed several similar procedures into them without any consideration that all levels from Non-Sensitive to High Sensitivity data categories could appear within a single Service. This problem caused the sensitivity level of the Services to vary enormously in many cases, where a Service was identified as having either

Medium or High Sensitivity in a single Component, while in another it was declared irrelevant to sensitive data.

To improve the detection results somewhat and address these issues, a major strengthening of vulnerability detection was applied to GPT-4, whereby if a Service was categorized as handling sensitive data based on its behavior in any Component of the application, it was treated as sensitive in the scope of the whole application and assigned the highest sensitivity level from any affected Component. A very spectacular improvement in the results can be seen in Table 4.6.

Table 4.6: *Comparison of the original and more strict Vulnerability Detection rules.*

	GPT-4 Original	**GPT-4 Strict**
Detected Vulnerability	40	79
False Vulnerability	0	0
Undetected Vulnerability	49	10
Undetected High Sensitivity Vulnerability	10	0
Undetected Medium Sensitivity Vulnerability	31	3
Undetected Low Sensitivity Vulnerability	8	7

The detection success rate has jumped from 45% to 88.76%, and the High Sensitivity category misses have completely disappeared. In other words, with this tightening, which could have been done in the Sensitive Element Detection step as a post-processing step, we achieved an improvement of 44%. The remaining anomaly is expected to be overcome by further refining the data and Service sensitivity prompts in light of these results, and the increased token limit will allow GPT-4 to examine the different data used in an even wider context. In the case of GPT-3.5, although some improvement may occur with similar tightening and further refinement of the prompts, misinterpretations and anomalies such as hallucinated AuthGuards and high misinterpretation in the detection of sensitive data strongly question the usefulness of the model for such purposes and are in line with the observations of Cheshkov et al. al. [32], who obtained better results even with a dummy categorizer than with GPT-3.5. While the results obtained with GPT-4 support the conclusion of Borji and Mohammadian [24] that GPT-4 is possibly the most promising LLM currently available for software development and source code interpretation tasks.

4.6 Threats to Validity

Conclusion Validity: Although we have attempted to select 12 of the largest and most complex available projects, there may be others whose structure and unique solutions pose new challenges. However, since our results demonstrate that large language models—GPT-4 especially—are capable of interpreting code at a sufficient level by following the examples and thought processes defined in the prompts, we hypothesize that these individual challenges can be overcome using the same methodology and by fine-tuning the rules defined in the prompts.

Internal Validity: We have not identified any threat to internal validity.

Construct Validity: As pointed out during evaluations, if the developers of the evaluated project significantly deviated from the Angular framework's development conventions, for example by using monolithic Services or other questionable code quality solutions or by overcomplicating parent-child relationships, the likelihood of GPT-4 misinterpreting or having trouble detecting contextual relationships may also increase. The 44% improvement obtained by refining the criteria at the conclusion of the Vulnerability Detection phase demonstrates that such problems can be mitigated by augmentation and data cleansing steps following the static source code feature extraction step. On the other hand, the problems identified in the RQ1 evaluation would most likely be considered code smells or vulnerabilities of a type that cannot be detected without a deeper understanding of the code base. Their mere presence has a negative impact on code quality, code interpretability, and code security, so their detection alone could be useful to development teams interested in static analysis and evaluation of their source code.

External Validity: Although, when defining the categories of data sensitivity and protection, we attempted to create a general taxonomy centered on the potential damage caused by data leakage, which we assumed would be easy to interpret and follow by AI, which we achieved while also revealing more complex problems through the investigation of the vulnerabilities in the source code, there may be instances in which a specific sensitive data type needs to be detected, requiring a different scale. Nevertheless, as we suggested in the Conclusion Validity threat assessment, since our methodology with the GPT-4 API performed well overall in the evaluations, according to our hypothesis based on the results so far, these assumptions require mere modifications to the content of the prompts.

In the future, we plan to both further validate our work, either with the C/C++ datasets [83, 111] discussed in the survey literature - after all, the underlying concept of our methodology does not depend on the specific source code and framework - or fine-tune it to explore further areas of research, such as navigation, as a result of which we might at some point even enable a new kind of dynamic, artificial intelligence-based visual description language for code bases where this has so far proved impossible.

4.7 Chapter conclusion

Our team has provided an overview of the primary directions in which artificial intelligence is increasingly being used in programming. I defined a categorization for determining both the nature of sensitive data and the application's vulnerability, and developed a process based on the GPT API to detect the CWE-653 vulnerability and its consequences, the data leakage types of the CWE-200 family. Although GPT-3.5 underperformed in the model evaluation, GPT-4 obtained 88.76% accuracy, confirming our hypotheses regarding the static code analysis and interpretation capabilities of large language models. We hope that our research has contributed to a deeper understanding of the potential of artificial intelligence in software engineering and that the methodology presented here can facilitate the design and implementation of code analysis techniques that can significantly contribute to the analysis of more complex source code, to improve its quality and vulnerability detection for development teams, and to the promotion of similar directions for fellow research groups.

The author of this PhD thesis is responsible for the following contributions presented in this chapter:

II / 1. I defined a categorisation of sensitive data in front-end applications, which categorises them into three different categories based on the level of damage caused by their leakage and unauthorised access. I presented a further categorization, which divides the protection levels of the application components into categories such that they are representative of the protection required for the sensitivity categories.

II / 2. I presented a GPT-enhanched methodology that uses the categorization defined in the previous thesis to detect sensitive data in front-end applications and, based on this data, to tag the code elements that focus on operations on such data. The author validated the categorisation using an artificial intelligence-based evaluation that classifies the elements of a 200-word collection of variable names into one of the defined categories. The author followed this by presenting the results obtained on the sensitive data detection from a total of 292 components from open source applications.

II / 3. I defined a GPT-supported static source code analysis pipeline that uses the results of the previous thesis to detect vulnerabilities where components are not protected or whose protection level is insufficient for the sensitivity of the data they handle, in line with the CWE-653 type software vulnerability.

Chapter 5

Conclusions

5.1 Results

While the data privacy requirements of the e-health data path are complex and demanding, during my years of research I have managed to explore and analyze several aspects of it, and propose significant results and solutions for several of its challenges. I defined a taxonomy distinguishing the four policy categories that together can encompass the access control requirements; I have separated two types of edge nodes based on their capabilities: processing edge and caching edge. As a member of a research team, I have examined the stability of smartphone-based peer-to-peer networks, and came to the conclusion that further testing and simulations are required to determine whether they can implement a local e-health infrastructure as well. I defined test rules for each policy category, established test environments for the edge types. With the aid of the Open Policy Agent (OPA), I implemented my proposed access control solution based on the Policy Enforcement Point (PEP) principle, whose performance I evaluated in both test environments with an ever-increasing volume of documents. The results showed that the latencies caused by my access control solution accounted for less than 35% of the total runtime on the processing edge, even with the largest amount of data, while the proportion was less than 30% on smaller data amounts; on the caching edge the delays were much more significant, as long as only a few hundred documents were handled, the method proved to be effective on both desktop PCs, smartphones, and tablets. To measure the exact effects of the delays caused, I have developed an open-source patient flow simulator that makes it possible to construct scenarios, parameters, and thresholds that enable more accurate validation of access control implementations.

Building upon the recent breakthroughs in the domains of artificial intelligence and natural language processing (NLP), I have also tackled the challenge of static code analysis of the web applications at the ends of the data path, whose dynamic and complex nature has made several types of vulnerability analysis nearly impossible until now. I defined taxonomies for determining both the nature of sensitive data and the application's protection levels and developed a process based on the GPT API to detect possible points of data leakage using techniques of prompt engineering instead of classical training. I demonstrated the potential of the GPT models in classifying the sensitivity levels of a dictionary of variable names and then in detecting sensitive data elements and critical services in open source web application projects using their ability to interpret and analyze the meaning and context of the program code. After evaluating the protection level of said services, I have also demonstrated the effectiveness of my methodology in detecting points of potential data leakage—points in the source code where the sensitivity of the handled data is unmatched by the protection level.

5.2 Future Work

It goes without saying that the results of my years of research only encompass a portion of the telemedicine data path, which only demonstrates the complexity of the data privacy issues in e-health. To support the ever-evolving e-health technologies, we must, however, establish comprehensive solutions and address the entire scope of the problem.

Implementing the OPA-based access control solution at even more locations along the data path in order to determine where and what kind of access control evaluations are most effective is the next step in my research. If these results are validated with simulations, I may be able to define a set of established patterns and anti-patterns of access control along the telemedicine data path in the near future. This would be a significant step towards a standardized security approach in the modern e-health domain.

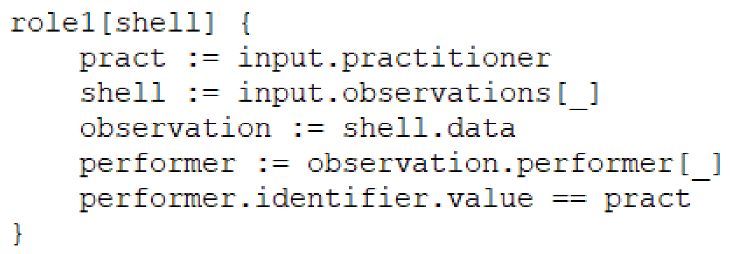
In order to accomplish this, I also intend to enhance the patient flow simulation tool by adding even more detail, options, and detailed logging options, and by adapting it to even more intricate scenarios. In the future, it will be necessary to combine it with a network simulation tool utilizing peer-to-peer data from our Stunner research project in order to evaluate the applicability of smartphone-based serverless peer-to-peer networks in specific scenarios.

At the conclusion of the data path, we intend to evaluate additional projects using my GPT-based data leakage detection methodology. Experimenting with smaller, less-demanding models trained specifically for this problem; providing increasingly detailed evaluation context with the GPT models' ever-expanding token limits; and experimenting with other vulnerabilities from the CWE database that, like CWE-653, require contextual information to evaluate and, as a result, were not detectable using only traditional static code analysis tools.

