A Software Security Assessment using OWASP's Application Security Verification Standard

Results and Experiences from Assessing the DHIS2 Open-Source Platform

Andrei Eismont

Thesis submitted for the degree of
Master in Informatics: Programming and Networks
60 credits

Department of Informatics

Faculty of mathematics and natural sciences

UNIVERSITY OF OSLO

Spring 2020

A Software Security Assessment using OWASP's Application Security Verification Standard

Results and Experiences from Assessing the DHIS2 Open-Source Platform

Andrei Eismont

A Software Security Assessment using OWASP's Application Security Verification Standard

http://www.duo.uio.no/

Printed: Reprosentralen, University of Oslo

Abstract

Security in a software product is a property organizations can not overlook in this day and age, due to the sensitive and personal information they need to protect. To ensure that software is secure and that "due care" has been exercised, a standard can be utilized to measure one's security posture against industry best practices. This study aims to examine the application of the Application Security Verification Standard (ASVS) on the District Health Information System 2 (DHIS2) platform. By conducting an audit of DHIS2, the thesis explores how DHIS2 compares against ASVS and what the benefits and challenges of assessing DHIS2 using this standard are.

Interviews were conducted with members of the DHIS2 team, wherein the participants would select applicable ASVS controls to DHIS2 and then score them. The auditing results showed that DHIS2 failed to achieve complete coverage of ASVS. In analyzing the results for themes and patterns and drawing from personal experiences from conducting the assessment, several benefits and challenges were discovered. First, the data showed that ASVS's selection of control was relevant to DHIS2. Second, the ASVS results can be leveraged to address the shortcomings of DHIS2, to find vulnerabilities, and to fix them through a secure Software Development Life Cycle and other methods. The value of ASVS, however, is dependent on the organization's maturity and willingness to embrace and invest in this process.

Acknowledgements

First of, I would like to give a big thanks to my supervisors Johan Ivar Sæbø and Nils Gruschka for their guidance and inspiration throughout the project. Moreover I would like to thank the HISP team for their involvement in this project. Thanks to Bob Joliffe for providing invaluable input on my research.

And finally, a special thanks to my family for their continuous support and encouragement.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The necessity of handling system security and information security in particular is obvious when you look at the security breaches that have occurred in the last decade alone. The web service provider Yahoo, for instance, suffered an attack in 2013-14 in which the names, email addresses, and dates of birthdays of approximately 3 billion users was compromised [25]. The danger to the health industry is even greater, where a disruption can lead to the endangerment of patient lives. In late 2019, a small town in the Czech Republic suffered a cyberattack in which the staff was unable to use x-rays, ultrasound or laboratory instruments, forcing operations to be postponed or relocated to nearby hospitals [71].

Individual steps can be taken to prevent a breach but, identifying critical measures without a comprehensive guidance document can be challenging. To get an overview of one's security posture and identify critical weaknesses, a security assessment is required. A security assessment is a measurement of the security apparatus (people, process, technology) of a system, facility or organization [3]. In other words, the assessment looks at the way security is designed and implemented and identifies weaknesses and vulnerabilities in the implemented security controls. There are many approaches to security assessments, like penetration testing, risk assessment, threat modeling and vulnerability assessment. The method used

here is an internal audit of a health information system, using the Application Security Verification Standard (ASVS) [47] standard of security- techniques, and best practices to evaluate the company's defenses and security architecture. In other words, this means going through the requirements found in ASVS, and deciding if they are applicable and if the proposed solutions are in place. By comparing one's personal security system with a given frame of reference, an organization can discover critical weaknesses and raise awareness [24].

This thesis looks at the application of ASVS on the District Health Information System 2 (DHIS2) software, a health management platform that stores information of 67 low and middle-income countries in which it is used today [4]. Here the focus is on data security, centered around the CIA principles: confidentiality, integrity and availability, all an integral part of maintaining a healthy and secure information system. By using a standard to assess DHIS2's security, we also examine the process, the benefits and the challenges of this process.

1.1 Background

This thesis project was among several initiated by the HISP group at the University of Oslo (UiO), as part of a larger effort to evaluate and strengthen the security of the DHIS2 platform. Penetration tests by external organizations using DHIS2 had been done before, and for the findings to be fixed requested, but no systematic security audit had been done previously.

This research involved collaboration with DHIS2 developers and other members of the HISP group, who provided their knowledge and expertise in things related to DHIS2.

1.2 Motivation

Security breaches can and will happen, from large corporations such as Yahoo to smaller hospitals and medical service providers. At the time of writing, the Coronavirus (COVID-19) has become a global pandemic [16], and the health-care sector is more important now than ever. Still, hacker groups are targeting hospital and medical services, with two recently reported cases: an attack at the Champaign Urbana Public Health District in Illinois, USA, taken offline on March 10. 2020 by ransomware attack[1]; and again in the Czech Republic, an attack at the biggest hospital in the country that shut down all computers and delayed coronavirus test results [52]. These recent events underscore the importance of a capable security infrastructure. DHIS2's reach is enormous, and it is operated in countries with poor resources and poor health systems [12]. Its databases contain both aggregated and patient data. Therefore, protecting the privacy, integrity, and availability of that data from external threats is critical, especially today in the wake of COVID-19.

Scarfone, Benigni, and Grance from the National Institute of Standards and Technology (NIST) state that, "while it is impossible to eliminate all threats, improvements in cybersecurity can help manage security risks by making it harder for attackers to succeed and by reducing the effect of attacks that do occur" [54, Chapter 1]. No such systematic audit has been done before for DHIS2. A security audit is, therefore, important to assess the organization's security posture against industry best practice, to identify vulnerabilities and measure the effectiveness of the organization's controls. The motivation for doing this research is to explore this process and the applicability of an ASVS-type standard on DHIS2. It is the wish of the author that the research can also be relevant for similar organizations and software. The audit results will give the reader insight into DHIS2's

[1]Ransomware is a type of malware attack that blocks access to the victim's system or files until a ransom is paid

security posture, and the following discussion will explore the benefits and challenges of the ASVS auditing process.

1.3 Scope

This thesis looks at the auditing process from selecting the security controls found in ASVS, to scoring them as either pass or fail. Though instigated by the author, the selection and scoring is done by the various HISP employees at UiO who contributed to this research.

1.4 Research question

The purpose of this thesis is to do a security audit and then explore the benefits and challenges in assessing the security of a software product using a standard. First, benefits includes taking a broad perspective on the value of a standard, as well as exploring what the audit results revealed about DHIS2 and how this knowledge can be utilized. Second, challenges involves reflecting on the limitations and issues discovered during the auditing process. More specifically, the purpose of the thesis is reflected in the following research questions:

> *How does DHIS2's security compare against ASVS?*
>
> *What are the benefits and challenges of using ASVS to assess DHIS2's security?*

The first research question will be answered by first doing the audit and giving an overview of the results, and the second will be answered through an analysis of both the results and drawing from personal experiences from doing the audit.

1.5 Chapter Overview

This thesis is organized into 7 chapters as follows:

Chapter 1: Introduction: presents the background of this thesis, motivation and proposes the research question.

Chapter 2: Research context: provides information on HISP and the DHIS2 platform.

Chapter 3: Relevant Literature: presents the relevant literature, including an overview of software development, technical concepts, security standards, and finally a look at security regulations and threats to DHIS2.

Chapter 4: Research Approach: presents the research approach and how the application of ASVS to DHIS2 was conducted.

Chapter 5: Results: presents the results from the ASVS auditing process.

Chapter 6: Discussion and Analysis: evaluates the results from the case and present the learning outcome from the ASVS exercise.

Chapter 7: Conclusion: summarizes the discussion and concludes the thesis.

Chapter 2

Research context

To follow the case study, the contextual background of this thesis is required. First will be given a brief introduction to health information systems and the Health Management Information System Project (HISP). Next, an overview of the District Health Information System 2 (DHIS2), with a particular focus on its architectural composition.

2.1 Health Management Information Systems and HISP

An information system (IS) is defined as "formal, sociotechnical, organizational systems designed to collect, process, store, and distribute information" [27, p. 28]. A health information system (HIS) is an information system specific to the health sector, "designed to manage different kinds of healthcare-related data and is one of the foundational blocks of every health system" [6, p. 73]. Over the last couple of decades, there has been an increased global effort to accommodate the inadequate health status in developing countries, as existing HIS's were either not well-functioning or non-existing [6]. To tackle the many health-related challenges developing countries faced, a comprehensive health management system (HMIS) needed to be implemented, such that the governments or

Non-Governmental Organizations (NGOs) would be able to "monitor, detect and respond appropriately to public health emergencies" [6, p. 73]. The Health Management Information System Project (HISP) is one of those efforts.

Originated in South Africa after the fall of apartheid in 1994, the project was backed and developed in collaboration with local health activists, university students, and information system developers [6]. Coordinated from the University of Oslo (UiO) in Norway, HISP today is a global network comprising of several independent HISP groups, all of who help design, implement and sustain health information systems in over 100 developing countries around the world, including Indian states and the majority of Africa. The HMIS at the core of this effort is the Distributed Health Information System (DHIS).

2.2 Distributed Health Information System

Braa and Sahay define the DHIS software as a "free and open-source software platform for the collection, management, analysis, and use of health data" [13, p. 1]. Following its inception in 1994, DHIS has gone through several iterations. The latest generation, DHIS2, was launched in 2005 and is currently on its 33rd release. Decades later, today the software is relied upon by Ministries of Health and NGO's all across the globe, covering an estimated 2.28 billion people [6]. While the software core is developed by a team at UiO, it is part of a larger HISP ecosystem, local groups who "support the implementation of DHIS2 in their countries and regions based on their domain knowledge, technical and implementation expertise and experience" [6, p. 75]. These on-site observations from local HISP groups and other action-based research projects ensure that DHIS2 evolves to meet the many use cases and challenges that people need to solve.

2.2.1 DHIS2 as a platform

A platform can be defined as a group of technologies - at the center of which is a core used as a foundation on which outside parties can build products or services [64]. The platform provides a set of core functionality and an interface through which applications can interoperate. DHIS2 can be perceived as a platform based on these criteria. The following is an explanation of DHIS2's three main components and their boundaries, as illustrated in figure 2.1, as well as the interface that links them all.

Generic core. The main component of the DHIS2 platform is the core, a web-based solution that can run on most browsers and phones. The core consists of two parts: an SQL database and the Java-written codebase. This is the building block on which other applications can build upon.

Bundled apps. The second component is the bundled apps that come with the core package, developed by the same core DHIS2 team at UiO.

Custom apps. The third component is the custom apps. The platform's modular architecture allows it to be extended not only by apps but add-ons, plugins et cetera. This allows local developers to create applications specific to their use cases.

The Web API. The API supplies the interface between the user and the application. An external or internal application fetches data from the database in the form of a resource, in order words, anything the client might want to interact with. This resource could be patient data, configuration information et cetera. The web API allows the implementors to create custom apps that communicate with the core and the underlying database. Although each core given to the local implementors is the same, custom apps can alter the design of an individual DHSI2 instance significantly.