Appendix A

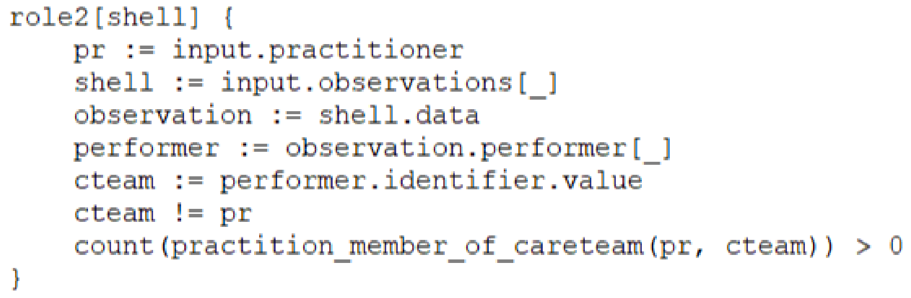
Supplementary Algorithms and Tables for Vulnerability Detection

A.1 Algorithms



```
role1[shell] {
    pract := input.practitioner
    shell := input.observations[_]
    observation := shell.data
    performer := observation.performer[_]
    performer.identifier.value == pract
}
```

Figure A.1: *A Role Evaluation policy in Rego returning only the Observations where the current user is the Practitioner*



```
role2[shell] {
    pr := input.practitioner
    shell := input.observations[_]
    observation := shell.data
    performer := observation.performer[_]
    cteam := performer.identifier.value
    cteam != pr
    count(practition_member_of_careteam(pr, cteam)) > 0
}
```

Figure A.2: *A Role Evaluation policy in Rego returning only the Observations where the current user is member of one of the CareTeams, who received access from the Patient*

```
contextual1[shell] {
    timearray := time.clock([time.now_ns(),
        "Europe/Budapest"])
    hour := timearray[0]
    shell := observation_of_pract[_]
    shell.data.status == "active"
    hour >= 8
    hour < 19
}
```

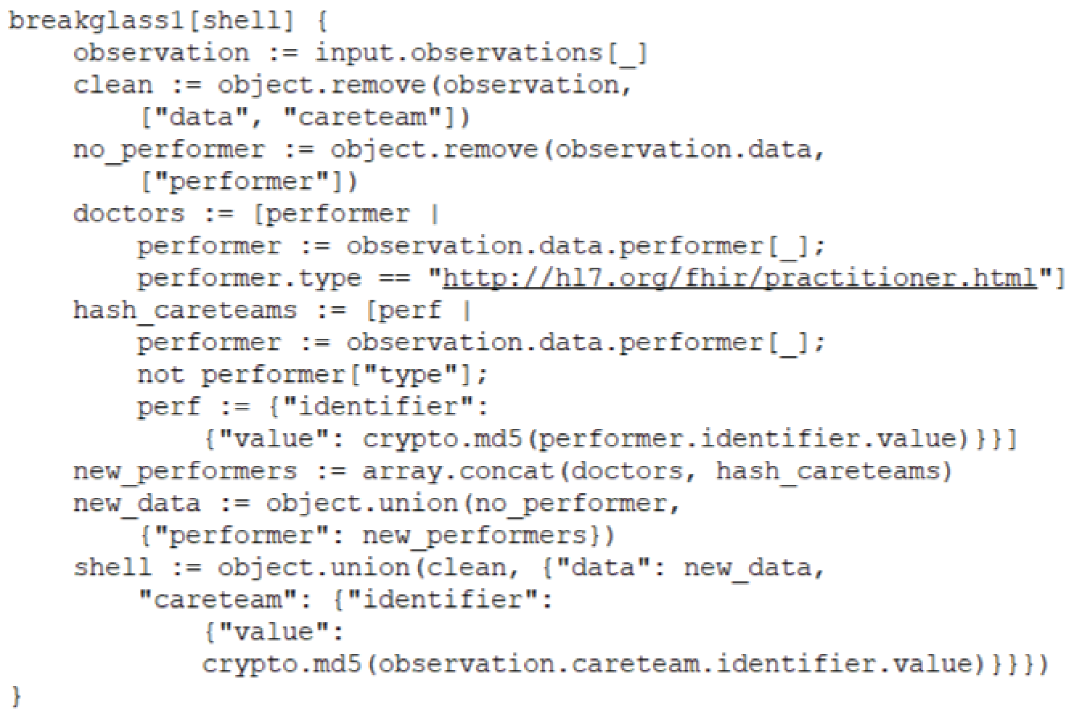
Figure A.3: *A Contextual Evaluation policy where access is granted only if the status of the Observation is active and the time of access takes places between 8:00 and 19:00*

```
modification1[shell] {
    observation := input.observations[_]
    no_outside := object.remove(observation,
        ["patient", "data"])
    no_inside := object.remove(observation.data,
        ["subject"])
    shell := object.union(no_outside,
        {"data": no_inside})
}
```

Figure A.4: *A Contextual Modification policy where we remove the Patient identifiers from Observation*

```
modification2[shell] {
    observation := input.observations[_]
    clean := object.remove(observation,
        ["data"])
    empty_observation := object.remove(observation.data,
        ["component"])
    new_components := [component |
        component := observation.data.component[_];
        component.code.coding[0].code != "18748-4"]
    new_observation := object.union(empty_observation,
        {"component": new_components})
    shell := object.union(clean, {"data": new_observation})
}
```

Figure A.5: *A Contextual Modification policy where we remove a specific nested component from every Observation*



```
breakglass1[shell] {
    observation := input.observations[_]
    clean := object.remove(observation,
        ["data", "careteam"])
    no_performer := object.remove(observation.data,
        ["performer"])
    doctors := [performer |
        performer := observation.data.performer[_];
        performer.type == "http://hl7.org/fhir/practitioner.html"]
    hash_careteams := [perf |
        performer := observation.data.performer[_];
        not performer["type"];
        perf := {"identifier":
            {"value": crypto.md5(performer.identifier.value)}}]
    new_performers := array.concat(doctors, hash_careteams)
    new_data := object.union(no_performer,
        {"performer": new_performers})
    shell := object.union(clean, {"data": new_data,
        "careteam": {"identifier":
            {"value":
            crypto.md5(observation.careteam.identifier.value)}}})
}
```

Figure A.6: *A Break-the-Glass policy where we hash the identifier of the CareTeam in the Observations*

```
breakglass2[shell] {
    observation := input.observations[_]
    clean := object.remove(observation,
        ["data", "careteam",
        "basedOn", "practitioner", "patient"])
    removed_body := object.remove(observation.data,
        ["performer", "subject", "basedOn"])
    new_body := object.union(removed_body,
        {"subject": {"display":
            crypto.md5(observation.data.subject.display),
        "identifier": {"value":
            crypto.md5(observation.data.subject.identifier.value)}},
        "performer": [perf
            | performer := observation.data.performer[_];
        perf := {"identifier":
            {"value":
                crypto.md5(performer.identifier.value)}}]})
    shell := object.union(clean, {"data": new_body,
        "patient": crypto.md5(observation.patient),
        "practitioner": crypto.md5(observation.practitioner),
        "careteam": {"identifier":
        {"value":
        crypto.md5(observation.careteam.identifier.value)}}})
}
```

Figure A.7: *A Break-the-Glass policy where we hash every identifier for Patients, Practitioners and CareTeams*

A.2 Tables

Table A.1: *Major events of the main ED flow*

Name	ID	Description	Type	Req
Patient Registration	register_patient	Admitting the patient to the hospital	event	HE
Evaluate Urgency	evaluate_urgency	The urgency, as a binary information is determined for the patient. Urgent cases are immediately forwarded to pre-visit	event	GN
Abandon	abandon	The patient decides to leave the hospital before pre-visit	end_state	-
Pre-visit	pre_visit	Quick measurements and examination by a nurse to determine the triage level and severity of the case.	event	GN
Assign ESI	assign_esi	Emergency severity index is assigned to the patient, who is either transferred to internal clinic or sent to the full visit process	event	GN
Manage outcome	manage_outcome	Management of results and outcomes resulting from the visit process.	event	SN

Table A.2: *Elements of the Visit subprocess*

Name	ID	Description	Type	Req
Collect History	collect_history	The nurse collects and organizes the healthcare history of the patients via direct interview with the patient and/or database queries	event	SN

Table A.2: *Elements of the Visit subprocess*

Name	**ID**	**Description**	**Type**	**Req**
Hypothesize Diagnosis	hyp_diag	The doctor evaluates the results so far, examines the patient and either establishes a diagnosis or may require further tests (blood tests and X-rays) before doing so.	event	DR
Take Blood Sample	take_blood	Taking a blood sample as prescribed by the doctor.	event	SN
Laboratory	laboratory	The blood sample is transferred to laboratory examinations. If there is any information available quickly, it is sent back to the doctor.	event	DR
Transfer to radiology	trans_rad	The patient is transferred to radiology and prepared for the X-ray recordings.	event	GO
Radiology	radiology	Usage of diagnostic imaging procedures, such as ultrasound, CT, MR.	event	SN
Establish diagnosis	estab_diag	Based on the results and the information, diagnosis is established	event	DR
Define therapy	def_therap	If possible, therapy is defined by the doctor	event	DR

Appendix B

Supplementary Algorithms, Prompts and Tables for Vulnerability Detection

B.1 Tables

Table B.1: *The list of the related works with the most relatable solutions to our own research, and the differences with our planned methodology.*

Work	Comparison with our research
Wu [120]	Our approach is predicated on the premise that for models such as GPT-3.5 and GPT-4, due to the large number of diverse resources used in their training and their contextual interpretation capabilities, teaching is not required; rather, it is sufficient to refine and transfer the additional knowledge required to complete the task during prompting. In our case, pre-processing and extraction of the code base serve as a comparable starting point.
Thapa et. al. [111] & Omar [83]	Both studies searched for vulnerabilities in C/C++ source code using GPT models with a similar processing principle to ours, with the exception that we rely on prompting rather than further training and focus on a problem that requires a deeper comprehension of the code's meaning.

Table B.1: *The list of the related works with the most relatable solutions to our own research, and the differences with our planned methodology.*

Work	**Comparison with our research**
Sun et. al. [104]	They were searching for vulnerabilities in smart contracts and, consistent with our ideas, understood that GPT has tremendous potential to address problems that require deeper meaning and context interpretation than semantic and logical interpretation of the control and data flow.
Cheshkov et. al. [32]	The prompts were based on ChatGPT, which, at the time of writing our paper, did not provide a formal method to set the temperature and accomplish determinism in responses. The presented prompts are rudimentary, and the use of prompt engineering techniques is only mentioned as a potential direction for future research at the conclusion of the paper.
Feng & Chen [42]	They have also attempted to fine-tune a previously trained model solely through prompting, and due to the prompt engineering techniques that we have also used to develop our own prompts, they have achieved quite encouraging results.

Table B.2: *The output structure of the json_mapping operation.*

Name	**Content**
Type	The type of the analyzed file, Component, Service, etc.
Selector	If the type is Component, this field contains selector string specified in the decorator method of the class, which indicates the rendering locations in the html template codes which are handled by the Component..
Name	The name of the analyzed class.
File	The name of the analyzed file.
Path	If the parsed file type is Component, the string of the application's routing configuration to access it; empty string for embedded child components only.
Guarded	Boolean value indicating the presence of an AuthGuard or AuthGuards protecting the class if it is a Component.

Table B.2: *The output structure of the json_mapping operation.*

Name	Content
GuardLevel	A numeric value in the 0 - 3 interval according to the security levels defined in the previous section.
Guards	The names of the AuthGuard classes protecting this class if it is a Component.
Navigations	A list containing the names of the Components to which we can navigate forward from this class.
Injections	A list of injected Services in the class, also containing the exact interactions with them which might be Asynchrons, Attributes or Functions-
Parents	The list of Components, which contain the selector of this class (if it is a Component) in their html template-
Children	If the analyzed file is a Component, then a list of the Components, that are contained in its html template by their selectors.