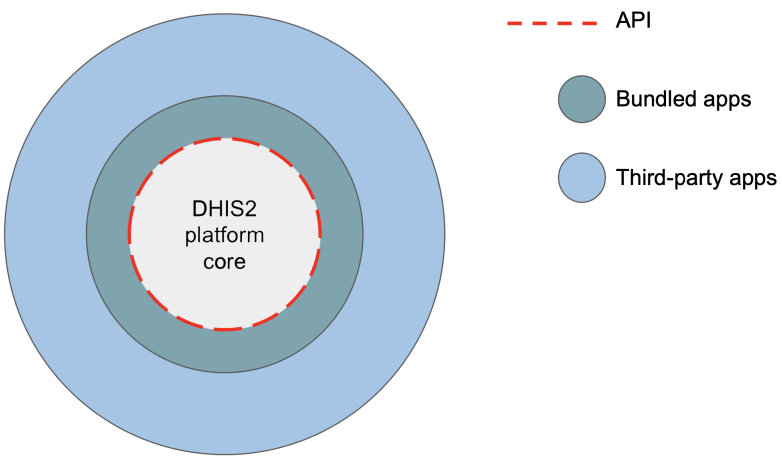


Figure 2.1: DHIS2 platform components (source: [39])

2.2.2 Configuring the platform and cloud hosting

When downloading DHIS2, it is empty of any content. The platform instance needs to be configured to a specific context and filled with data. The basic idea of DHIS2 is to provide a basic set of functionality to all countries and health facilities that implement it. By providing a flexible data structure, implementors can adapt the software to suit their specific problem or the laws or regulations that exist on the handling of private data in their country or region.

An implementors responsibility for deploying a reliable and secure DHIS2 instance includes the following (selected) actions [22]:

- Human resources with skills in relevant technologies such as web servers and database systems.
- Reliable backup of your system including safe storage at a remote server.
- Use of SSL (HTTPS/encryption) to keep private information like passwords secure.
- Monitoring of server resources and application performance.
- Secure server environment to avoid unauthorized access, theft and

fire.

When the DHIS2 project was created, it was important from the start that "the project invested significant resources in building expertise in HIS in the countries where HIS was implemented" [6, p. 74]. Today, this effort includes training in system implementation, maintenance, integration with other systems and software development of external apps. The implication of having a highly configurable, open-source platform is that aspects of DHIS2 can only be secured by the UiO development team, while other aspects of security need to be addressed at the implementation level.

2.2.3 Collected data and metadata

DHIS2 data can be described in two terms: collected data and metadata. Collected data is the data gathered from patients and districts, like data entered by a desk clerk or a doctor. There are three dimensions to this data: the what, where and when. *What* refers to the value measured by the indicator, e.g. the number of new measles outbreaks, pregnancies. *Where* records where a data value was collected or where a health event took place. *When* gives a period or a timestamp for when the data was collected.

Metadata refers to the data "describing your data" [1]. For instance, say you are tracking the mortality rate among infants aged less than five years old. This data would be collected in what DHIS2 calls a **dataset**. This dataset is configured by the implementors, who specify how often the data is to be collected, what indicators are to be used, which health facilities it applies to et cetera.

2.2.4 Aggregate and patient data

DHIS2 stores three types of data: user data, patient data, and aggregated data. Patient data, recorded by a health worker during a patient's visit, pertains to information about a single individual, such as their name, age, medical history and diagnosis. There are two functionalities (apps) for

registering individual entities: the Tracker and Event Capture modules. In the case of pregnancy, the Tracker app can be used to log when the baby is due, when the next scheduled checkup is set, et cetera. The Event Capture app is used to capture singular events. An event is not necessarily tied to a specific individual by a name or id, but information such as height, gender and the nature of the event are registered, as are any comments written by the health worker.

Aggregate is a collection of data, such as the number of pregnancies within a region or the cases of ebola outbreaks. This data is used as a statistic to generate reports to help health facilities and governments to track events and allocate their resources accordingly.

2.2.5 DHIS2 development

DHIS2 is developed in (agile) teams (more on the definition of Agile development in the next chapter). Four main teams work on different parts of the platform: Apps, Android, the Tracker module, and Analytics. Each team consists of a frontend and backend, both of which have team leads. Also, there are supporting services: UX and Quality Assurance.

The DHIS2 team keeps track of two separate Jira boards. A Jira board is used to report issues, follow requirements, progress, and roadmap for the DHIS2 software [23]. One is public, and the other is private to the core team.

2.2.6 DHIS2 security landscape

DHIS2 has experienced security breaches before. Machines have been taken over and data has been exposed (through an open database configuration). There are known architectural defects, the results from penetration tests, reports from the field et cetera. These can be addressed by the software team. However, the main challenge in regards to security is with implementation.

The implementors are not security experts. The different countries in which DHIS2 is implemented also frequently do not have a strong regulatory environment or institutions to promote good security practices. The consequence of this are many badly configured systems, with poor (or no) TLS configuration, naive (or no) proxy setup, no firewalls, no monitoring, inadequate physical environments, and poor management of users and roles. There are remedies to these problems, such as better guidance, better training materials, and better tools. And there has been significant progress in these areas, particularly in regards to training for server administrators. For the development team, the software release process has improved greatly also.

Chapter 3

Relevant literature

This chapter will present relevant literature that is deemed appropriate for the topics concerned in this thesis. First is an introduction to software development models followed by an overview of relevant security concepts. Second, the concept of a standard will be presented as well as relevant standards for the assessment of DHIS2. And finally, a brief explanation of security laws and regulations within the health sector and data protection in general, followed by an overview of the threat-landscape for an HMIS.

3.1 Software development

A development model describes the software development lifecycle of a development process [60]. It helps provide a common understanding of the software building process, such that developers can easier define what work has to be accomplished during each phase. One of the oldest of the development models is the *Waterfall model* [8]. Here, a development process consists of five phases: requirements, design, coding, testing, and maintenance. Each phase needs to be fully completed before continuing to the next, and this process lasts throughout the entire project. At the opposite end of that spectrum is the *Agile model*, currently used by the

DHIS2 development team. Here, the software is developed in iterations lasting between 2 and 3 weeks, called a sprint, during which each phase of the software development lifecycle (SDLC) is repeated for that sprint. Before each sprint begins, tasks are selected from a backlog to be developed during that iteration. In stark contrast to the Waterfall method, the agile method places a large emphasis on quick delivery and allows for a clearer set of priorities for developers to work on.

3.1.1 Secure Software Development Life Cycle

An SDLC defines the process used by organizations to build an application or a feature from start to finish [55]. Regardless of the development model, the prime reason for using an SDLC is that it provides a framework for a standard set of activities, such as planning, building and maintanence, and ensures that the project is planned, tracked and controlled throughout its life cycle [66]. In the past, it was common practice to perform security related-activities only as part of testing. However, this resulted in security-related bugs being discovered too late. To address this issue, the **secure software development life cycle** was invented, meant to integrate security-related activities in every step of the development process [41]. This ensures that activities such as penetration testing, code review, and architecture analysis are an integral part of the development effort [57]. Alternatively, these principles can be referred to as *Security by Design* and *Secure by Design*, two terms often used interchangeably, meaning that software has been designed from the foundation to be secure.

The integration of an security into your SDLC has many benefits, some of which are [43]:

- Making security a continuous concern.
- Raise awareness of security issues.
- Early detection of flaws in the system.

An example of an SSDLC is the Microsoft Security Development Lifecycle, launched in 2004 as a response to the growing PC market and equally growing threat landscape [5]. The SDLC was developed such that it would help integrate security-related practices found in the SDLC into the agile development environment, including training in secure coding techniques, secure code review and threat modeling.

3.1.2 Secure by Default

The United Kingdom's National Cyber Security Centre (NCSC) writes that "technology which is Secure by Default has the best security it can without you even knowing it's there, or having to turn it on" [56]. Though closely related to Security by Design, the *by Default* philosophy revolves around building software that is configured by default to be as secure as possible. This is beneficial not only to system administrators but, as Gorski and Iacano explain, also to non-security workers, who "benefit from safeguards being in place by default as this avoids vulnerabilities resulting from a lack of knowledge, comprehension or attention" [31, p. 1]. A selected few of Secure by Default principles as listed by the NCSC are [56]:

- security should never compromise usability – products need to be secure enough, then maximize usability
- security is never a goal in and of itself, it is a process – and it must continue throughout the lifetime of the product
- security should not require extensive configuration to work, and should just work reliably where implemented
- security should not require specific technical understanding or non-obvious behavior from the user.

Poor set-up or configuration of technology increases the risk of system vulnerabilities being exploited, either unintentionally by its users or purposefully by hackers.

3.2 Security Concepts

Security is a wide field of study with many concepts, principles, and services. This section will give a brief overview of a few concepts relevant to the discussion on security and standards.

3.2.1 Security services

How secure a system is strongly depends on the security requirements and the security measures that are in place. With respect to information systems and system security in general, the CIA (Confidentiality - Integrity - Availability) principles are important [53]. The security spectrum also includes accountability, authentication, non-repudiation, and other concepts. For this thesis, the CIA principles should provide the reader with adequate background on information system security.

Integrity

Integrity is to protect assets from unauthorized modifications or destruction. Integrity encompasses both data integrity (all digital information) and system integrity (state of the system) [30, p. 34-37]. In the context of DHIS2, tampering with health data is a breach of integrity.

Confidentiality

Confidentiality is to protect assets from being disclosed to unauthorized individuals [30]. A breach of confidentiality can both be accidental and intentional and can come from a bug or poor configuration that leads to disclosure of information.

Availability

Availability ensures the availability of data and resources to authorized individuals. It is crucial to all systems, in particular cloud services and information systems. An attack on availability is best illustrated by the DoS (Denial of Service) attack, which overloads a server's resources, thus hindering legitimate users from accessing their data.

3.2.2 Security controls

IT security is about protecting assets that are of value to the organization. This includes people, property and information. A standard will often include a series of controls for the purpose of reducing or mitigating the risk to those assets. Security controls are divided into these three categories [51, p. 12-13]:

- Physical
- Technical
- Administrative

The control category is selected based on the organization's objective. The aforementioned three categories are by type, but controls can also be categorized by function: preventive, detective, and corrective. For this thesis, only the former is required to understand ASVS. Note that in this thesis, the words *controls* and *requirements* are often used interchangeably. A security requirement can be defined as a "security feature required by system users or a quality the system must possess to increase the users trust in the system they use" [69]. The ASVS standard, for instance, names each category and subcategory as *requirements*. The items themselves, are typically referred to as controls.

The following is an overview of the three control types [51, p. 12-13].

Physical controls

Physical controls deal with the tangible, as in preventing or detecting unauthorized access to information through the use of locks, security guards, alarms, motions sensors, and similar measures.

Technical controls

Technical controls are security measures that use technology as a basis for authorizing access to sensitive data and protecting assets. This includes authentication solutions, firewalls, antivirus software, cryptography, intrusion detections, forensics, and other measures.

Administrative controls

Administrative controls (also called procedural in literature) are security measures that take into account the human factor. It refers to the policies, procedures, or guidance that define work-place practices when handling sensitive information, be it at rest or in use.

3.3 Security standards

A definition of a standard as defined by Webster is that it is "something set up and established by authority as a rule for the measure of quantity, weight, extent, value, or quality" [20]. Standards are applied everywhere: it is the celsius scale, date format and the rules by which the web communicates. In the context of health information, a standard is a common set of guidelines for the "the collection, reporting and use of health information by all developing countries and global agencies" [26]. In the case of application security, a standard provides a set of best-practice security techniques or requirements that, if followed, gives confidence that the application is secure. Further, the standard allows developers to

compare "their personal security system with a given frame of reference adopted at an international level" [65, p. 128]. In this chapter, the words standard and framework are sometimes used interchangeably.

This section will give an overview of a few standards relevant to DHIS2. In the security field, standards are typically categorized based on the objective. The following sections give info on three such categories: cyber-security standards; information security standards; and application security. Note that a standard can belong to multiple categories.

3.3.1 Cyber security standards

"Cyber Security, a subset of information security, is the practice of protecting systems, networks, and programs from digital attacks" [68]. Cyberattacks caused by cyber threats seek to access, change and destroy sensitive information. Part of the cybersecurity process is knowing what the critical data is, where it is stored, and what technology needs to be implemented to protect it. An example of a cybersecurity standard are the CIS controls, though all standards in this chapter contain cybersecurity safeguards.

3.3.2 Information Security Management Systems

An ISMS is a "systematic approach to managing sensitive company information so that it remains secure" [34], which includes people, processes, and IT systems. Like cybersecurity standards, an ISMS seeks to minimize risk and reduce the impact of a security breach. However, an ISMS focuses more on the management side of the company, meaning it recognizes that security breaches can occur both due to human errors and errors in the company's structure. For example, an employee can cause a security breach by accident, with no malicious intent, or purposely, in both cases causing significant damage. Examples of an ISMS is the ISO/IEC 27000-series and NIST's SP 800-53.

3.3.3 Application security

Application security is the measures taken to improve the security of an application during the application's lifecycle. This involves finding, fixing and preventing security vulnerabilities, through a wide range of techniques such as code review, white-box security review and black-box security audit. Whereas cybersecurity is directly aimed at defending against cyberattacks, application security involves the entire SDLC process. The Application Security Verification Standard is an example of an application security standard.

3.4 Overview of selected security standards

The project was presented with ASVS in mind. Chapter 5 will present an argument why it was the correct choice. But to understand ASVS's place within the standard landscape and present a valid argument for ASVS's selection, giving background on and comparing ASVS to other standards was important. Not all software products are alike, nor are all organizations and their security goals. Available manpower, cost, and regulatory compliance are all driving factors behind selecting a standard.

These standards presented here as candidates for DHIS2 were selected based on three criteria. First, the standard had to be general-purpose or health-specific. Unfortunately, standards tailored to the health-sector are few, and those that are relevant, such as HIPAA (Health Assurance Portability and Accountability Act) [62] - often referred to as a standard, are more concerned about information from a management point of view, rather than a technical one. The second criteria is that the standard had to be internationally-recognized and accepted by the professional community. And third, the standard had to be relatively modern.

Section 3.4.1 presents a few possible standards that could have been used to audit DHIS2. Section 3.4.2 then gives an in-depth description of ASVS,

written in more detail compared to the other standards as it was chosen for the task of auditing DHIS2.

3.4.1 Possible standard candidates

ISO/IEC 27000-series

The International Organization of Standardization (ISO) develops and publishes standards, one of which is the ISO/IEC 27000-family [34]. The pillar of the series is the ISO/IEC 27001 standard. An example of an ISMS, the standard provides guidelines on how to establish, implement, operate, monitor and maintain an information system. Moreover, the Annex A section of the publication contains over a hundred security controls, covering both the physical, the administrative, and the technical side of security. However, ISO/IEC 27001 only establishes what to do, but not how to do it. A sibling standard, the ISO/IEC 27002, provides concrete implementation guidelines on how to achieve a requirement from the Annex. While 27001 does not mandate a specific action, 27002 includes plenty of suggestions for corrective and preventive actions.

The Norwegian National Security Authority (NSM) advises corporations to follow a standard like the ISO/IEC 27001 as their security framework [45]. A bank, for instance, with several offices and possibly hundreds of employees handling high-value transactions, will require a management policy, internal organization structure, incident management rules and so on, such as provided by ISO/IEC 27001 – in addition to the technical and physical controls.

NIST SP 800-53

NIST SP 800-53, shorthand for National Institute of Standards and Technology Special Publication 800-53, also an ISMS, provides a catalog of technical, operational and administrative security controls [44]. For each

of its 1000+ security controls, SP 800-53 outlines what to examine, whom to interview, and what to test. The publication is relatively large, due to the fact that it was designed for governmental applications and information systems – in particular, to facilitate compliance with federal legislations and directives. A smaller, international software company may find the size of its content daunting, and its vast amount of controls irrelevant.

Center for Internet Security

An example of a Cybersecurity standard, the CIS controls (Center for Internet Security) consist of 20 actionable control categories, spread across 3 sections: **basic** (the essentials for cybersecurity defense), **fundamental** (technical best practice), and **organizational** (aimed at the people involved in maintaining a cybersecurity defense) [63]. At a glance, the framework lacks the depth and specialization that other standards such as ASVS offer. And that is by design. As the document states: "this is not a one-size-fits-all solution, in either content or priority" [14, p. 3]. The CIS controls provide a baseline for a strong security foundation that applies to all industries, from healthcare to retail, and seek to secure various asset types, such as the application, network, data, users and devices.

3.4.2 Application Verification Security Standard

The following section is an in-depth description of the Application Security Verification Standard. ASVS was developed by the Open Web Application Security Project (OWASP), a non-profit, community-driven organization dedicated to web application security [70]. In order words, ASVS was created by developers for developers. The latest version, 4.0.1, was released in March of 2019. The objective of the standard is to "normalize the range in the coverage and level of rigor available in the market when it comes to performing Web application security verification using a commercially-workable open standard" [47]. What this means in practice is that the

standard provides measures that can be adapted to any project of any size.

The applications of ASVS are many. One listed by OWASP is to use ASVS as a "blueprint to create a Secure Coding Checklist specific to your application, platform or organization" [9, p. 10]. Another, as a security architecture guidance, or for secure development training. For this project, the security audit route was taken.

Verification levels

ASVS operates with three ranges of levels to help developers classify or describe the size of their project. Each tier offers additional security requirements targeted at more security-critical projects (see figure 3.1). The levels are:

- Level 1. According to the document, level 1 is the "bare minimum that all applications should strive for" [9, p.10]. It is also the only level that can be checked either by using automatic tools or manually, without access to the source code.
- Level 2. This is for most applications. Projects that should strive for this level of verification are applications that process transaction data, store personal information about individuals or process health care information.
- Level 3. This is the highest verification that can be achieved. Projects that should strive for this level of verification are applications that require a significant level of security. This can include health and safety, military or day-to-day governance in critical infrastructure.

Security requirement sections

As of the 2019 revision, there are 14 security requirement sections. Each section has controls typically grouped into subsections that are of similar functionality. The session management category, for example, consists

#	Description	L1	L2	L3	CWE	NIST §
2.1.1	Verify that user set passwords are at least 12 characters in length. (C6)	✓	✓	✓	521	5.1.1.2
2.1.2	Verify that passwords 64 characters or longer are permitted. (C6)	✓	✓	✓	521	5.1.1.2
2.1.3	Verify that passwords can contain spaces and truncation is not performed. Consecutive multiple spaces MAY optionally be coalesced. (C6)	✓	✓	✓	521	5.1.1.2

Figure 3.1: Example of ASVS Authentication controls

of 7 subcategories, such as cookie-based requirements and token-based requirements. This split helps the user in tailoring the standard to their project, by selecting sections and subsections that apply to their application. For each category and sub-category of requirements, there is a description, along with external references for further reading.

Below is a short description of three of those sections.

V2. Authentication

Authentication is concerned with the authentication between the user and the service. This concerns, for example, how passwords are generated, which cryptographic algorithms are used, and how those passwords are stored and encrypted. An example control is to verify that passwords are at least 12 characters in length.

V5. Validation, Sanitization and Encoding

The most common web application weaknesses are those concerning a lack of input validation from the client, which leads to several well-known vulnerabilities such as Cross-Site Scripting (XSS) and SQL injection. An example control in this section is to verify that unstructured data is sanitized, i.e. the input is checked against allowed characters and length.

V13. API and Web Service

These are measures to ensure that applications that use API's are properly secured. This requires proper authentication, session management, and authorization. A possible situation that could occur is a user sending malicious data as part of the POST payload. To combat this, input validation of all parameters is required.

The remaining security requirement sections are

- V1. Architecture, Design and Threat Modeling
- V3. Session Management
- V4. Access Control
- V6. Stored Cryptography
- V7. Error Handling and Logging
- V8. Data Protection
- V9. Communications
- V10. Malicious Code
- V11. Business Logic
- V12. File and Resources
- V14. Configuration

Note that the 3.0 version of the ASVS standard contained a mobile section, which in the 4.0 release has been retired in favor of a separate checklist dedicated entirely to mobile security. The document also contains an Internet of Things (IoT) appendix, which is part of a separate OWASP IoT project. However, it is not considered part of the main branch, and due to its irrelevance to the DHIS2 project, it was not included in the audit.

3.5 Security and privacy regulations

3.5.1 Security regulations

Security regulations are directives on how information technology and computer systems should be protected [19]. There are industry-specific regulations, such as the Payment Card Industry Data Security Standard (PCI-DSS) [50] for the banking industry or the HIPAA for the U.S. health sector [62], and then there are general data protection laws. In Norway, the EU's GDPR (General Data Protection Regulation) applies [29]. Many African countries have taken inspiration from GDPR and implemented regulations with a similar set of laws [36]. Therefore, the GDPR is worth exploring with respect what it says on information security.

3.5.2 General Data Protection Regulation (GDPR)

The GDRP is a privacy and security law that came into force on May 25, 2008, with the mission to "modernize laws that protect the personal information of individuals" [28]. Though formed by the European Union (EU), its laws apply to all who target and collect data on EU's citizens [15]. The stated goal of the legislation is to *harmonize* data privacy laws across Europe, in addition to giving greater data protection and rights to individuals [29].

GDPR covers personal and sensitive personal data. Personal data in this context means a piece of information that can be used to identify a person, such as a name, address et cetera, whereas sensitive data encompasses genetic data, religion, sexual orientation. Of particular interest to this thesis is Article 25, which communicates requirements for data protection (privacy) **by design** and **by default** when handling sensitive data [10]. Both concepts share similarities, but take different approaches to data protection. Closely related to Security by Design as defined in section 3.1.1, *Data Protection by Design* is centered on data protection rather than software

development in general [61], encouraging organizations to "implement technical and organizational measures, at the earliest stages of the design of the processing operations, in such a way that safeguards privacy and data protection principles right from the start" [67]. *Data Protection by Default* is concerned with ensuring that personal data has the highest privacy protection, e.g. short storage period, limited accessibility, and more information is not disclosed than necessary.

3.6 Security requirements and threats to an HMIS

The digital revolution has brought many benefits to the health sector, including the ability to track and mitigate disease outbreaks at a speed hitherto impossible, but it has also created a new threat landscape [12]. This section examines the security threats to DHIS2 and the health industry in general.