Table B.3: *The output structure of the sensitivity_detection operation.*

Name	Content
Element	The name of the sensitive object or variable.
Type	The type of the sensitive element, following a conviction similar to the list in prompt B.2.1.
Level	The sensitivity level of the detected element.
Origin	UserInput, Service or LocalStorage, the three most common sources of sensitive data in a web application.
ServiceName	If the Origin is Service, then the name of the Service which is the source of the sensitive element.
Function	The role of the sensitive element in the Component where it was detected.
Goal	The fate of the sensitive element after its usage: Stored, Read or Service.
GoalName	If the value of the Goal field was Service, then the name of the Service to which the sensitive element is passed to.
Classname	The name of the class in which the sensitive element was detected.

B.2 Prompts

B.2.1 The prompt for detecting sensitive elements in the Component code

Inspect the following Angular code, list all variables or attributes that could contain sensitive information for the user (Usernames, Passwords, Email Addresses, Physical Addresses, Phone Numbers, Birthdates, Social Security Numbers, Credit Card Numbers, Bank Account Numbers, Routing Numbers, Driver's License Numbers, Passport Numbers, Health Insurance Numbers, Medical Records, IP Addresses, Biometric Data, Purchase History, Tax IDs, Company Confidential Information, Employee IDs, Salary Information, Private Messages or Chats, Security Questions and Answers, Location Data, Photographs, Educational Records, Cookies, Browser History, User Behavior Data, Web Server Logs, API Keys, Encryption Keys, Cloud Storage Keys, OAuth Tokens, Cryptocurrency Wallets, Session IDs, Device IDs, Personal Identification Numbers (PINs), Legal Documents, Employment History, Marital Status, Family Information, Religious Affiliation, Sexual Orientation, Political Views, Racial or Ethnic Origin, Trade Union Membership), their origin (meaning, how it is retrieved - user input, database, service or storage - in the latter case you must include the exact name of the service or storage), and function in a CSV format. In the function try to include what happens with the attribute in the code, how it is stored, where it is forwarded Be conceise! Avoid redundancy, for example if an object containts sensitive attributes, you don't need to include them individually, just include the object and in its function mention that it contains email address, password, etc. The code is as follows, separated by three backticks:
"'{text}"'
In your answer, use this CSV format:
Element;Type;Level;Origin;ServiceName;Function;Goal;GoalName;Classname
Where the Element is the name of the sensitive element, the Type is its type based on the list above i.e. Username,Password,Email Address, etc. the reason why this element is identified as sensitive. The Level is a numberic value from 1 - 3, signifying how sensitive this information is. 1 means Low Severity, such as browser history, user behaviour, cookies or the identifiers of such information, 2 means Medium Severity such as username, email, password, birthdate, ip address, location data, political/religious views, sexual orientation, etc., 3 means High Severity, information that might lead to identity theft, financial loss, or significant privacy invasion such as social security number, health insurance, drivers license, legal documents, passport, medical information, credit card data, encryption keys, tax ids. The Origin is one of the following values: ('UserInput', 'LocalStorage', 'Service') The ServiceName is the name of the Service from where the element came if the value of the Origin is 'Service', else

its empty string. The Function is a description of what the given element is used for The Goal is where the element is forwarded, stored, what happens to it, it must be one of the following values: ('Read', 'Stored', 'Service') The GoalName is the name of the Service the element is forwarded to if the value of the Goal field was 'Service'. The Classname is the name of the Angular class where the element was detected. For example:
"'
customersArrayData; CustomerInformation;2; Service; DbServiceService;Array of Customer objects containing sensitive information about the customers; Read; ; CustomersComponent
id;User Identifier; 1; UserInput; Id to query a Customer object from the database;Used to call the GetCustomer function of the DbServiceService; Service, DbServiceService; CustomersComponent
"'
Your answer must include only the CSV format and nothing else! If there are no sensitive elements in the file, your answer must be exactly 'There are no sensitive elements in the file!' The element and variable names in the files might not be English! Even if parts of the source code is in Spanish, German, Italian, Hungarian or other languages, etc. you must correctly identify the sensitive elements!

B.2.2 The definition of the mapping JSON, to be used my multiple prompts

{ **"type"**: "type of the analyzed file, for example, Component, Service, Guard, etc.",
"selector": "if the type of the analyzed file is a Componet, this field should contain its selector string from its class decorator.
If the file is not a Component, the value should be an empty string", **"name"**: "name of the class in the analyzed file, for example AppComponent, LoginService, etc.",
"file": "the name of the analyzed file. If you received multiple files for a component - the .ts and the .html, this should be the name of the .ts file",
"path": "the path of the component - if there is any - from the navigation modules. In this step, the value of this should remain an empty list",
"guarded": "if the class is a component, and it is protected by an AuthGuard, the value should be 1, the default value is 0",
"guardLevel": "The value of this should be 0",
"guards": "The names of the guard classes that protect the class in the analyzed file. The default value is an empty list []. If the analyzed class is not an Angular Component, it must remain an empty list.",
"navigations": "A list of possible navigations from the file by class names. If you cannot find out the class name of the component

targeted by the navigation, you should include it's path. An optimal value of this field would be: ['LoginComponent', 'DashboardComponent']
an acceptable value would be ['/login', '/dashboard']. If there are no possible navigations from the analyzed class, the list should be empty.
You also need to take into account navigations that can possibly happen via a service, such as this.superService.navigate('home') or this.tereloService.menj('/profile')
If the analyzed file was an Angular Module, the value should be an empty list ([])!",
"injections": "A dict, where the keys are the names of the services and other injectables that were injected into the analyzed class via the constructor. The values of the keys is a dict, containing and organizing the calls when said injectable was used in the file. The keys of this inner dict are Asynchrons, Functions and Variables. Asynchron refers to Promises and Observables used by the analyzed file, Functions to the names of functions called by the analyzed file, Variables to the various variables of the injected accessed directly by the analyzed file. The values of this inner dict are lists which are either empty, if that type of interaction with the injectable did not happen, or the names of said interaction. An example value for this field would be: {'LoginService': 'Functions': ['login', 'forgotPassword'], 'Variables': [], 'Asynchrons: ['currentUser']', 'LoggerService: 'Functions': ['logInfo', 'logError'], 'Variables': ['log'], 'Asynchrons': []'} If there are no injectables in the file, the value should be an empty dict!",
"parents": "If the analyzed file is a Component and it has parent Components, this should be a list that contains the names of those components, like ['ProfileComponent'], or ['LoginComponent', 'RegisterComponent']. The default value is an empty list.",
"children": "If the type of the analyzed file is a Component, this list should contain the names - or if the names cannot be determined from the context, then the selector strings of every child Component that are contained by the HTML template of the Component. For example: ['SpinnerComponent', 'alert-component']. If the type of the file is not Component or it doesn't have any children, then the value of this field should be an empty list. The router-outlet should also be considered to be a child, so if the .html template contains it, it should be included here!"
}

B.2.3 The mapping prompt that converts the minified TypeScript files into JSONs

You will receive a minified version of a TypeScript file from an Angular web application, separated by three backticks, describing either a Component or a Service of an Angular project. Your goal is to create JSON from said file, which contains the services that were injected into it via the constructor, the names of the functions called

from said services, the possible navigation operations, and in case of components, the names of the children components in a list. If any of the mentioned elements are missing, its place in the JSON should contain the key, but only an empty list as its value. The JSON should follow this format:
{firstResultFormat}
A short example of a correct output:
{firstResultFormatExample}
The file to be analyzed is the following:
“‘
{section}
“‘
Your answer should only contain the JSON and nothing else, as your answer will be instantly parsed as a JSON by the receiver! Use double quotation marks (”) and no single quotation mark (’) under any circumstance!

B.2.4 The access control evalution prompt

You will receive a JSON object, which is describing a Component from an Angular project and their relations to each other. The JSON object will follow this format: {firstResultFormat} With this JSON you will also receive minified versions of the Angular files, that will contain information concerning the routing and the authguard configurations of the web application. Based on these files you will need to extend the JSON representing the Component with the following steps.
1. Modify the path value if their ’path’ is an empty list. If there is no path defined in any of the Route[] arrays in the received files that contain a path value for that particular Component, then the value of its path should remain an empty list in the JSON. There is a good chance for example, that Components that are children to another Component should not have any paths, because they are most likely to be accessible through their parent Component. For example: If there is {{ path: ’register’, component: RegisterComponent }} in one of the routing modules Then in the JSON, where type: ’Component’ and name: ’RegisterComponent’, ’path’ should be [’/register’] Make sure that even if there’s already a value in the path field, it has a unified format, such as ’/register’, ’/home’, ’/profile/:id’ The reason why the path is a list and not a string is because you also need to include the default/home path to the appropriate JSON (’/’) and the wildcard path (’**’) to the appropriate JSON if they exist in the route definitions. The paths might be also stored in multiple files if there is a module-submodule relationship. For example, if one routing module contains the following:
{{path: ’users/:id’,loadChildren: () = import(”./users-module/users.module”).then(m = m.UsersModule)}}, then the UsersModule imports another routing module, for

example a UsersRoutingModule, where there is a line such as:
{{path: 'addition', component: AddUserComponent}},
then the path for the AddUserComponent is '/users/:id/adddition' if there is
{{path: ", redirectTo: 'profile'}}
in that module, then the component with the '/users/:id/profile' path should also get the '/users/:id/' route in their paths array. Do not under any circumstance mix the paths and the possible navigations from that given Component! - only Components have paths! - most Component only have one path. If the Component is used as the child Component of another, it is very likely that it doesn't have a path in the routing information, therefore its path list should remain empty. - the only exceptions are usually the default routes: ", '/' and the wildcard route: '**'
2. List every 'Guard' by their class names in the 'guards' list of the JSON that are protecting the Component based on the canActivate parts of the routing information. For example:
if there is a configuration like this in one of the module files {{path: 'profile', component: ProfileComponent, canActivate: [AuthguardGuard]}} then in the JSON, where 'type': 'Component' and 'name': 'ProfileComponent', 'guarded' should be set to 1, and the 'guards' should be: ['AuthguardGuard'] Again, take into account the module level imports: if there is a {{path: 'users/:id', canActivate: [RoleGuard], loadChildren: () = import("./users-module/users.module").then(m = m.UsersModule)}} then every route/component that is imported by the UsersModule already has the RoleGuard, in addition to any other they might have individually! for example a {{path: 'addition', component: AddUserComponent}} in the UsersModule's imported routes would give the AddUserComponent's JSON the ['RoleGuard'] as its guard, while a {{path: 'deletion', component: DeletionUserComponent, canActivate: [AdminGuard]}} would give the DeletionUserComponent's JSON the ['RoleGuard', 'AdminGuard'] list as its 'guards'.
3. As a final step, you need to calculate the guardLevel if the JSON has 'Component' as its type, based on the following rules:
- if the 'guarded' value is 0 and their 'guards' list is still empty after the previous steps, the 'guardLevel' is 0 - if it has at least one Guard in their 'guards' list, but based on the implementation of it, that Guard only checks whether the user is logged in in or not, the 'guardLevel' should be 1. - if the implementation of Guard or Guards in the 'guards' list not only check whether they are logged in, but they also check for access level - admin or normal user; patient, relative, nurse or practitioner; buyer or seller - the 'guardLevel' should be 2. - if the implementation of the Guards contain everything necessary for guardLevel 2, but go beyond them and check for extra information, for example, location, IP address, timezone, device/browser type, the guardLevel should be 3.
Your answer should contain the full version of the modified JSON, it will have to

include every object which was in the input in the same format and contain every modification you have made to it.
This is the input JSON file, separated by three backticks:
“‘
{firstResult}
“‘
And these are the minified files containing the routing and guard information, also separated by three backticks:
“‘
{filteredGuards}
“‘

Bibliography

[1] About Modeler — Camunda Platform 8 Docs — docs.camunda.io. https://docs.camunda.io/docs/components/modeler/about-modeler/. [Accessed 15-Sep-2022].