3.6.1 Security in an HMIS

With the digitization of health information, this has created new tempting targets for hostile actors, seeking to exploit vulnerabilities in today's digital systems [12]. Health data stored in electronic form is today at its greatest risk, particularly in third-world countries who may lack the infrastructure to respond or recuperate from an attack. The type of actors and their motivation, for example, could be "states wishing to exaggerate or cover up an outbreak by hacking these systems or falsifying data within them; mask an ongoing biochemical or bioweapons attacks; or prosecute a politically motivated attack on an individual or group through the corruption of health data" [12, p. 1].

Breach of integrity and availability are the greatest threats to digital systems [12]. For example, changing a persons' blood type in the HMIS database could have severe consequences for a patient's health, whereas

the availability of DHIS2 could impact the health worker's ability to record the correct data (though DHIS2 supports offline mode). This would affect the government's or health worker's ability to monitor the health status of its citizens, and hinder their ability to track and monitor a pandemic.

A health system, like any other software, can be compromised through a variety of attacks, such as phishing, malware, ransomware, SQL injection, Cross-Site Request Forgery, Denial of Service (DoS), et cetera. Concerning DHIS2, the authors of *Weaponizing Digital Health Intelligence* list insufficient encryption and software errors as the biggest challenges to DHIS2's security [12]. The encryption concern deals with the implementors' reliance on cloud services such as Amazon and their unfamiliarity with the vulnerabilities found in those services. The second issue: software errors, stresses the importance of a secure software product. An error, a bug, could give absolute control to the attacker, and thus access to the entire database.

3.6.2 Regulatory considerations when selecting a standard

Depending on the country or political union in which software is operated, it may be subject to local regulations and laws. These compliance frameworks would set the conditions and guidelines for how organizations should store, transmit and process personal information. In Europe, there is the GDPR, which covers all personal information. In the United States, there is HIPAA for the health sector. The story is entirely different in third-world countries. For many of the countries that use DHIS2 as their main HMIS, regulations are either non-existing or in their infancy. Still, data protection laws in Africa have seen an increased push in the last 20 years, particularly in the last decade [37]. As of this writing, 25 out of 54 African countries have passed some form of data protection laws, such as Uganda, Nigeria, and Egypt [40]. India, too, has seen an increased push to regulate citizen data, with a new, GDPR-inspired data protection bill proposed at the end of 2019 [35].

3.7 Summary of Literature

Security is important in the development process and of the finished product. To achieve confidence that your application is secure and followed methods endorsed by the international security community, a standard can be followed, of which there are many. One is ASVS, which will be used in this thesis to assess DHIS2's security. ASVS was designed for the web, providing a large selection of controls for web applications and web services of all types. The leveled structure of this standard allows for both larger and smaller projects to scale and achieve verification.

Chapter 4

Research approach

The following chapter will introduce the research approach and the methodologies used in this thesis. It will begin with a quick recap of the project, followed by a methodology section describing the research strategy. Next, an overview of the general methods applied to all phases of the data gathering process will be given, followed by a closer look at each phase in detail. Next will be a brief overview of how the data is analyzed and how it will be used to answer the research question. Finally, the chapter will conclude with a few reflections on the research approach.

In this thesis, the words "assessment" and "audit" are often used interchangeably, although each word has a slightly different meaning. According to Abdel-Aziz, “auditing is measuring something against a standard, while assessing is determining how good or bad something is, but not necessarily measuring it against a specific standard”, and that while “the main focus of an audit is to check for compliance, the main focus of an assessment is to help the organization improve its security posture” [3, p. 4]. An audit is a type of security assessment. The main task of this thesis is an audit of DHIS2 using ASVS. But in doing the audit, we also seek to give the results some context, and determine if they are "good or bad", and to provide guidance on how to use that information to improve DHIS2’s "security posture". In other words, the thesis is an assessment of DHIS2 using

an auditing approach.

4.1 Project summary

The original idea of this project centered on continuous security testing, using ASVS as a secure coding checklist during the development process. Throughout the research process and discussion held with my supervisors, the research aims were redefined to what it is now: an audit of DHIS2's security controls and a look at the benefits and challenges to assessing DHIS2 using ASVS. The research approach was also highly influenced by the availability and willingness of HISP to contribute to this project. The plan was for the audit to be completed before or during the summer of 2019, by the participants in their own time, without my intervention. Then, group interviews were considered but were unrealized due to scheduling conflicts.

The main source of data for this project came from the ASVS audit, the nature of the study being of a qualitative nature, but with quantitative results. What this means is that while the results contain a measure of statistical data in the form of failed and passed controls, the majority of the data comes from the one-to-one interviews held with DHIS2 team. The quantitative results are used to answer the first objective of this thesis: *how does DHIS2's security compare against ASVS?* This is used in conjunction with the qualitative approach to give context to the results, highlighting patterns and themes. The data from the interviews would tell a story, one of people with different positions at HISP, with a different approach and possibly contradictory opinions. The same data, together with personal experiences of conducting the audit, is then used to answer the second objective of this thesis.

4.2 Methodology

The methodological approach of this project is the ASVS audit. This process will utilize two modes of data collection: qualitative and quantitative. Both of these approaches are described below. Also, the choice to go with ASVS is argued for.

4.2.1 Qualitative and quantitative

Research methods are generally classified in two ways: a qualitative and a quantitative approach. There are several ways to distinguish between them. One such distinction, made by Shields and Twycross, defines qualitative as a "method used when something needs to be measured, while qualitative methods are used when a question needs to be described and investigated in depth" [59, p. 24]. While a measure of quantitative data is part of this thesis, and answers the first objective, the research of this project falls under the qualitative canopy, as the main source of data originates from the one-to-one interviews conducted with the HISP team. The research is driven by the need to examine how various people at the HISP team evaluate the applicability or scoring of security requirements, based on unique and possibly conflicting points of view. The qualitative approach allows us to capture this form of data, as it is best suited for the "collection, organization, and interpretation of textual material derived from talk or conversation" [32, p.109].

The qualitative approach best suited to record this form of data is the interview, considered to be the most commonly used qualitative data collection tool [17][7]. Using an excel spreadsheet of ASVS controls, this project will utilize the qualitative approach to interview people who work at HISP. By inquiring how developers, security engineers et cetera evaluate security aspects of their system, we form stories, stories that help paint a detailed picture of DHIS2 and the process of conducting a security audit. By examining the quantitative results and the qualitative data gathered

through the interviews, this thesis can address the second objective and discuss the strengths and weaknesses of using ASVS to assess DHIS2.

4.2.2 Why ASVS?

Though the project was presented with ASVS in mind, it was important to consider other options and give a broader look at the standards landscape. In the end, the selection of ASVS came down to one deciding factor: the standard's relevance and applicability to the development of the DHIS2 core and apps. The main, deciding factor is OWASP's focus on web security, but other factors played a role as well.

First, the focus on the development of DHIS2 meant that only technical controls were directly applicable to DHIS2. Although HISP assists in setting up local instances of DHIS2, they cannot provide nor enforce physical and administrative guidelines on the implementors. An ISMS, in general, would a perfect fit for DHIS2, with its focus on information security and day-to-day security policies and procedures in the handling of that information. In theory, any governmental or health group is free to implement a standard such as ISO 27001 to guide their administrative policy. However, that is beyond the reach of the development team, and outside the scope of this thesis.

Second, all the standards presented in section 3.4 have different aims. The goal of the CIS controls, for instance, is to protect against cyberattacks. That is an essential part of every standard listed in this thesis, but it does not address information security directly, nor the root cause of security vulnerabilities: a secure software development life cycle. Both the ISO/IEC 27000-series and NIST's SP 800-53 address all three security controls types: technical, administrative, and physical, only one of which is relevant to DHIS2. The relative simplicity of ASVS is important too. The sheer size of standards like the ISO 27001 and 27002 make it too complicated for a thesis of this size, as translating its numerous controls into checkable

requirements would be a massive task in on itself.

4.3 General methods for data collection

This section outlines the general data collection methods used for this project. This will include an overview of the interview method, auditing methods, tools used to collection the data, participant selection, and the general approach to the interviews.

Note that since the last round of interviews in early December, several requirements have since been fixed. To avoid having to continuously update the results and the text on which it is based, a line must be drawn. The ASVS results presented in this thesis will, therefore, be based on the December 2019 ASVS findings, as will the discussion that follows. A short summary of the updated scoring is presented in section 5.1.11.

4.3.1 Interviews

The interview method comes with a set of preconceptions. Coughlan defines the one-to-one interview as offering "the researcher the opportunity to interpret non-verbal cues through observation of body language, facial expression, and eye contact" [17, p. 310]. The aim of this study is only to capture the spoken or written word about technical matters, and is not a study of the participants themselves. The interview method was chosen because it is particularly useful for getting the story behind a participant's choice or experience [46]. As Grossoehme notes, interviews "explore experiences of individuals, and through a series of questions and answers, the meaning individuals give to their experiences" [32, p. 110]. The interview approach taken for this thesis is elaborated below.

Semi-structured interview format

There are three interview types to consider: structured, semi-structured and unstructured. The interview type that you choose depends on the "research question and the information needed to provide holistic answers to these questions" [7, p. 150]. The type chosen for this thesis is a semi-structured interview approach, a hybrid of structured and unstructured. Less rigid than structured, and more focused than unstructured, this type of interview is where "questions are pre-planned prior to the interview but the interviewer gives the interviewee the chance to elaborate and explain particular issues through the use of open-ended questions" [7, p. 151]. This form of an interview follows an interview schedule with predetermined topics and questions. However, as Coughlan explains, "the flexibility of the semi-standardized interview allows the interviewer to pursue a series of less questioning and also permits the exploration of spontaneous issues raised by the interviewee to be explored." [17, p. 310].

The reasoning for selecting the semi-structured approach is two-fold. First, structured interviews are considered too strict, which removes the "flexibility of the interviewer in terms of being able to interrupt, and the interviewee to elaborate, is restricted" [7, p. 151]. They are akin to a questionnaires survey, whereby there are no deviations from the sequence of questions. While the interview is structured in the sense that there are a set of predefined questions, and the interviewee is asked to discuss each of the security requirements in sequence, the questions themselves are broad, and the participant is allowed to take the discussion to hitherto unthought-of places. The depth of the responses is highly dependent on the participant's familiarity with the topic. Due to this fact, it is not a requirement that all questions about a particular control are answered.

4.3.2 Auditing methods

This section outlines the methods used during the auditing process. Data collection in this thesis comes in three phases. The ability to be able to proceed to the next phase was dependent on the completion of the previous phase. The three phases of data collection can be described as follows:

1. **Applicability**. In the first phase, security controls applicable to the DHIS2 platform from ASVS are identified and checked as either applicable or non-applicable.
2. **Scoring**. The second phase consists of scoring the applicable security controls as either pass or fail.
3. **Ranking**. The final phase is to rank and discuss the importance of each failed control.