[2] Business Process Model And Notation Specification Version 2.0 — omg.org. https://www.omg.org/spec/BPMN/2.0/. [Accessed 10-Sep-2022].

[3] Index - fhir v4.0.1. https://www.hl7.org/fhir/. (Accessed on 09/16/2020).

[4] Introduction to the angular docs. https://angular.io/docs. Last accessed on 2023-07-20.

[5] Mongodb official documentation. https://docs.mongodb.com/manual/. (Accessed on 09/16/2020).

[6] Open policy agent - policy performance. https://www.openpolicyagent.org/docs/latest/policy-performance/. (Accessed on 09/16/2020).

[7] Open policy agent official site. https://www.openpolicyagent.org/. (Accessed on 09/16/2020).

[8] Patient flow analysis : an overview of national and international application — apps.who.int. https://apps.who.int/iris/handle/10665/59190?locale-attribute=es&. [Accessed 29-Sep-2022].

[9] Simcad Simulation Software - Patient Flow Simulation: Predictive Modeling and Analytics. https://www.createasoft.com/patient-flow-simulation. [Accessed 24-Jan-2023].

[10] Simul8 - Improve patient flow and enhance service quality. https://www.simul8.com/applications/healthcare/improving-emergency-department-processes-with-simulation. [Accessed 24-Jan-2023].

[11] Visual Paradigm - How to Create BPMN Diagram? https://www.visual-paradigm.com/tutorials/how-to-create-bpmn-diagram/. [Accessed 24-Jan-2023].

[12] What Is Patient Flow? — catalyst.nejm.org. https://catalyst.nejm.org/doi/full/10.1056/CAT.18.0289. [Accessed 29-Sep-2022].

[13] What is the difference between the gpt-4 models? https://help.openai.com/en/articles/7127966-what-is-the-difference-between-the-gpt-4-models. Last accessed on 2023-07-20.

[14] SpiffWorkflow 1.1.6 documentation — spiffworkflow.readthedocs.io. https://spiffworkflow.readthedocs.io/en/latest/, 2014. [Accessed 22-Sep-2022].

[15] Webrtc 1.0: Real-time communication between browsers, 2018.

[16] Seham Alnefaie, Asma Cherif, and Suhair Alshehri. Towards a distributed access control model for iot in healthcare. In *2019 2nd International Conference on Computer Applications & Information Security (ICCAIS)*, pages 1–6. IEEE, 2019.

[17] Ahmad Altamimi. Secfhir: A security specification model for fast healthcare interoperability resources. *International Journal of Advanced Computer Science and Applications (ijacsa)*, 7(6), 2016.

[18] Emilien Arnaud, Mahmoud Elbattah, Christine Ammirati, Gilles Dequen, and Daniel Aiham Ghazali. Use of artificial intelligence to manage patient flow in emergency department during the COVID-19 pandemic: A prospective, single-center study. *International Journal of Environmental Research and Public Health*, 19(15):9667, 2022.

[19] Hany F Atlam, Robert J Walters, and Gary B Wills. Fog computing and the internet of things: A review. *big data and cognitive computing*, 2(2):10, 2018.

[20] Hasiba Ben Attia, Laid Kahloul, and Saber Benharzallah. A new hybrid access control model for security policies in multimodal applications environments. *Journal of Universal Computer Science*, 24(4):392–416, 2018.

[21] Nureni Ayofe Azeez and Charles Van der Vyver. Security and privacy issues in e-health cloud-based system: A comprehensive content analysis. *Egyptian Informatics Journal*, 20(2):97–108, 2019.

[22] Rahime Belen Saglam, Jason R.C. Nurse, and Duncan Hodges. Personal information: Perceptions, types and evolution. *Journal of Information Security and Applications*, 66:103163, May 2022.

[23] Á Berta, V. Bilicki, and M. Jelasity. Defining and understanding smartphone churn over the internet: a measurement study. In *14-th IEEE International Conference on Peer-to-Peer Computing*. IEEE, 2014.

[24] Ali Borji and Mehrdad Mohammadian. Battle of the wordsmiths: Comparing chatgpt, gpt-4, claude, and bard. *SSRN Electronic Journal*, 2023.

[25] Stefan Bosnic, Ištvan Papp, and Sebastian Novak. The development of hybrid mobile applications with apache cordova. In *2016 24th Telecommunications Forum (TELFOR)*, pages 1–4. IEEE, 2016.

[26] Víctor Botti-Cebriá, Elena del Val, and Ana García-Fornes. Automatic detection of sensitive information in educative social networks. *13th International Conference on Computational Intelligence in Security for Information Systems (CISIS 2020)*, page 184–194, Aug 2020.

[27] Gracie Carter, Ben Chevellereau, Hossain Shahriar, and Sweta Sneha. Openpharma blockchain on fhir: An interoperable solution for read-only health records exchange through blockchain and biometrics. *Blockchain in Healthcare Today*, 2020.

[28] Basil Harris Chaballout, Ryan Jeffry Shaw, and Karin Reuter-Rice. The smart healthcare solution. *Advances in Precision Medicine*, 2(1), 2017.

[29] Songqing Chen, Tao Zhang, and Weisong Shi. Fog computing. *IEEE Internet Computing*, 21(2):4–6, 2017.

[30] Shiyuan Cheng, Jie Zhang, and Yuji Dong. How to understand data sensitivity? a systematic review by comparing four domains. *2022 4th International Conference on Big Data Engineering*, May 2022.

[31] X. Cheng, L. Fang, X. Hong, and L. Yang. Exploiting mobile big data: Sources, features, and applications. *IEEE Network*, 31(1):72–79, 2017.

[32] Anton Cheshkov, Pavel Zadorozhny, and Rodion Levichev. Evaluation of chatgpt model for vulnerability detection, 2023.

[33] Hui Na Chua, Jie Sheng Ooi, and Anthony Herbland. The effects of different personal data categories on information privacy concern and disclosure. *Computers Security*, 110:102453, Nov 2021.

[34] Ed Conley and Matthias Pocs. Gdpr compliance challenges for interoperable health information exchanges (hies) and trustworthy research environments (tres). *European Journal of Biomedical Informatics*, 14(3), 2018.

[35] Razvan Craciunescu, Albena Mihovska, Mihail Mihaylov, Sofoklis Kyriazakos, Ramjee Prasad, and Simona Halunga. Implementation of fog computing for reliable e-health applications. In *2015 49th Asilomar conference on signals, systems and computers*, pages 459–463. IEEE, 2015.

[36] Jianfeng Cui, Lixin Wang, Xin Zhao, and Hongyi Zhang. Towards predictive analysis of android vulnerability using statistical codes and machine learning for iot applications. *Computer Communications*, 155:125–131, Apr 2020.

[37] Christine SM Currie, John W Fowler, Kathy Kotiadis, Thomas Monks, Bhakti Stephan Onggo, Duncan A Robertson, and Antuela A Tako. How simulation modelling can help reduce the impact of COVID-19. *Journal of Simulation*, 14(2):83–97, 2020.

[38] Jianyang Deng and Yijia Lin. The benefits and challenges of chatgpt: An overview. *Frontiers in Computing and Intelligent Systems*, 2(2):81–83, Jan 2023.

[39] Peter Dieckmann, Kjetil Torgeirsen, Sigrun Anna Qvindesland, Libby Thomas, Verity Bushell, and Hege Langli Ersdal. The use of simulation to prepare and improve responses to infectious disease outbreaks like COVID-19: practical tips and resources from Norway, Denmark, and the UK. *Advances in Simulation*, 5(1):1–10, 2020.

[40] Michael M Dinh and Saartje Berendsen Russell. Overcrowding kills: How COVID-19 could reshape emergency department patient flow in the new normal. *Emergency Medicine Australasia*, 33(1):175–177, 2021.

[41] Rasheed El-Bouri, Thomas Taylor, Alexey Youssef, Tingting Zhu, and David A Clifton. Machine learning in patient flow: a review. *Progress in Biomedical Engineering*, 3(2):022002, 2021.

[42] Sidong Feng and Chunyang Chen. Prompting is all you need: Automated android bug replay with large language models, 2023.

[43] José Luis Fernández-Alemán, Inmaculada Carrión Señor, Pedro Ángel Oliver Lozoya, and Ambrosio Toval. Security and privacy in electronic health records: A systematic literature review. *Journal of biomedical informatics*, 46(3):541–562, 2013.

[44] David Ferraiolo, Ramaswamy Chandramouli, Rick Kuhn, and Vincent Hu. Extensible access control markup language (xacml) and next generation access control (ngac). In *Proceedings of the 2016 ACM International Workshop on Attribute Based Access Control*, pages 13–24, 2016.

[45] David Ferraiolo, Janet Cugini, and D Richard Kuhn. Role-based access control (rbac): Features and motivations. In *Proceedings of 11th annual computer security application conference*, pages 241–48, 1995.

[46] David Ferraiolo, Janet Cugini, D Richard Kuhn, et al. Role-based access control (rbac): Features and motivations. In *Proceedings of 11th annual computer security application conference*, pages 241–48, 1995.

[47] Pedro Garcia Lopez, Alberto Montresor, Dick Epema, Anwitaman Datta, Teruo Higashino, Adriana Iamnitchi, Marinho Barcellos, Pascal Felber, and Etienne Riviere. Edge-centric computing: Vision and challenges. *SIGCOMM Comput. Commun. Rev.*, 45(5):37–42, September 2015.

[48] F. Gechter, Alastair R. Beresford, and A. Rice. Reconstruction of battery level curves based on user data collected from a smartphone. In *Artificial Intelligence: Methodology, Systems, and Applications*, pages 289–298. Springer International Publishing, 2016.

[49] Massimiliano Giordano and Giuseppe Polese. Visual computer-managed security: A framework for developing access control in enterprise applications. *IEEE Software*, 30(5):62–69, 2013.

[50] Lu He, Sreenath Chalil Madathil, Amrita Oberoi, Greg Servis, and Mohammad T Khasawneh. A systematic review of research design and modeling techniques in inpatient bed management. *Computers & Industrial Engineering*, 127:451–466, 2019.

[51] A. Heydon, M.W. Maimone, J.D. Tygar, J.M. Wing, and A.M. Zaremski. Miro: visual specification of security. *IEEE Transactions on Software Engineering*, 16(10):1185–1197, 1990.

[52] Jennifer Hitzek, Antje Fischer-Rosinskỳ, Martin Möckel, Stella Linnea Kuhlmann, and Anna Slagman. Influence of weekday and seasonal trends on urgency and in-hospital mortality of emergency department patients. *Frontiers in Public Health*, 10, 2022.

[53] Md Rakib Hossain Misu and Kazi Sakib. Fantasia: A tool for automatically identifying inconsistency in angularjs mvc applications. 10 2017.

[54] Hussam Hourani, Ahmad Hammad, and Mohammad Lafi. The impact of artificial intelligence on software testing. *2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT)*, Apr 2019.

[55] SM Riazul Islam, Daehan Kwak, MD Humaun Kabir, Mahmud Hossain, and Kyung-Sup Kwak. The internet of things for health care: a comprehensive survey. *IEEE access*, 3:678–708, 2015.

[56] Ghasem Janbabai, Sajjad Razavi, and Ali Dabbagh. How to manage perioperative patient flow during COVID-19 pandemic: a narrative review. *Journal of Cellular & Molecular Anesthesia*, 5(1):47–56, 2020.

[57] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. Survey of hallucination in natural language generation. *ACM Computing Surveys*, 55(12):1–38, Mar 2023.

[58] Lin Jiang, Hui Liu, and He Jiang. Machine learning based recommendation of method names: How far are we. *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Nov 2019.

[59] Nan Jiang, Thibaud Lutellier, and Lin Tan. CURE: Code-aware neural machine translation for automatic program repair. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, may 2021.

[60] Jose L. Jimenez and Zhe Peng. Covid-19 airborne transmission tool available. `https://cires.colorado.edu/news/covid-19-airborne-transmission-tool-available`, Nov 2020.

[61] Alistair EW Johnson, Tom J Pollard, Lu Shen, H Lehman Li-Wei, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G Mark. Mimic-iii, a freely accessible critical care database. *Scientific data*, 3(1):1–9, 2016.

[62] Maithilee Joshi, Karuna Joshi, and Tim Finin. Attribute based encryption for secure access to cloud based ehr systems. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, pages 932–935. IEEE, 2018.

[63] Seung Woo Kang and Hyun Soo Park. Emergency department visit volume variability. *Clinical and experimental emergency medicine*, 2(3):150, 2015.

[64] Sasha Karakusevic. Understanding patient flow in hospitals. `https://www.abhi.org.uk/media/1215/understanding_patient_flow_in_hospitals-nuffield-trust.pdf`, 2016.

[65] Grácián Kokrehel and Vilmos Bilicki. The impact of the software architecture on the developer productivity. *Pollack Periodica*, 17(1):7–11, Mar 2022.

[66] Charles Lang, Charlotte Woo, and Jeanne Sinclair. Quantifying data sensitivity. *Proceedings of the Tenth International Conference on Learning Analytics Knowledge*, Mar 2020.

[67] J. K. Laurila, Daniel Gatica-Perez, I. Aad, J. Blom, Olivier Bornet, Trinh-Minh-Tri Do, O. Dousse, J. Eberle, and M. Miettinen. The mobile data challenge: Big data for mobile computing research. 2012.

[68] Seunghoon Lee and Young Hoon Lee. Improving emergency department efficiency by patient scheduling using deep reinforcement learning. In *Healthcare*, volume 8, page 77. MDPI, 2020.

[69] Di Leva and Emilio Sulis. A business process methodology to investigate organization management: A hospital case study. *WSEAS Transactions on Business and Economics*, 14:100–109, 2017.

[70] L. Li, B. Beitman, M. Zheng, X. Wang, and F. Qin. edelta: Pinpointing energy deviations in smartphone apps via comparative trace analysis. In *2017 Eighth International Green and Sustainable Computing Conference (IGSC)*, pages 1–8. IEEE, 2017.

[71] Yang-Kuei Lin and Yin-Yi Chou. A hybrid genetic algorithm for operating room scheduling. *Health care management science*, 23(2):249–263, 2020.

[72] Eugene Litvak and Michael C Long. Cost and quality under managed care: Irreconcilable differences. *Am J Manag Care*, 6(3):305–12, 2000.

[73] Yiheng Liu, Tianle Han, Siyuan Ma, Jiayue Zhang, Yuanyuan Yang, Jiaming Tian, Hao He, Antong Li, Mengshen He, Zhengliang Liu, Zihao Wu, Dajiang Zhu, Xiang Li, Ning Qiang, Dingang Shen, Tianming Liu, and Bao Ge. Summary of chatgpt/gpt-4 research and perspective towards the future of large language models, 2023.

[74] D. MacDonald and L. Bruce. NAT behavior discovery using session traversal utilities for NAT (STUN). No. RFC 5780., 2010.

[75] A. Mandalari, A. Lutu, A. Dhamdhere, M. Bagnulo, and kc claffy. Tracking the Big NAT across Europe and the U.S. Technical report, Center for Applied Internet Data Analysis (CAIDA), Apr 2017.

[76] Bob Martin, Mason Brown, Alan Paller, Dennis Kirby, and S Christey. Cwe. *SANS top*, 25, 2011.

[77] Robert C Martin. Getting a solid start. *Robert C. Martin-objectmentor. com*, 2013.

[78] Mohammad Y. Mhawish and Manjari Gupta. Predicting code smells and analysis of predictions: Using machine learning techniques and software metrics. *Journal of Computer Science and Technology*, 35(6):1428–1445, Nov 2020.

[79] Shima Rahimi Moghaddam and Christopher J. Honey. Boosting theory-of-mind performance in large language models via prompting, 2023.

[80] Pouyan Momeni, Yu Wang, and Reza Samavi. Machine learning model for smart contracts security analysis. *2019 17th International Conference on Privacy, Security and Trust (PST)*, Aug 2019.

[81] L. Moroney. *Firebase Cloud Messaging*, pages 163—188. Apress, Berkeley, CA, 2017.

[82] Subhojeet Mukherjee, Indrakshi Ray, Indrajit Ray, Hossein Shirazi, Toan Ong, and Michael G Kahn. Attribute based access control for healthcare resources. In *Proceedings of the 2nd ACM Workshop on Attribute-Based Access Control*, pages 29–40, 2017.

[83] Marwan Omar. Detecting software vulnerabilities using language models, 2023.

[84] Hideki Otsuki, Yoshitaka Murakami, Kazunori Fujino, Kazuhiro Matsumura, and Yutaka Eguchi. Analysis of seasonal differences in emergency department attendance in Shiga Prefecture, Japan between 2007 and 2010. *Acute Medicine & Surgery*, 3(2):74–80, 2016.

[85] Shantanu Pal, Michael Hitchens, Vijay Varadharajan, and Tahiry Rabehaja. Fine-grained access control for smart healthcare systems in the internet of things. *EAI Endorsed Transactions on Industrial Networks and Intelligent Systems*, 4(13), 2018.

[86] Seunghyun Park and Jin-Young Choi. Malware detection in self-driving vehicles using machine learning algorithms. *Journal of Advanced Transportation*, 2020:1–9, Jan 2020.

[87] Mor Peleg, Dizza Beimel, Dov Dori, and Yaron Denekamp. Situation-based access control: Privacy management via modeling of patient data access scenarios. *Journal of Biomedical Informatics*, 41(6):1028–1040, 2008.

[88] Uthpala Premarathne, Alsharif Abuadbba, Abdulatif Alabdulatif, Ibrahim Khalil, Zahir Tari, Albert Zomaya, and Rajkumar Buyya. Hybrid cryptographic access control for cloud-based ehr systems. *IEEE Cloud Computing*, 3(4):58–64, 2016.

[89] Harsha S Gardiyawasam Pussewalage and Vladimir A Oleshchuk. An attribute based access control scheme for secure sharing of electronic health records. In *2016 IEEE 18th International Conference on e-Health Networking, Applications and Services (Healthcom)*, pages 1–6. IEEE, 2016.

[90] Indrakshi Ray, Bithin Alangot, Shilpa Nair, and Krishnashree Achuthan. Using attribute-based access control for remote healthcare monitoring. In *2017 Fourth International Conference on Software Defined Systems (SDS)*, pages 137–142. IEEE, 2017.

[91] Indrakshi Ray, Toan C Ong, Indrajit Ray, and Michael G Kahn. Applying attribute based access control for privacy preserving health data disclosure. In *2016 IEEE-EMBS International Conference on Biomedical and Health Informatics (BHI)*, pages 1–4. IEEE, 2016.

[92] Fatemeh Rezaeibagha and Yi Mu. Distributed clinical data sharing via dynamic access-control policy transformation. *International journal of medical informatics*, 89:25–31, 2016.

[93] P. Richter, F. Wohlfart, N. Vallina-Rodriguez, M. Allman, R. Bush, A. Feldmann, and V Paxson. A multi-perspective analysis of carrier-grade nat deployment. In *Proceedings of the 2016 Internet Measurement Conference*. ACM, 2016.

[94] Marco Rosa, Cristiano Faria, Ana Madalena Barbosa, Hilma Caravau, Ana Filipa Rosa, and Nelson Pacheco Rocha. A fast healthcare interoperability resources (fhir) implementation integrating complex security mechanisms. *Procedia Computer Science*, 164:524–531, 2019.

[95] Alex Rousskov and Valery Soloviev. A performance study of the squid proxy on http/1.0. *World Wide Web*, 2(1-2):47–67, 1999.

[96] John M.M. Rumbold and Barbara K. Pierscionek. What are data? a categorization of the data sensitivity spectrum. *Big Data Research*, 12:49–59, Jul 2018.

[97] Haya Salah and Sharan Srinivas. Predict, then schedule: Prescriptive analytics approach for machine learning-enabled sequential clinical scheduling. *Computers & Industrial Engineering*, page 108270, 2022.

[98] Yaira K Rivera Sánchez, Steven A Demurjian, and Mohammed S Baihan. Achieving rbac on restful apis for mobile apps using fhir. In *2017 5th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, pages 139–144. IEEE, 2017.

[99] Katharine Sanderson. Gpt-4 is here: what scientists think. *Nature*, 615(7954):773–773, Mar 2023.

[100] Advait Sarkar, Andrew D. Gordon, Carina Negreanu, Christian Poelitz, Sruti Srinivasa Ragavan, and Ben Zorn. What is it like to program with artificial intelligence?, 2022.

[101] Paul J Schramm, Ambarish Vaidyanathan, Lakshmi Radhakrishnan, Abigail Gates, Kathleen Hartnett, and Patrick Breysse. Heat-related emergency department visits during the northwestern heat wave—United States, June 2021. *Morbidity and Mortality Weekly Report*, 70(29):1020, 2021.

[102] Tushar Sharma, Maria Kechagia, Stefanos Georgiou, Rohit Tiwari, Indira Vats, Hadi Moazen, and Federica Sarro. A survey on machine learning techniques for source code analysis, 2022.

[103] Jianfei Sun, Hu Xiong, Hao Zhang, and Li Peng. Mobile access and flexible search over encrypted cloud data in heterogeneous systems. *Information Sciences*, 507:1–15, 2020.