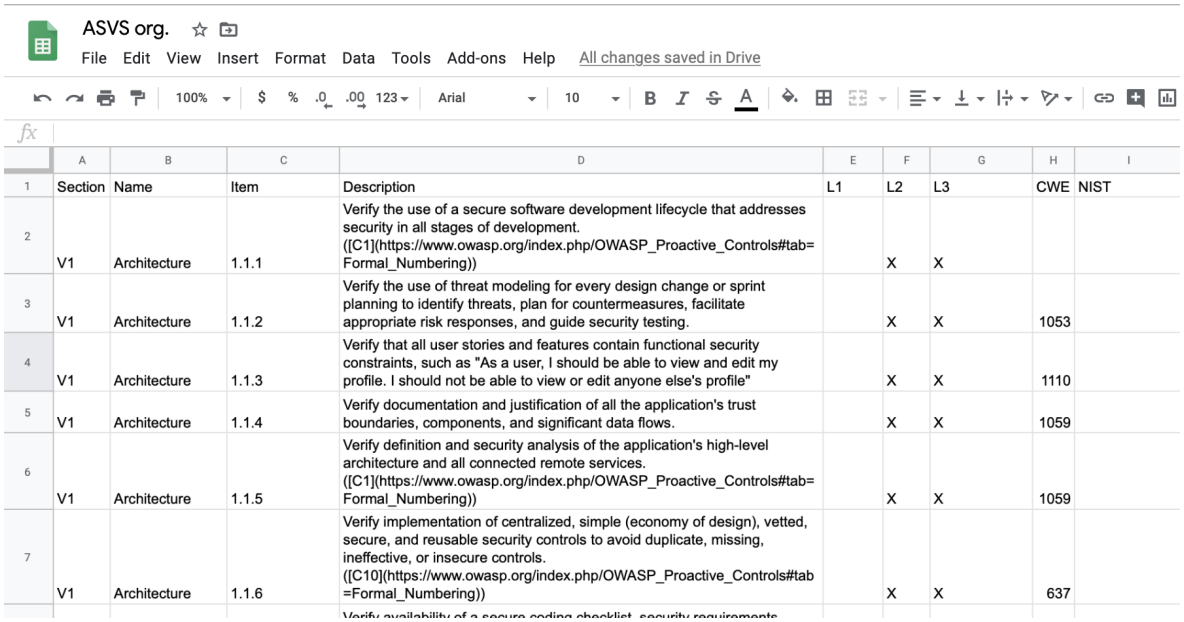
The first two phases are part of the ASVS auditing process. The third phase was added to better address the second objective of this thesis. By having the participants rank the importance of the failed controls, this thesis is able to reflect on the relevance of ASVS controls and ASVS in general.

All phases share similarities in how the participants were chosen and how the interviews were conducted. The following sections describe the similarities, including a brief overview of the participants, tools used for data gathering, and the procedure.

4.3.3 Tools used for data collection

ASVS contains 286 security controls spread across 14 chapters. To present the list in an easily readable and available manner, an excel spreadsheet was utilized, downloaded directly from OWASP's official ASVS website. This spreadsheet had each control categorized into one of the 14 sections, and gives three columns for each level, with a x indicated whether given control is applicable to level 1, level 2 or level 3. For each phase, this spreadsheet was forked and modified with new columns to gather new

data, and each modification was in turn duplicated for every participant. Each of these files was kept on a personal Google Drive account.



ASVS org.

File Edit View Insert Format Data Tools Add-ons Help All changes saved in Drive

	A	B	C	D	E	F	G	H	I
1	Section	Name	Item	Description	L1	L2	L3	CWE	NIST
2	V1	Architecture	1.1.1	Verify the use of a secure software development lifecycle that addresses security in all stages of development. ([C1](https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering))		X	X		
3	V1	Architecture	1.1.2	Verify the use of threat modeling for every design change or sprint planning to identify threats, plan for countermeasures, facilitate appropriate risk responses, and guide security testing.		X	X	1053	
4	V1	Architecture	1.1.3	Verify that all user stories and features contain functional security constraints, such as "As a user, I should be able to view and edit my profile. I should not be able to view or edit anyone else's profile"		X	X	1110	
5	V1	Architecture	1.1.4	Verify documentation and justification of all the application's trust boundaries, components, and significant data flows.		X	X	1059	
6	V1	Architecture	1.1.5	Verify definition and security analysis of the application's high-level architecture and all connected remote services. ([C1](https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering))		X	X	1059	
7	V1	Architecture	1.1.6	Verify implementation of centralized, simple (economy of design), vetted, secure, and reusable security controls to avoid duplicate, missing, ineffective, or insecure controls. ([C10](https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering))		X	X	637	

Figure 4.1: Original ASVS spreadsheet by OWASP

4.3.4 Participant selection

The goal with selecting participants is to do "purposeful sampling" [7, p. 152], i.e. pick people at DHIS2 with different areas of expertise that can provide different viewpoints for the assessment. For example, one participant in this study is a backend developer, while another helps with implementation. Whereas the former might evaluate a security control form a technical point of view, the latter might look at it from the imlementor's point of view. The goal was to get as many people involved as possible, but this was not a priority. Nor was it necessarily required. Alsaawi argues that having several people involved is good, but that you should be careful of saturation, wherein "the interviewer begins to hear the same information he/she has already obtained from the previous interviews" [7, p. 152]. The DHIS2 team is relatively small, with few people who have a technical overview of the entire system. For example, there

are presently two backend tech leads. The remaining backend developers mainly work on specific modules, and therefore might lack the same level of knowledge of the entire system to be able to contribute to the audit.

In total 5 people participated in this project, all of whom are either a part of the DHIS2 development team or part of HISP, all with different roles within the organization. Not all participated in each phase of the project, but for every phase, there were always at least two participants. The following is an overview of the five participants.

Participant 1. Participant number 1 is a backend developer. Having joined the company in 2015 on a part-time basis, he is now one of the two backend tech leads.

Participant 2. Number 2 has worked as a frontend tech lead and product manager for the platform's web app development since 2017.

Participant 3. The third participant joined the DHIS2 core team in late 2019 as a software security engineer, whose job is to oversee the entire security framework of the platform.

Participant 4. The fourth participant is responsible for quality assurance for new releases of DHIS, and has been with the company for approximately two years. In this text, this participant is commonly referred to as the release manager.

Participant 5. The last participant is a senior implementation engineer and was part of the group who proposed this project. This person works directly with implementation and helps set up new instances of DHIS2, and has a good overview of the challenges with security in DHIS2 at the implementation level. In this text, this person will be referred to as the implementor or implementation engineer.

4.3.5 Approach to interviews

The interview approach differed for each phase and with each participant, but there were also similarities, which this section will highlight. First, each participant was given a clean copy of an ASVS spreadsheet. During the interview, the interviewer would sit down with the interviewee and go through all of the 286 controls on the list. In a couple cases, this was also done through a video call. What is meant by *clean copy* is that the participants were not aware of each other's answers. This was done for two reasons. The first is so a participant would not be influenced by another participant's answer, and the second was to gather an alternative opinion and point of view on an already checked and/or unchecked control. Given that group sessions was not an option, it was important to gather each set of data in isolation. However, the results from one session were occasionally used during another session to gather more data on specific controls. For example, if one participant was unsure about a control's applicability or pass/fail status, or a particular comment was unclear, then an effort was made on the interviewers part to draw the interviewees attention towards that control.

Second, during each interview, the participant would either write down their opinion directly in the spreadsheet themselves, or the interviewee would. Simple note-taking is generally not ideal, due to the large amount of data that can come from an interview [7]. However, that which was written down was often a summarization of participant's words and not a direct transcription of every word spoken. Recording audio, while also registering which of the 286 controls the participant was currently looking at, would have been problematic and could possibly have prolonged the interview. Their spoken word was written down and refined in the moment.

When needed, an email was sent to a participant asking to clarify or elaborate their opinion on a particular control.

4.4 Approach to the Applicability Phase

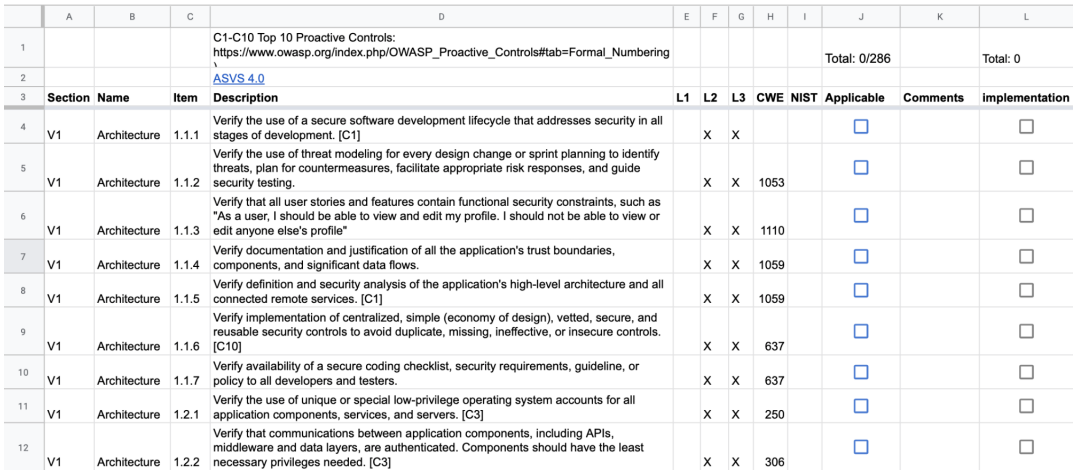
The first phase of this project is to establish which ASVS security requirements apply to the DHIS2 platform. The spreadsheet will differentiate between four different states of applicability and non-applicability.

- **Software**. The first state are controls that are directly applicable to and can only be addressed by the DHIS2 development team.
- **Implementation**. The second state concerns controls that target security aspects outside of the development team's control, such as server setup, firewalls, and configuration. The responsibility for securing these controls lies with the implementors.
- **Semi-applicable**. This revolves around controls that apply both to the development team and the implementors.
- **Non-applicable**. The last state are controls that do not apply to DHIS2, either due to irrelevant functionality or other causes.

While controls that are implementation only are technically non-applicable - and the results chapter will list them as such, it was important to register the cause for why they are not applicable. The interesting context of DHIS2 is the difference between software and implementation, which separates DHIS2 from other software products.

4.4.1 The checklist

The original ASVS spreadsheet was forked and modified to include additional columns, as seen in figure 4.2. First is an *Applicable* column, used to register a control as applicable to software and the responsibility of the development team. Second is a column for comments. And the third is to register a control as applicable to *Implementation*, either exclusively or in combination with software.