[104] Yuqiang Sun, Daoyuan Wu, Yue Xue, Han Liu, Haijun Wang, Zhengzi Xu, Xiaofei Xie, and Yang Liu. When gpt meets program analysis: Towards intelligent detection of smart contract logic vulnerabilities in gptscan, 2023.

[105] Nigar M. Shafiq Surameery and Mohammed Y. Shakor. Use chat gpt to solve programming bugs. *International Journal of Information technology and Computer Engineering*, (31):17–22, Jan 2023.

[106] Z. Szabó and V. Bilicki. Felhőben tárolt egészségügyi adatok védelme abac modellel. 2018.

[107] Zoltán Szabó and Vilmos Bilicki. Access control of ehr records in a heterogeneous cloud infrastructure. *Acta Cybernetica*, 25(2):485–516, Dec 2021.

[108] Qais Tasali, Chandan Chowdhury, and Eugene Y Vasserman. A flexible authorization architecture for systems of interoperable medical devices. In *Proceedings of the 22nd ACM on Symposium on Access Control Models and Technologies*, pages 9–20, 2017.

[109] Mahdieh Tavakoli, Reza Tavakkoli-Moghaddam, Reza Mesbahi, Mohssen Ghanavati-Nejad, and Amirreza Tajally. Simulation of the COVID-19 patient flow and investigation of the future patient arrival using a time-series prediction model: a real-case study. *Medical & Biological Engineering & Computing*, 60(4):969–990, 2022.

[110] Gaute Terning, Eric Christian Brun, and Idriss El-Thalji. Modeling patient flow in an emergency department under COVID-19 pandemic conditions: A hybrid modeling approach. In *Healthcare*, volume 10, page 840. Multidisciplinary Digital Publishing Institute, 2022.

[111] Chandra Thapa, Seung Ick Jang, Muhammad Ejaz Ahmed, Seyit Camtepe, Josef Pieprzyk, and Surya Nepal. Transformer-based language models for software vulnerability detection. In *Proceedings of the 38th Annual Computer Security Applications Conference*, pages 481–496, 2022.

[112] V. Thirion, K. Edeline, and B. Donnet. *Traffic Monitoring and Analysis*, chapter Tracking Middleboxes in the Mobile World with TraceboxAndroid, pages 79—91. Springer, 2015.

[113] Csaba Varga, Zsuzsanna Lelovics, Viktor Soós, and Tibor Oláh. Betegforgalmi trendek multidiszciplináris sürgősségi osztályon. *Orvosi Hetilap*, 158(21):811–822, 2017.

[114] Pedro H Vilela, Joel JPC Rodrigues, Rodrigo da R Righi, Sergei Kozlov, and Vinicius F Rodrigues. Looking at fog computing for e-health through the lens of deployment challenges and applications. *Sensors*, 20(9):2553, 2020.

[115] Daniel T. Wagner, A. Rice, and Alastair R. Beresford. Device analyzer: Understanding smartphone usage. In *Mobile and Ubiquitous Systems: Computing, Networking, and Services*, pages 195–208. Springer International Publishing, 2014.

[116] Ji Wang, Bokai Cao, Philip S. Yu, Lichao Sun, Weidong Bao, and Xiaomin Zhu. Deep learning towards mobile applications. In *IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pages 1385–1393, July 2018.

[117] Z. Wang, Z. Qian, Q. Xu, Z. Mao, and M. Zhang. An untold story of middleboxes in cellular networks. In *Proceedings of the ACM SIGCOMM 2011 conference*. ACM, 2011.

[118] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler,

Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. Emergent abilities of large language models, 2022.

[119] Youn Kyoung Won, Tae ho Hwang, Eui Jung Roh, and Eun Hee Chung. Seasonal patterns of asthma in children and adolescents presenting at emergency departments in Korea. *Allergy, asthma & immunology research*, 8(3):223–229, 2016.

[120] Jiajie Wu. Literature review on vulnerability detection using nlp technology, 2021.

[121] Milad Yousefi, Moslem Yousefi, and Flavio S Fogliatto. Simulation-based optimization methods applied in hospital emergency departments: A systematic review. *Simulation*, 96(10):791–806, 2020.

[122] E. Yuan and J. Tong. Attributed based access control (abac) for web services. *IEEE International Conference on Web Services (ICWS'05)*, 2005.

[123] Eric Yuan and Jin Tong. Attributed based access control (abac) for web services. In *IEEE International Conference on Web Services (ICWS'05)*. IEEE, 2005.

[124] Sani Yusuf. *Ionic framework by example*. Packt Publishing Ltd, 2016.

[125] R. Zullo, A. Pescapé, K. Edeline, and B. Donnet. Hic sunt NATs: Uncovering address translation with a smart traceroute. In *2017 Network Traffic Measurement and Analysis Conference (TMA)*. IEEE, 2017.

Summary in English

This dissertation explores the challenges and problems generated by the complex access control requirements of modern telemedicine applications and demonstrates different solutions that meet or facilitate the resolution of these challenges.

The scientific results are presented in two groups, which are detailed in the third and fourth chapters of the thesis.

The first group of theses focuses on the complex data path of heterogeneous elements in the infrastructure of e-health applications. In it, I have presented the requirements that an access control solution must meet in order to properly and efficiently enforce the defined rules without impeding the care process, and I have defined a taxonomy for the policy categories. I have defined two types of edge outside the cloud that are special cases for privilege management, the processing and caching edge, and examined smartphone-based peer-to-peer networks as an alternative, fully distributed edge type from a stability and efficiency perspective. I have presented a proposed framework that builds on the concept of a Policy Enforcement Point (PEP) to implement an entitlement management point that can be placed at multiple points in the data path, capable of partial analysis of the retrieved data and also partial modification or encryption of the data as necessary before granting access. I implemented the proposed framework using the Open Policy Agent (OPA) and evaluated its operation and efficiency in test environments simulating two specific edge types.

In the second group of theses, I investigated the challenges posed by static analysis of applications at the end of the data path. I defined two taxonomies: one for detecting and ranking sensitive data and the other for determining the level of protection of application components. I validated the usability of the categories first by classifying a 200-word collection of variant names and then on a test set of 292 components extracted from an open-source Angular project. The resulting sensitive data was used to investigate the effectiveness of the static source code in identifying the sensitive parts of the software. Similarly, I validated the taxonomy of vulnerability levels, also with the component set of 292 components, and then analyzed the effectiveness of detecting vulnerabilities defined as inadequate isolation of critical parts identified as vulnerabilities in CWE-653, which risk leaking sensitive data, by combining the results of the two tests. In addition to presenting the results, I have also

identified the flaws that cause the problems and, based on these, the improvements that can be achieved by tightening the detection principles.

Contributions of the thesis

Thesis Group I. In the following thesis group, I examined the requirements of the telemedicine data pathway for access management. I defined the conditions that must be met at each designated points of the telemedicine data path in order to reconcile security requirements and responsiveness. I defined a four-element taxonomy for the categories of policy enforcement to be implemented. I presented the edge as a critical case for access control, its two primary categories for access control, the processing edge and the storage edge, and our results on a special case of the edge, the analysis of the stability of smartphone peer-to-peer networks. Then, I defined two experimental environments, one for the processing edge and one for the storage edge, as well as my experimental results in said environments using a practical implementation of the access control framework in terms of latency. Lastly, I presented my patient flow simulator built with open source libraries, which can simulate the throughput of patients treated in a hospital ward and can be used to generate validation parameters for the developed access control framework in order to validate the system's delay tolerance.

- **Thesis I/1.** I explored the complex requirements of modern telemedicine applications in terms of access control. I defined a taxonomy to formalize the different types of access control policies and the TAPE requirements necessary to ensure that the implementation of the defined policies can guarantee a balance between data privacy compliance and responsiveness at any point in the telemedicine infrastructure.

- **Thesis I/2.** I proposed special categories of edge instances beyond the cloud that represent a special category of modern telemedicine infrastructure from the perspective of access management. I formally defined a categorization of these into storage and processing edges. I then discuss the increasingly prevalent smart device based peer-to-peer networks as an extreme case of edge solutions, and analyse their potential and stability to function as stand-alone edge networks.

- **Thesis I/3.** I presented the implementation of the proposed access control solution, then I will set up test environments to represent both edge types, and sample rules to validate the effectiveness of the implementation under increasing data volumes. During the measurements, I examined the resource requirements of the nodes performing the evaluation, as well as the latencies measured for the data retrieval processes. I present the measured latencies which confirmed, that for reasonable amounts of data at the various edge types my policy categories met the requirements established in the previous theses.

- **Thesis I/3.1.** I developed a simulation tool using open source libraries and tools that can simulate the distribution of patients and their waiting times in a hospital ward. The developed simulation tool can be used to validate the extent to which the increased waiting times due to the access control implementation from the previous theses might slow down the process based on the amount of handled data, and the impact this can have on the telemedicine data path, the patient flow churn and waiting times at crucial parts of the care process.

Related publications

[J1] **Szabó, Z.**, Bilicki, V. (2021). Evaluation of EHR Access Control in a Heterogenous Test Environment. *Acta Cybernetica*, 25, 485-516. SJR: Q3, **0.75 credits**

[J2] **Szabó, Z.** (2021). Evaluation of a policy enforcement solution in telemedicine with offline use cases. *Pollack Periodica*, 17(1), 12-17. SJR: Q3, **1 credits**

[J3] **Szabó, Z.**, Hompoth, E. A., Bilicki, V. (2023). Patient Flow Analysis with a Custom Simulation Engine. *Acta Cybernetica*, – accepted, under publication SJR: Q4, **0.60 credits**

[C5] **Szabó, Z.**, Bilicki, V., Berta, Á., & Jánki, Z. R. (2017). Smartphone-based data collection with stunner using crowdsourcing: lessons learnt while cleaning the data. *In the Proceedings of ICCGI17* **0,48 credits**

[C6] Jánki, Z. R., **Szabó, Z.**, Bilicki, V., Fidrich, M. (2017, November). Authorization solution for full stack FHIR HAPI access. In *2017 IEEE 30th Neumann Colloquium (NC)* (pp. 000121-000124). IEEE. **0,48 credits**

[C7] **Szabó, Z.**, Téglás, K., Berta, Á., Jelasity, M., & Bilicki, V. (2019). Stunner: A smart phone trace for developing decentralized edge systems. In *Distributed Applications and Interoperable Systems: 19th IFIP WG 6.1 International Conference, DAIS 2019*, Held as Part of the 14th International Federated Conference on Distributed Computing Techniques, DisCoTec 2019, Kongens Lyngby, Denmark, June 17–21, 2019, Proceedings 19 (pp. 108-115). Springer International Publishing. **0,50 credits**

[C8] Berta, Á., **Szabó, Z.**, & Jelasity, M. (2020, December). Modeling Peer-to-Peer Connections over a Smartphone Network. In *Proceedings of the 1st International Workshop on Distributed Infrastructure for Common Good* (pp. 43-48). **0,60 credits**

[F9] **Szabó, Z.**, Bilicki, V. (2018, June). A FHIR-based healthcare system backend with deep cloud side security. In *THE 11TH CONFERENCE OF PHD STUDENTS IN COMPUTER SCIENCE* (p. 184).

[F10] **Szabó, Z.**, Bilicki, V. (2020, June). EHR Data Protection with Filtering of Sensitive Information in Native Cloud Systems. In *THE 12TH CONFERENCE OF PHD STUDENTS IN COMPUTER SCIENCE* (p. 164).

[F11] **Szabó, Z.** Policy Enforcement in Telemedicine with the Deployment of Multiple Enforcement Points. In *The 16th Iványi Miklós International PhD & DLA Symposium,2020.*

[F12] **Szabó, Z.**, Hompoth, E. A., Bilicki, V. (2022, June). Evaluation of a Custom Patient Flow Modeling Framework for Hospital Simulation. In *THE 13TH CONFERENCE OF PHD STUDENTS IN COMPUTER SCIENCE* (p. 202)

Thesis Group II. In the second group of theses, I focused on a crucial component of the telemedicine infrastructure, namely the front-end applications at the end of the data path. I defined a taxonomy that defines and ranks sensitive data in applications based on the impact of data leaks, and another taxonomy to determine the protection level of applications, in such a way that the defined categories can be transferred to LLMs using prompt engineering techniques. Then, I presented my results in classifying the elements of a variable-name dictionary and then in detecting sensitive data from open-source front-end applications via the GPT-3.5 and GPT-4 APIs, with which I have validated the hypothesis that through the complex knowledge of the LLMs at the level of the current GPT models, machine learning in the classical sense can in some cases already be derived by passing the necessary knowledge in the form of well-written prompts to the LLMs. Lastly, I discussed my results in static code-based detection of potential application vulnerabilities by evaluating the application protection level classification and then combining it with the results of the sensitivity detection.

- **Thesis II/1.** I defined a taxonomy of sensitive data in front-end applications, which categorises them into three different categories based on the level of damage caused by their leakage and unauthorised access. I then presented another taxonomy, which divides the protection levels of the application components into categories such that they are representative of the protection required for the sensitivity categories.

- **Thesis II/2.** I developed a GPT-enhanced methodology that uses the categorization defined in the previous thesis to detect sensitive data in front-end applications and, based on this data, to tag the code elements that focus on operations on such data. I validated the categorisation using an artificial intelligence-based evaluation that classifies the elements of a 200-word collection of variable names into one of the defined categories. I followed this by analyzing a total of 292 components from open source applications to detect sensitive data and based on that, identify the services handling sensitive information in the application.

- **Thesis II/3.** I defined a GPT-supported static source code analysis pipeline that uses the protection level categorisation to identify the protection level of components in a frontend application and then uses the results from the protection level and sensitivity level evaluations to detect vulnerabilities where components are not protected or whose protection level is insufficient for the sensitivity of the data they handle, in line with the CWE-653 type software vulnerability.

Related publications

[J4] **Szabó, Z.**, Bilicki, V. (2023). A new Approach to Web Application Security: Utilizing GPT Language Models for Source Code Inspection. In *Future Internet* MDPI., – under publication SJR: Q1, **0.75 credits**

Table B.4: *Relation between the thesis points and the corresponding publications*

Publication	Thesis point									
	Credit	IF	SJR	I/1	I/2	I/3	I/3/1	II/1	II/2	II/3
[J1]	0.75		Q3	●	●	●				
[J2]	1		Q3		●	●				
[J3]	0.60		Q4				●			
[J4]	0.75	3.4	Q1					●	●	●
[C5]	0.48				●					
[C6]	0.48			●						
[C7]	0.50				●					
[C8]	0.60				●					
[F9]	-			●						
[F10]	-			●	●					
[F11]	-				●					
[F12]	-						●			

Magyar nyelvű összefoglaló

A disszertáció feltárja a modern telemedicina alkalmazások komplex jogosultságkezelési igényei által generált kihívásokat és problémákat, és különböző megoldásokat demonstrál, amelyek teljesítik vagy megkönnyítik az említett kihívások megoldását.

A tudományos eredményeket két csoportra osztva mutattam be, melyek az értekezés harmadik és negyedik fejezetében kerültek részletes bemutatásra.

Az első téziscsoport a telemedicina alkalmazások infrastuktúrájának komplex, heterogén elemekből álló adatútjára koncentrál. Bemutattam benne a követelményeket, amelyeket egy jogosultságkezelési megoldásnak teljesítenie kell annak érdekében, hogy megfelelően és hatékonyan, az ellátási folyamat akadályozása nélkül érvényesíthesse a definiált szabályokat. Felvázoltam a felhőn kívül edge két, jogosultságkezelés szempontjából speciális esetet jelentő típusát, a feldolgozó és tároló edge-t, illetve megvizsgáltam az okostelefon alapú peer-to-peer hálózatokat stabilitás és hatékonyság szempontjából egy alternatív, teljesen elosztott edge típusként. Bemutattam egy tervezett keretrendszert, mely a Policy Enforcement Point (PEP) koncepciójára építve implementál egy, az adatút számos pontján elhelyezhető jogosultságkezelési pontot, mely képes a lekért adatok részleges elemzésére és azok szükségszerű részleges módosítására vagy titkosítására is a hozzáférés biztosítása előtt. A tervezett keretrendszert implementáltam az Open Policy Agent (OPA) segítségével, működését, hatékonyságát pedig kiértékeltem az két speciális edge típust szimuláló tesztkörnyezetekben.

A második téziscsoportban az adatút végén elhelyezkedő alkalmazások statikus elemzése támasztotta kihívásokat vizsgáltam. Definiáltam két kategorizálást, az egyiket az érzékeny adatok detektálása és rangsorolása, a másikat az alkalmazás komponensek védettségi szintjének megállapítására. A kategóriák használhatóságát validáltam előbb egy 200 szavas változónév-gyűjtemény besorolásával, majd egy nyílt forráskódú Angular projetekből kinyert, összesen 292 komponenst számláló teszthalmazon. Az így azonosított érzékeny adatok alapján megvizsgáltam, mennyire eredményesen azonosíthatóak a statikus forráskód alapján a szoftverek érzékeny elemekkel foglalkozó részei. Hasonlóan validáltam a védettségi szintek taxonómiáját is, szintén a 292 elemű komponenshalmazzal, majd a két vizsgálat eredményének összesítésével elemeztem, mennyire hatékonyan detektálható a CWE-653-as sérülékenységként azonosított, kritikus részek nem megfelelő izolációjaként definiálható sebezhetőség, mely

érzékeny adatok kiszivárgását kockáztatja. Az eredmények bemutatása mellett feltártam a problémákat okozó hibákat is, majd azok alapján a detektálási elvek szigorításával elérhető javulást.

A tézis eredményei

I. téziscsoport Az alábbi téziscsoportban megvizsgáltam telemedicina adatút követelményeit jogosultságkezelés tekintetében. Meghatároztam azokat a feltételeket, amelyeket az adatút minden pontján teljesíteni kell a biztonsági előírások és a válaszidő összeegyeztetése érdekében. Definiáltam egy négyelemű taxonómiát a szükséges jogosultságkezelési kategóriák számára. Bemutattam az edge-t, mint a jogosultságkezelés kritikus eseteit, meghatároztam két különleges típusát, a feldolgozó edge-t és a tároló edge-t, valamint bemutattam eredményeinket egy speciális edge esetről, az okostelefon alapú peer-to-peer hálózatok stabilitásának elemzéséről. Ezután két kísérleti környezetet hoztam létre, egyet a feldolgozó edge és egyet a tároló edge modellezésére, valamint teszt szabályokat az említett környezetekben a jogosultságkezelési keretrendszer gyakorlati megvalósításának validációjához késleltetés szempontjából. Végül bemutattam a nyílt forráskódú betegfolyam szimulátoromat, amely szimulálja a kórházi osztályokon kezelt betegek áramlását, és felhasználható a kifejlesztett jogosultságkezelési keretrendszer validálási paramétereinek generálására.

- **I/1. tézis** Megvizsgáltam a modern telemedicina alkalmazások komplex követelményeit jogosultságkezelés tekintetében. Definiáltam egy taxonómiát a különböző típusú jogosultságkezelési szabályok és az általam meghatározott TAPE-követelmények formalizálására, amelyek szükségesek annak biztosításához, hogy az implementált jogosultságkezelési megoldás garantálja az infrastruktúra bármely pontján a megfelelő adatvédelem és sebesség közötti egyensúlyt.

- **I/2. tézis** Speciális kategóriákat definiáltam a felhőn kívüli edge típusainak megkülönböztetésére, amelyek jogosultságkezelés szempontjából a modern telemedicina infrastruktúra különleges kategóriáját képviselik. Formálisan definiáltam ezeknek a kategóriákat feldolgozó és tároló edge-ként. Ezután megvizsgáltam az egyre elterjedtebb okostelefonokon alapuló peer-to-peer hálózatokat, mint a szélsőséges megoldások eseteit, és elemeztem azok potenciálját és stabilitását, hogy önálló edge hálózatokként működhessenek az adatúton.

- **I/3. tézis** Bemutattam a javasolt jogosultságkezelési megoldásom gyakorlati implementációját, majd tesztkörnyezeteket állítottam fel az edge típusok számára a négy kategóriához tartozó tesztszabályokkal, és ezekben a környezetekben mértem a végrehajtás hatékonyságát növekvő adatmennyiség mellett. A mérések során megvizsgáltam az értékelést végző csomópontok erőforrásigényeit, valamint az adatútra mért késleltetéseket. Az altézisben bemutatom a mért késleltetéseket, amelyek megerősítették, hogy a különböző edge típusokban észszerű mennyiségű adat esetében az általam felvázolt megoldás megfelel az előző tézisekben meghatározott követelményeknek.

- **I/3.1. tézis** Kifejlesztettem egy nyílt forráskódú szimulációs eszközt, amely nyílt forráskódú könyvtárakat és eszközöket használ a betegek eloszlásának és várakozási idejüknek - a betegfolyamnak - a modellezésére a kórházi osztályokon. A kifejlesztett szimulációs eszköz felhasználható annak validálására, hogy az előző altézisben mért megnövekedett átfutási idő milyen mértékben lassíthatja a folyamatot a kezelt adatok mennyisége alapján, és ez milyen hatással lehet a telemedicina adatútra, a betegfolyamra és a várakozási időkre az ellátási folyamat kulcsfontosságú pontjain.