	A	B	C	D	E	F	G	H	I	J	K	L
1				C1-C10 Top 10 Proactive Controls: https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering						Total: 0/286		Total: 0
2				ASVS 4.0								
3	**Section**	**Name**	**Item**	**Description**	**L1**	**L2**	**L3**	**CWE**	**NIST**	**Applicable**	**Comments**	**implementation**
4	V1	Architecture	1.1.1	Verify the use of a secure software development lifecycle that addresses security in all stages of development. [C1]		X	X			☐		☐
5	V1	Architecture	1.1.2	Verify the use of threat modeling for every design change or sprint planning to identify threats, plan for countermeasures, facilitate appropriate risk responses, and guide security testing.		X	X	1053		☐		☐
6	V1	Architecture	1.1.3	Verify that all user stories and features contain functional security constraints, such as "As a user, I should be able to view and edit my profile. I should not be able to view or edit anyone else's profile"		X	X	1110		☐		☐
7	V1	Architecture	1.1.4	Verify documentation and justification of all the application's trust boundaries, components, and significant data flows.		X	X	1059		☐		☐
8	V1	Architecture	1.1.5	Verify definition and security analysis of the application's high-level architecture and all connected remote services. [C1]		X	X	1059		☐		☐
9	V1	Architecture	1.1.6	Verify implementation of centralized, simple (economy of design), vetted, secure, and reusable security controls to avoid duplicate, missing, ineffective, or insecure controls. [C10]		X	X	637		☐		☐
10	V1	Architecture	1.1.7	Verify availability of a secure coding checklist, security requirements, guideline, or policy to all developers and testers.		X	X	637		☐		☐
11	V1	Architecture	1.2.1	Verify the use of unique or special low-privilege operating system accounts for all application components, services, and servers. [C3]		X	X	250		☐		☐
12	V1	Architecture	1.2.2	Verify that communications between application components, including APIs, middleware and data layers, are authenticated. Components should have the least necessary privileges needed. [C3]		X	X	306		☐		☐

Figure 4.2: ASVS applicability spreadsheet

4.4.2 The procedure

Four people participated in this phase of the project. The following paragraphs describe the process for each participant.

Implementation engineer. The participant lives in the United Kingdom, and the interviews were therefore done through video calls. Due to time limitations, all but the last two chapters of ASVS were covered.

Release manager. This interview took place at UiO. Here, approximately half the controls were covered.

Frontend developer. This interview took place at UiO. All comments were written by the participant during the interview.

Backend developer. This session occurred simultaneously with the scoring phase, since contact with the participant was not established until after the first phase had been completed. During this session, several controls skipped over by the previous participants were looked over.

4.4.3 Combining the data

The data from the participants was copied into a single document that lists their responses side-by-side for comparison. Then, a set of criteria was

applied to combine the responses and create a single column of applicable requirements that can be used in the next phase. An example of this is shown in figure 4.3. The frontend developer, for example, selected 242 controls as applicable out of 286.

			C1-C10 Top 10 Proactive Controls: https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_				Total: 242/286	Total: 132/286	Total: 196/286	Total: 256/286	Total: 257/286
			ASVS 4.0				Front-end	Release m.	Implementor	Backend	Combined
Section	**Name**	**Item**	**Description**	**L1**	**L2**	**L3**	**Applicable**	**Applicable**	**Applicable**	**Applicable**	**Applicable**
V1	Architecture	1.1.1	Verify the use of a secure software development lifecycle that addresses security in all stages of development. [C1]		X	X	☑	☑	☑	☐	☑
V1	Architecture	1.1.2	Verify the use of threat modeling for every design change or sprint planning to identify threats, plan for countermeasures, facilitate appropriate risk responses, and guide security testing.		X	X	☑	☑	☑	☑	☑
V1	Architecture	1.1.3	Verify that all user stories and features contain functional security constraints, such as "As a user, I should be able to view and edit my profile. I should not be able to view or edit anyone else's profile"		X	X	☑	☑	☑	☑	☑
V1	Architecture	1.1.4	Verify documentation and justification of all the application's trust boundaries, components, and significant data flows.		X	X	☑	☑	☑	☑	☑
V1	Architecture	1.1.5	Verify definition and security analysis of the application's high-level architecture and all connected remote services. [C1]		X	X	☑	☑	☑	☑	☑

Figure 4.3: ASVS applicably spreadsheets combined

The following is the criteria by which the collected checkboxes were evaluated:

- A control remains checked if all participants have checked it.
- A control remains checked if three of the four participants have checked it.
- A control remains checked if two of the four participants have checked it.

For example, in figure 4.3, 3 of 4 participants selected control 1.1.1 as applicable. Therefore, the control is checked in the rightmost column.

For all these criteria, there are exceptions. A comment may be provided that tips the scale in one direction or another. For example, a control wherein only one participant checked it as applicable is checked as applicable in the final list only if a good argument was provided. Ideally, the interviewer, whose knowledge of DHIS2 is that of an outside party, should not be the one to evaluate the strength of the argument. In a few instances, however, strength of the argument could not be ignored and the comment was taken

into consideration. Where comments and opinions differed, and there was no clear resolution, the control was checked, to be re-evaluated in the next phase. The same set of criteria was applied on the implementation column.

4.5 Approach to the Scoring Phase

The second phase of this project is to score the applicable controls from the previous phase, as either pass or fail. While controls for all three levels of ASVS will be graded, the primary goal is to establish a benchmark against the first level.

4.5.1 The checklist

The applicable results from the previous phase were copied into a new spreadsheet and new columns were added. A **pass** column was added to register when a control passed, and an **uknown** column was used to register when a participant did not have an answer. This is show in figure 4.4.

			C1-C10 Top 10 Proactive Controls: https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal						Total: 258/286	Pass: 0/258			Total: 79
			ASVS 4.0										
Section	**Name**	**Item**	**Description**	**L1**	**L2**	**L3**	**CWE**	**NIST**	**Applicable**	**Pass**	**Unknown**	**Comments**	**implementation**
V1	Architecture	1.1.1	Verify the use of a secure software development lifecycle that addresses security in all stages of development. [C1]		X	X			☑	☐	☐		☐
V1	Architecture	1.1.2	Verify the use of threat modeling for every design change or sprint planning to identify threats, plan for countermeasures, facilitate appropriate risk responses, and guide security testing.		X	X	1053		☑	☐	☐		☐
V1	Architecture	1.1.3	Verify that all user stories and features contain functional security constraints, such as "As a user, I should be able to view and edit my profile. I should not be able to view or edit anyone else's profile"		X	X	1110		☑	☐	☐		☐
V1	Architecture	1.1.4	Verify documentation and justification of all the application's trust boundaries, components, and significant data flows.		X	X	1059		☑	☐	☐		☐
V1	Architecture	1.1.5	Verify definition and security analysis of the application's high-level architecture and all connected remote services. [C1]		X	X	1059		☑	☐	☐		☑
V1	Architecture	1.1.6	Verify implementation of centralized, simple (economy of design), vetted, secure, and reusable security controls to avoid duplicate, missing, ineffective, or insecure controls. [C10]		X	X	637		☑	☐	☐		☐
V1	Architecture	1.1.7	Verify availability of a secure coding checklist, security requirements, guideline, or policy to all developers and testers.		X	X	637		☑	☐	☐		☐
V1	Architecture	1.2.1	Verify the use of unique or special low-privilege operating system accounts for all application components, services, and servers. [C3]		X	X	250		☐	☐	☐		☑
V1	Architecture	1.2.2	Verify that communications between application components, including APIs, middleware and data layers, are authenticated. Components should have the least necessary privileges needed. [C3]		X	X	306		☑	☐	☐		☑

Figure 4.4: ASVS scoring spreadsheet

4.5.2 The procedure

Three people participated in this phase of the project. The discussion around each control was kept open, though a few guidelines were given before the interviews began. In particular, the participants were encouraged to comment on the failed controls. For both the frontend and backend developer, the interview sessions took place at their offices at UiO. The third participant, the security engineer, was initially only approached to contribute to the third phase of the project. However, several controls had been left unresolved from the second phase due to conflicting answers. The security engineer offered to provide a third opinion on these controls. The participant was given full editing rights to a copy of the results, and scored them at his own time.

4.5.3 Combining the data

The data from the participants was copied into a single document that lists their responses side-by-side for comparison. Then, a set of criteria was applied to combine the responses and create a single column, to set the controls in one of four states: pass, fail, unknown, or conflict. An example of which is shown in figure 4.5, where the first three columns belong to the participants, and the rightmost column is the combined result.

V8	Data	8.1.4	Verify the application can detect and alert on abnormal numbers of requests, such as by IP, user, total per hour or day, or whatever makes sense for the application.		X	X	770		☑	☐	☐	☐	☐
V8	Data	8.2.1	Verify the application sets sufficient anti-caching headers so that sensitive data is not cached in modern browsers.	X	X	X	525		☑	☑	☐	☐	☑
V8	Data	8.2.2	Verify that data stored in client side storage (such as HTML5 local storage, session storage, IndexedDB, regular cookies or Flash cookies) does not contain sensitive data or PII.	X	X	X	922		☑	☑	☑	☐	☑
V8	Data	8.2.3	Verify that authenticated data is cleared from client storage, such as the browser DOM, after the client or session is terminated.	X	X	X	922		☑	☑	☐	☐	☑
V8	Data	8.3.1	Verify that sensitive data is sent to the server in the HTTP message body or headers, and that query string parameters from any HTTP verb do not contain sensitive data.	X	X	X	319		☑	☑	☑	☐	☑
V8	Data	8.3.2	Verify that users have a method to remove or export their data on demand.	X	X	X	212		☑	☐	☑	☐	☐
V8	Data	8.3.3	Verify that users are provided clear language regarding collection and use of supplied personal information and that users have provided opt-in consent for the use of that data before it is used in any way.	X	X	X	285		☑	☐	☑	☐	☐
V8	Data	8.3.5	Verify accessing sensitive data is audited (without logging the sensitive data itself), if the data is collected under relevant data protection directives or where logging of access is required.		X	X	532		☑	☐	☐	☐	☐
V8	Data	8.3.6	Verify that sensitive information contained in memory is overwritten as soon as it is no longer required to mitigate memory dumping attacks, using zeroes or random data.		X	X	226		☑	☐	☐	☐	☐

Figure 4.5: ASVS scoring spreadsheets combined

The following is the criteria by which the checkboxes were evaluated:

- A control is passed (green) if two out of three participants passed it.
- A control is passed (green) if checked by at least one participant and skipped by the rest.
- A control is marked as a conflict (yellow) if the responses differ.
- A control is marked as unknown (pink) if all participants skipped the control.
- A control is marked as failed (red) if failed by majority.

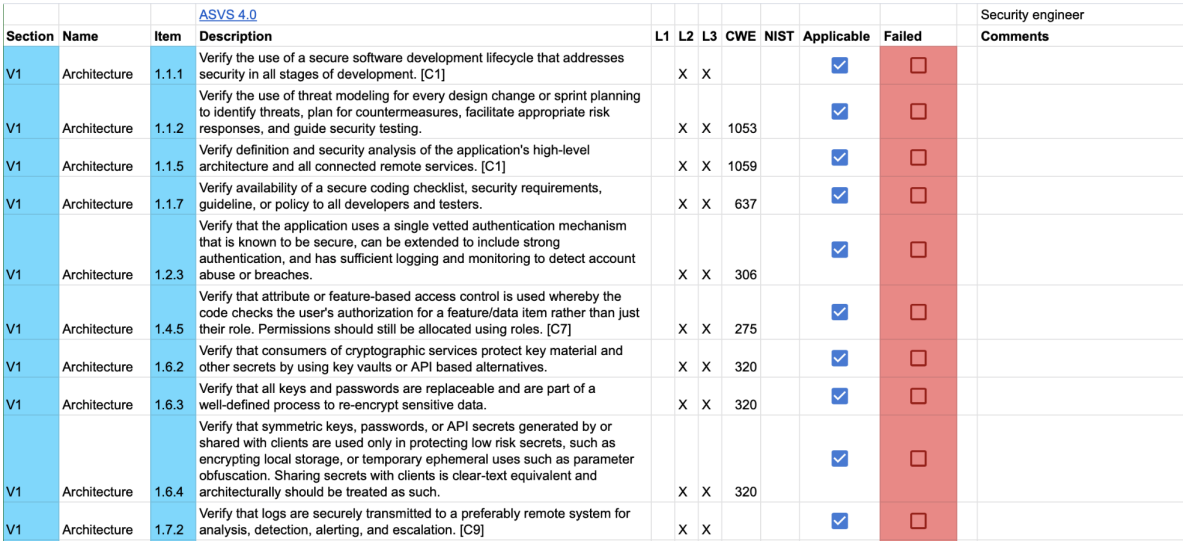
Unlike in the previous phase, no attempt was made by the author to resolve the conflicts by evaluating the comments given by the participants. The final results should stand as they are, a reflection of the agreements and disagreements between the participants.