Kapcsolódó publikációk

[J1] **Szabó, Z.**, Bilicki, V. (2021). Evaluation of EHR Access Control in a Heterogenous Test Environment. *Acta Cybernetica,* 25, 485-516. SJR: Q3, **0.75 credits**

[J2] **Szabó, Z.** (2021). Evaluation of a policy enforcement solution in telemedicine with offline use cases. *Pollack Periodica*, 17(1), 12-17. SJR: Q3, **1 credits**

[J3] **Szabó, Z.**, Hompoth, E. A., Bilicki, V. (2023). Patient Flow Analysis with a Custom Simulation Engine. *Acta Cybernetica*, – accepted, under publication SJR: Q4, **0.60 credits**

[C5] **Szabó, Z.**, Bilicki, V., Berta, Á., & Jánki, Z. R. (2017). Smartphone-based data collection with stunner using crowdsourcing: lessons learnt while cleaning the data. *In the Proceedings of ICCGI17* **0,48 credits**

[C6] Jánki, Z. R., **Szabó, Z.**, Bilicki, V., Fidrich, M. (2017, November). Authorization solution for full stack FHIR HAPI access. In *2017 IEEE 30th Neumann Colloquium (NC)* (pp. 000121-000124). IEEE. **0,48 credits**

[C7] **Szabó, Z.**, Téglás, K., Berta, Á., Jelasity, M., & Bilicki, V. (2019). Stunner: A smart phone trace for developing decentralized edge systems. In *Distributed Applications and Interoperable Systems: 19th IFIP WG 6.1 International Conference, DAIS 2019,* Held as Part of the 14th International Federated Conference on Distributed Computing Techniques, DisCoTec 2019, Kongens Lyngby, Denmark, June 17–21, 2019, Proceedings 19 (pp. 108-115). Springer International Publishing. **0,50 credits**

[C8] Berta, Á., **Szabó, Z.**, & Jelasity, M. (2020, December). Modeling Peer-to-Peer Connections over a Smartphone Network. In *Proceedings of the 1st International Workshop on Distributed Infrastructure for Common Good* (pp. 43-48). **0,60 credits**

[F9] **Szabó, Z.**, Bilicki, V. (2018, June). A FHIR-based healthcare system backend with deep cloud side security. In *THE 11TH CONFERENCE OF PHD STUDENTS IN COMPUTER SCIENCE* (p. 184).

[F10] **Szabó, Z.**, Bilicki, V. (2020, June). EHR Data Protection with Filtering of Sensitive Information in Native Cloud Systems. In *THE 12TH CONFERENCE OF PHD STUDENTS IN COMPUTER SCIENCE* (p. 164).

[F11] **Szabó, Z.** Policy Enforcement in Telemedicine with the Deployment of Multiple Enforcement Points. In *The 16th Iványi Miklós International PhD & DLA Symposium,2020.*

[F12] **Szabó, Z.**, Hompoth, E. A., Bilicki, V. (2022, June). Evaluation of a Custom Patient Flow Modeling Framework for Hospital Simulation. In *THE 13TH CONFERENCE OF PHD STUDENTS IN COMPUTER SCIENCE* (p. 202)

II. téziscsoport A disszertáció második téziscsoportjában a telemedicina adatút egyik kulcsfontosságú elemét elemeztem, az adatút végén fellelhető felhasználói alkalmazásokat. Olyan taxonómiát definiáltam, amely az adatok szivárgásának hatása alapján határozza meg és rangsorolja az alkalmazásokban az érzékeny adatokat, illetve egy másikat, amely meghatározza a alkalmazások komponenseinek védelmi szintjét, oly módon, hogy a meghatározott kategóriák prompt engineering technikák segítségével átadhatóak legyenek LLM-eknek is. Ezután bemutattam eredményeimet előbb egy változónevekből álló szótár elemeinek osztályozásával, majd a nyílt forráskódú frontend alkalmazások érzékeny adatainak detektálásával a GPT-3.5 és GPT-4 API-k segítségével, amellyel megerősítettem azt a hipotézist, hogy az LLM-ek összetett ismerete révén a jelenlegi GPT modellek szintjén a klasszikus értelemben vett tanítás bizonyos esetekben már kiváltható a szükséges tudás prompt alapú átadásával is. Végül összeállítottam egy metodológiát, mellyel elemeztem a válogatott nyílt forráskódú alkalmazásokat az érzékeny adatok, majd a védettségi szintek detektálásával, a két eredmény kombinációjával pedig a potenciális adatszivárgási pontok azonosításával.

- **II/1. tézis** Definiáltam a frontend alkalmazásokban található érzékeny adatok taxonómiáját, amely három különböző kategóriába sorolja őket a kiszivárgásuk és illetéktelen hozzáférésük által okozott kár mértéke alapján. Ezt követően bemutattam egy további kategorizálást, amely az alkalmazáskomponensek védelmi szintjeit úgy osztja szintekre, hogy azok reprezentálják az érzékenységi kategóriák alapján szükséges védelmet.

- **II/2. tézis** Kidolgoztam egy GPT API alapú módszertant, amely az előző tézisben meghatározott taxonómiát használja a frontend alkalmazásokban található érzékeny adatok felismerésére, és az ilyen adatokon végzett műveletekre összpontosító kódelemek megjelölésére. A kategorizálást egy kiértékeléssel validáltam, amely egy 200 szavas változónév-gyűjtemény elemeit a meghatározott kategóriák egyikébe sorolja. Ezt követően lefuttattam ezt a módszert nyílt forráskódú alkalmazásokból származó, összesen 292 komponensre, hogy azonosítsam bennük az érzékeny adatokat, ezáltal pedig az azok kezelésével foglalkozó serviceket.
- **II/3. tézis** Felvázoltam egy GPT alapú statikus forráskód-elemző pipelinet, amely a védelmi szintek taxonomiáját használja a frontend-alkalmazás összetevőinek védelmi szintjének azonosítására, majd a védelmi szint és az érzékenységi szint kiértékelésének eredményeit felhasználja a sebezhetőségek, potenciális adatszivárgások felderítésére, ahol a komponensek nem védettek, vagy amelyek védelmi szintje nem elegendő az általuk kezelt adatok érzékenységéhez képest, összhangban a CWE-653 típusú szoftver sebezhetőséggel.

Kapcsolódó publikációk

[J4] **Szabó, Z.**, Bilicki, V. (2023). A new Approach to Web Application Security: Utilizing GPT Language Models for Source Code Inspection. In *Future Internet* MDPI., – under publication SJR: Q1, **0.75 credits**

Table B.5: *A tézispontok és a publikációk kapcsolatai*

Publication	Thesis point									
	Credit	IF	SJR	I/1	I/2	I/3	I/3/1	II/1	II/2	II/3
[J1]	0.75		Q3	●	●	●				
[J2]	1		Q3		●	●				
[J3]	0.60		Q4				●			
[J4]	0.75	3.4	Q1					●	●	●
[C5]	0.48				●					
[C6]	0.48			●						
[C7]	0.50				●					
[C8]	0.60				●					
[F9]	-			●						
[F10]	-			●	●					
[F11]	-				●					
[F12]	-						●			

Publications

Journal papers

[J1] **Szabó, Z.**, Bilicki, V. (2021). Evaluation of EHR Access Control in a Heterogenous Test Environment. *Acta Cybernetica*, 25, 485-516. SJR: Q3, **0.75 credits**

[J2] **Szabó, Z.** (2021). Evaluation of a policy enforcement solution in telemedicine with offline use cases. *Pollack Periodica*, 17(1), 12-17. SJR: Q3, **1 credits**

[J3] **Szabó, Z.**, Hompoth, E. A., Bilicki, V. (2023). Patient Flow Analysis with a Custom Simulation Engine. *Acta Cybernetica*, – accepted, under publication SJR: Q4, **0.60 credits**

[J4] **Szabó, Z.**, Bilicki, V. (2023). A new Approach to Web Application Security: Utilizing GPT Language Models for Source Code Inspection. In *Future Internet* MDPI., – under publication SJR: Q1, **0.75 credits**

Full papers in conference proceedings

[C5] **Szabó, Z.**, Bilicki, V., Berta, Á., & Jánki, Z. R. (2017). Smartphone-based data collection with stunner using crowdsourcing: lessons learnt while cleaning the data. *In the Proceedings of ICCGI17* **0,48 credits**

[C6] Jánki, Z. R., **Szabó, Z.**, Bilicki, V., Fidrich, M. (2017, November). Authorization solution for full stack FHIR HAPI access. In *2017 IEEE 30th Neumann Colloquium (NC)* (pp. 000121-000124). IEEE. **0,48 credits**

[C7] **Szabó, Z.**, Téglás, K., Berta, Á., Jelasity, M., & Bilicki, V. (2019). Stunner: A smart phone trace for developing decentralized edge systems. In *Distributed Applications and Interoperable Systems: 19th IFIP WG 6.1 International Conference, DAIS 2019*, Held as Part of the 14th International Federated Conference on Distributed Computing Techniques, DisCoTec 2019, Kongens Lyngby, Denmark, June 17–21, 2019, Proceedings 19 (pp. 108-115). Springer International Publishing. **0,50 credits**

[C8] Berta, Á., **Szabó, Z.**, & Jelasity, M. (2020, December). Modeling Peer-to-Peer Connections over a Smartphone Network. In *Proceedings of the 1st International Workshop on Distributed Infrastructure for Common Good* (pp. 43-48). **0,60 credits**

Further related publications

[F9] **Szabó, Z.**, Bilicki, V. (2018, June). A FHIR-based healthcare system backend with deep cloud side security. In *THE 11TH CONFERENCE OF PHD STUDENTS IN COMPUTER SCIENCE* (p. 184).

[F10] **Szabó, Z.**, Bilicki, V. (2020, June). EHR Data Protection with Filtering of Sensitive Information in Native Cloud Systems. In *THE 12TH CONFERENCE OF PHD STUDENTS IN COMPUTER SCIENCE* (p. 164).

[F11] **Szabó, Z.** Policy Enforcement in Telemedicine with the Deployment of Multiple Enforcement Points. In *The 16th Iványi Miklós International PhD & DLA Symposium,2020.*

[F12] **Szabó, Z.**, Hompoth, E. A., Bilicki, V. (2022, June). Evaluation of a Custom Patient Flow Modeling Framework for Hospital Simulation. In *THE 13TH CONFERENCE OF PHD STUDENTS IN COMPUTER SCIENCE* (p. 202)

Further publications

[13] Nagy, Á., Dombi, J., Fülep, M. P., Rudics, E., Hompoth, E. A., **Szabó, Z.**, ... & Szendi, I. (2023). The Actigraphy-Based Identification of Premorbid Latent Liability of Schizophrenia and Bipolar Disorder. MDPI, *Sensors*, 23(2), 958.

[14] Rudics, E., Nagy, Á., Dombi, J., Hompoth, E. A., **Szabó, Z.**, Horváth, R., ... & Szendi, I. (2023). Photoplethysmograph Based Biofeedback for Stress Reduction under Real-Life Conditions in Healthcare Frontline. MDPI, *Applied Sciences*, 13(2), 835.

Table B.6: *Relation between the thesis points and the corresponding publications*

Publication	Thesis point									
	Credit	IF	SJR	I/1	I/2	I/3	I/3/1	II/1	II/2	II/3
[J1]	0.75		Q3	●	●	●				
[J2]	1		Q3		●	●				
[J3]	0.60		Q4				●			
[J4]	0.75	3.4	Q1					●	●	●
[C5]	0.48				●					
[C6]	0.48			●						
[C7]	0.50				●					
[C8]	0.60				●					
[F9]	-			●						
[F10]	-			●	●					
[F11]	-				●					
[F12]	-						●			