4.6 Approach to the Ranking Phase

In the third and final phase of this project, we rank the severity and importance of the failed controls from the previous phase. This phase is not part of the official audit process. Rather, the purpose of this phase is primarily to gather data on the relevance of ASVS controls. The idea is to replicate a process or discussion that could occur after an audit is completed where the participants deliberate and prioritize which issues are most critical to fix. All controls that are unequivocally failed are evaluated here, regardless of level.

4.6.1 The checklist

The aggregated spreadsheet from the previous phase was copied and a new column added for comments.



			ASVS 4.0								Security engineer
Section	**Name**	**Item**	**Description**	**L1**	**L2**	**L3**	**CWE**	**NIST**	**Applicable**	**Failed**	**Comments**
V1	Architecture	1.1.1	Verify the use of a secure software development lifecycle that addresses security in all stages of development. [C1]		X	X			☑	☐	
V1	Architecture	1.1.2	Verify the use of threat modeling for every design change or sprint planning to identify threats, plan for countermeasures, facilitate appropriate risk responses, and guide security testing.		X	X	1053		☑	☐	
V1	Architecture	1.1.5	Verify definition and security analysis of the application's high-level architecture and all connected remote services. [C1]		X	X	1059		☑	☐	
V1	Architecture	1.1.7	Verify availability of a secure coding checklist, security requirements, guideline, or policy to all developers and testers.		X	X	637		☑	☐	
V1	Architecture	1.2.3	Verify that the application uses a single vetted authentication mechanism that is known to be secure, can be extended to include strong authentication, and has sufficient logging and monitoring to detect account abuse or breaches.		X	X	306		☑	☐	
V1	Architecture	1.4.5	Verify that attribute or feature-based access control is used whereby the code checks the user's authorization for a feature/data item rather than just their role. Permissions should still be allocated using roles. [C7]		X	X	275		☑	☐	
V1	Architecture	1.6.2	Verify that consumers of cryptographic services protect key material and other secrets by using key vaults or API based alternatives.		X	X	320		☑	☐	
V1	Architecture	1.6.3	Verify that all keys and passwords are replaceable and are part of a well-defined process to re-encrypt sensitive data.		X	X	320		☑	☐	
V1	Architecture	1.6.4	Verify that symmetric keys, passwords, or API secrets generated by or shared with clients are used only in protecting low risk secrets, such as encrypting local storage, or temporary ephemeral uses such as parameter obfuscation. Sharing secrets with clients is clear-text equivalent and architecturally should be treated as such.		X	X	320		☑	☐	
V1	Architecture	1.7.2	Verify that logs are securely transmitted to a preferably remote system for analysis, detection, alerting, and escalation. [C9]		X	X			☑	☐	

Figure 4.6: ASVS ranking spreadsheet

4.6.2 The procedure

Two participants were involved in this phase of the project: the frontend developer and the security engineer. Both interviews occurred in early December 2019, through a video call with the security engineer, and at UiO with the frontend developer. For each failed control, the participants were encouraged to answer these three questions:

- How important is this requirement? If so, why?
- Has DHIS2 experience and/or is experiencing any issues related to this control?
- Are there any challenges to implementing this control?

The participants were also asked to give a control a rank from a scale of 1 to 5, where 5 is high and 1 is low. The qualitative data would give insight into what the participants think is most important to fix and not to fix, whereas the scales would be used to get an overview of the number of critical and non-critical controls. A scale of a 1-5 range was selected due to its simplicity, which is particularly useful to get concise and direct answers that do not require interpretation or deep analysis.

4.7 Data analysis

The data analysis will be used as a means to make sense of the audit results. In analyzing the data, the first and partially the second research question is answered. To address the first objective, the quantitative results from the scoring phase are merged, and a final verdict is given on DHIS2's coverage against ASVS.

To answer the second research question, all three phases will need to be analyzed to find patterns and themes in both the quantitative and qualitative results. For example, controls will be categorized based on responses. A control relevant to implementation due to regulatory reasons, is marked to one category, and a control relevant to implementation due to configuration, is marked to another category. This separation allows us to discover controls/vulnerabilities of similar types that can be addressed through similar measures.

4.8 Reflections

4.8.1 Discussion between participants

For the interviews, the decision was made not to let the participants see each other's answers. The benefit this approach yielded the project is that each participant was given the opportunity to provide an alternative opinion or background information on a given control. The concern with letting the participants see each other's contributions was that they would be less encouraged to elaborate or provide an alternative argument if already covered by someone else.

Had the participants been able to view their predecessors' answers, this would result in a different set of data, one in which the participants would be given a chance to weigh in on the previous answers, and possibly explain why they are wrong. By reading the previous responses, the

participants would also have the opportunity to discover an angle they had not considered. In the selected approach, this is not possible, and for this reason, the interviewer has to evaluate comments that contradict one another.

However, this alternate approach is still not ideal. The last participant always has the last say, and could, in theory, be wrong in their stance. In these situations, the interviewer would still need to evaluate the comments, both the last and the ones that came before. The previous participants are also unable to give a rebuttal. For these reasons, the data in this approach, while does form more of a discussion, is not necessarily more correct. Would this alternative approach have resulted in fewer conflicting opinions? This is difficult to determine without repeating the study.

4.8.2 Ethical considerations

"Sensitive information is data that must be protected for the privacy or security of an individual, group, or organization" [58]. There was concern that releasing this thesis could potentially disclose damaging security vulnerabilities in DHIS2. Disclosing such information, and if exploited by an attacker, could result in damage to the organization, DHIS2, and indirectly to the health and personal information stored within. Before the publication of this thesis, the data was looked over by DHIS2's security engineer and the content cleared for publication. The verdict was that the majority of information on vulnerabilities is abstract, and while may provide hints, the attacker will still need to do a lot of work to exploit the vulnerabilities found in this audit.

Chapter 5

Results

The results presented in this chapter were generated from conducting the three phases of data gathering, as described in previous chapter. Due to the size of the spreadsheets, this chapter will present an abridged, summarized version of the results. The results in full can be found in Appendix A, Appendix B, and Appendix C.

The first two phases are presented in section 5.1. In presenting the results, the first two phases will answer the first research question of this thesis:

> *How does DHIS2's security compare against ASVS?*

The full context and meaning of the results will be given in the next chapter. Here, the results will be presented quantitatively and qualitatively, giving the statistical results from the audit, as well as highlighting discovered patterns and themes in the answers gathered through the interviews. First will be given an overview of the results from the entire audit, followed by a deeper dive into a select few ASVS categories. These sections will highlight four types of themes: 1) applicable controls; 2) non-applicable controls; 3) controls applicable both to software and implementation; 4) and disagreements between the participants. The results from the third phase are then presented in section 5.2.

5.1 Part 1: Security audit results

5.1.1 Applicability

The following is an overview of the results from the applicability sessions, as show in table 5.1. The full results from this phase can be found in Appendix A.

- **Applicable**. 258 out of 286 ASVS controls are applicable to the DHIS2 development team.
- **Semi-applicable**. 69 of the 258 apply to software and implementation both. These are controls that have to be implemented both by the DHIS2 development team and by the local implementors.
- **Non-applicable**. 28 controls from ASVS are non-applicable. 16 of these concern implementation and 12 functionality that DHIS2 does not have.

	Total	Implementation	Other
Controls	286		
Applicable	258	69	189
Non-applicable	28	12	16

Table 5.1: Results from the applicability phase

5.1.2 Scoring

Table 5.2 shows the results from the scoring phase. The color-coding in the table reflects the colors assigned to the four states a control could be in: passed (green), fail (red), conflict (yellow) and unknown (pink). The full results from this phase can be found in Appendix B.

- **Passed**. 174 of the 258 applicable controls were assessed as pass. This equals to approximately 67% of all controls.

- **Fail**. 50 controls were assessed as fail. This equals to approximately 19% of all controls.
- **Conflict**. 5 controls had conflicting answers from the participants.
- **Unknown**. No one could give a definite score to 29 of the controls

Applicable controls	Pass	Fail	Conflict	Uknown
258	174	50	5	29

Table 5.2: Results from the scoring phase

As the last two points illustrate, the results in this phase are inconclusive.

Scoring by level

One goal of the assessment was to see how DHIS2 compares against the first level of ASVS, labeled by OWASP as the "bare minimum that all applications should strive for" [9, p. 10]. Achieving this level (and higher levels) is important to DHIS2 given its status as a health information management platform. Table 5.3 gives an overview of the results in each level.

For the first level, 92 out of 128 applicable controls are a pass, approximately 71% of all controls. The remaining controls are either fail or unresolved. The percentage number is 62 and 53 for the second and third level respectively.

	Level 1	Level 2	Level 3
Controls	131	136	20
Applicable	128	116	15
Passed	92 (71%)	73 (62%)	8 (53%)

Table 5.3: Results from the scoring phase by level

5.1.3 ASVS V1: Architecture, Design and Threat Modeling

This chapter covers all things related to secure software development, including many aspects of secure architecture: integrity, availability, non-repudiation, confidentiality, and privacy. Spanning 14 subcategories, covering the architectural requirements of each of the base ASVS sections, this chapter is the only section with level 2 and above requirements.

Controls	Applicable	Pass	Fail
42	37	23	11

Table 5.4: Results from ASVS section V1

Applicability

As table 5.4 shows, 37 of the 42 controls in this section are applicable to DHIS2. The remaining 5 apply to implementation only, one example of which is control 1.2.1, describing the testing of "unique or special low-privilege operating system accounts for all application components, services, and servers". This control, like many others, is only relevant for someone who is hosting the platform, as affirmed by all participants. The remaining 4 are similarly unchecked, dealing with firewalls, proxies and policies, all implementor responsibility.

Of the 37 applicable to software, 17 are also partly applicable to implementation. Control 1.7.1, concerning data and activity logging, is one such example. The frontend developer writes: "for DHIS2 this is applicable for the core team, for surrounding software such as Nginx (webserver), the implementor is responsible". In other words, most things logging-related on the server-side are an implementation responsibility, who themselves need to analyze the data.

An occurrence of disagreement was with control 1.8.1, dealing with the identification and classification of data into protection levels. The implementor left is unchecked, arguing that the local governments classify

the data, not the developers. Though left unchecked, the implementor also added that it is up to the developers to allow the application to classify data, an opinion seconded by the frontend developer, who, in contrast, checked it as applicable.

A correct configuration of DHIS2 is important to ensure that the platform is secure. A control applicable to implementation in this context is 1.12.2, which states: "Verify that user-uploaded files are stored outside of the web root". According to the release manager, this is configurable, and data can be configured to be stored inside the web root. An attacker or even a local user can then willingly or unwillingly upload a malicious file onto their DHIS2 instance.

Scoring

In the scoring phase, 23 controls passed, 11 failed, and 2 are unresolved. A control of interest is 1.14.6, which illustrates the point of needing to consult with different people with different areas of expertise. The control, dealing with the use of unsupported, insecure or deprecated technologies such as Flash (among the many listed), was checked as pass by the frontend developer. The backend developer, on the other hand, failed it, stating that DHIS2 currently uses other client technology that they know to be insecure. However, the control was still marked as a yellow, due to the conflicting scores.

5.1.4 ASVS V2: Authentication

"Authentication is the act of establishing, or confirming, someone [or something] as authentic and that claims made by a person or about a device are correct, resistant to impersonation, and prevent recovery or interception of passwords" [9, p. 20].

Controls	Applicable	Pass	Fail
57	44	27	9

Table 5.5: Results from ASVS section V2

Applicability

Table 5.5 shows that 44 out of 57 controls in this section are applicable to DHIS2. The remaining 13 requirements either address functionality that DHIS2 does not have or concern implementation. With the latter, an example is with control 2.2.6 on the use of one-time passwords (OTP), lookup codes or cryptographic authenticators for replay resistance. These services, according to all participants, are not provided by DHIS2, with the backend developer noting that they use another form of protection. The question here is whether the alternative method is good enough or is better than the solution proposed by ASVS. Although the control was marked as an implementation issue by all but one of the participants, it is not immediately clear why it is an implementor responsibility.

A selection of the 44 applicable controls are semi-applicable to implementation. For example, control 2.10.1, on the safe storage of secrets and credentials, was marked by two participants as applicable to software only. The frontend developer, however, argued that while the "DHIS2 core team is responsible for not putting credentials in the source code", it is also the implementor's responsibility for storing information securely on the server. Another example is control 2.5.4, which concerns the removal of default accounts from production, such as "root" or "admin". The release manager noted that there is a built-in admin account when the system is bootstrapped, used to create other accounts, that has to be disabled by the implementors. The implementation engineer added that the implementors often do not disable it and leave it present.

Scoring

In the scoring phase, 27 controls passed, 9 failed, and 8 are unresolved. Initially, the unresolved number was higher. The security engineer was brought on board to resolve as many conflicts as possible. One such issue now resolved as fail is control 2.4.4, dealing with password hashing function *bcrypt* for which "the work factor SHOULD be as large as verification server performance will also, typically at 13". The work factor, according to the security engineer, is currently set to 10, and it cannot be reconfigured by the implementors. This illustrates the need for a varied pool of opinions and expertise.

Control 2.1.1 brings up the issue of configuration. The control asks to verify that user passwords are at least 12 characters in length. The control was failed by the frontend developer. The backend developer, on the other hand, passed it, arguing that passwords can be configured to be 12 characters long. A possible question that could be raised is whether or not the implementors should be able to configure a password to be shorter than 12 characters long. It is unclear to me if any password-length restrictions are present.

5.1.5 ASVS V3: Session Management

This ASVS chapter concerns web-based applications and stateful API's, and the mechanisms by which they control and maintain the state of a user or a device. The technique used today are session tokens, unique to each individual and ideally not guessable.

Controls	Applicable	Pass	Fail
20	19	11	4

Table 5.6: Results from ASVS section V3

Applicability & scoring

In this chapter, 19 of 20 controls are applicable to DHIS2, 11 of which pass and 4 fail. The single non-applicable control was 5.3.5, which states: "Verify that stateless session tokens use digital signatures, encryption, and other countermeasures to protect against tampering, enveloping, replay, null cipher, and key substitution attacks". Two participants marked is as non-applicable, and two marked is as applicable. The frontend developer expressed that while it is not applicable right now, stateless session cookies are being worked on. Due to this argument, the deadlock was tipped towards non-applicable. Interestingly, the language in this control assumes that stateless session tokens are already in place. Based on these semantics, the control is not applicable if stateless cookies are not used. Nonetheless, work on stateless session cookies was already in progress with the DHIS2 development team, and for this reason, it can be argued that the control should have been applicable.

5.1.6 ASVS V4: Access Control

"Authorization is the concept of allowing access to resources only to those permitted to use them" [9, p. 33]. This ASVS chapter ensures that access is granted to persons with valid credentials and that users are associated with a defined set of roles or privileges.

Controls	Applicable	Pass	Fail
10	10	8	2

Table 5.7: Results from ASVS section V4

Applicability & scoring

In this ASVS category, all 10 controls are applicable, and 2 controls fail. Here we explore a control applicable to software and scored as fail by all

participants: 4.2.2. This control aims to ensure that "the application or framework enforces a strong anti-CSRF mechanism to protect authenticated functionality, and effective anti-automation or anti-CSRF protects unauthenticated functionality". First, some background. OWASP describes CSRF as a "attack that forces the end-user to execute unwanted actions on a web application in which they're currently authenticated" [18]. When the browser requests a resource from a website, it creates a cookie that stores the user's session information. When the user sends a request to the server, the browser will automatically include the cookie for authentication. An example attack scenario is one where the victim is logged in to a bank, and the attacker exploits this by sending a hyperlink in an email with a request for a transfer of money to the attacker's bank account. Because the cookie is included in the request, the transaction goes through.

One solution is to set the **SameSite** flag in the cookie. To summarize the fix, setting this flag ensures that the cookie is ignored for POST based CSRF attacks, meaning, the attacker's hyperlinks will be ignored [33]. This specific CSRF-fix is addressed in control 3.4.3, with half the participants checking it as an implementor responsibility also. Setting the SameSite attribute, according to the implementor, is a controversial topic amongst the DHIS2 team. Presently, strong anti-CSRF mechanisms are not implemented, mainly due to legacy reasons and how the API is used, and the SameSite attribute is therefore set by the implementors (with guidance provided by HISP).

This issue is elaborated on and its implications discussed in section 6.3.2.

5.1.7 ASVS V8: Data Protection

There are three key elements to data protection: confidentiality, integrity and availability (CIA). This ASVS chapter addresses the CIA, seeking to ensure that data is protected from unauthorized disclosures, that data is protected from unauthorized altering or deletion, and that data is available

to authorized users.

Controls	Applicable	Pass	Fail
17	13	7	4

Table 5.8: Results from ASVS section V8

Applicability & scoring

In this chapter, 13 out of 17 controls were selected as applicable, with 7 that pass and 4 that fail. Here two prominent examples will be showcased. The first is 8.3.4, an example of an implementation-only control. The control's description states: "Verify that all sensitive data created and processed by the application has been identified, and ensure that a policy is in place on how to deal with the sensitive data". As the frontend developer writes, policy is set by the district, ministry, country et cetera. In other words, while DHIS2 can allow for the classification of data, how the data is dealt with is entirely up to the implementors. The same is true for controls 8.1.1 and 8.2.2 also.

Control 8.3.2 is the second example. The control description states: "Verify that users have a method to remove or export their data on demand". To reiterate, DHIS2 stores two types of data: aggregated and patient data. Though the control is applicable, the frontend developer writes that there is a difference between exporting aggregated data and patient data, though does not elaborate further. During the scoring phase, the same participant gave the control a fail. The backend developer, on the other hand, gave it a pass, explaining that the user can export all their data on demand. Whether or not the participant considered the "aggregated" angle is unclear.

5.1.8 ASVS V10: Malicious Code

"Finding malicious code is proof of the negative, which is impossible to completely validate" [9, p. 50]. This ASVS chapter tackles many attacker-

centric requirements, such as handling malicious activity.

Controls	Applicable	Pass	Fail
10	10	7	3

Table 5.9: Results from ASVS section V10

Applicability & scoring

As seen in table 5.9, all 10 controls are applicable, 7 of which pass and 3 fail. One control of interest here is 10.3.1, which states: "Verify that if the application has a client or server auto-update feature, updates should be obtained over secure channels and digitally signed". During the applicability phase, the implementor said that when the DHIS2 file is downloaded, there is no hash code present to verify the validity of the package. However, it is unclear if this is true for updates as well. During the scoring phase - which the implementor did not participate in, the backend developer scored the control as a pass. This raises the issue of whether the control was properly scored. Security requirements unaddressed can result in severe consequences if the vulnerability is left unpatched and then exploited by an attacker.

5.1.9 ASVS V12: File and Resources

This ASVS chapter seeks to ensure that untrusted files are handled accordingly and securely, such as stored outside the web root and with limited permissions.

Controls	Applicable	Pass	Fail
15	15	10	5

Table 5.10: Results from ASVS section V12

Applicability & scoring

As seen in table 5.10, all controls are applicable, with 10 that pass and 5 that fail. Control 12.3.6 is an example of conflicting point of views. The control description states: “Verify that the application does not include and execute functionality from untrusted sources, such as unverified content distribution networks, JavaScript libraries, node npm libraries, or server-side DLLs”. The frontend developer checked it as fail, though also noting that they attempt to code review libraries they rely on. This can be a difficult and time consuming task, and developers must consider risk versus probability. For these reasons, both the backend developer and security engineer on the other hand gave it a pass.

5.1.10 ASVS V14: Configuration

The aim of this chapter, per ASVS’ description, is to create “secure, repeatable, automatable building environment” and “secure-by-default configuration, such that administrators and users have to weaken the default security posture” [9, p. 60].

Controls	Applicable	Pass	Fail
25	23	16	4

Table 5.11: Results from ASVS section V14

Applicability

Out of 25 controls in this category, 23 are applicable. In this ASVS chapter, only one control applies both to software and implementation. Control 14.5.3 asks to “verify that the cross-domain resource sharing (CORS) Access-Control-Allow-Origin header uses a strict whitelist of trusted domains to match against and does not support the "null" origin”. Both the frontend- and backend developer noted that the CORS whitelist

is controlled by the implementors or system administrator. This control is applicable to software also because the main functionality, and the check against the "null" origin, needs to be build by the core development team.

Scoring

In this phase, 16 controls are a pass, 4 fail, and a further 3 are unresolved. One control of interest is applicable both to software and implementation. This control, 14.3.2, concerns the use of a debug mode, used during development to display relevant information during the software's execution. If left on during production, it can reveal sensitive information, such as developer secrets and security disclosures, e.g. whatever the developer puts there. While the control was evaluated to a pass, the implementor expressed concern during the applicability phase, given that the implementors can turn on the debug operation in their local implementations. Though not activated by default, the implementors still need to be given guidance on what to do and not to do.

5.1.11 Updated scoring

In early April 2020, the security engineer asked to give a status update on the failed controls. Since our last conversation in early December, many vulnerabilities had been fixed or are in the process of being addressed. This section will present these results. However, the original audit results still stand and will be the focus of the coming discussions. The purpose of presenting the updated results here is to show the improvement that can be made in a relatively small amount of time. The full spreadsheet for this result can be seen in Appendix B, section B.1.

Table 5.12 summarizes the new results. The number of controls that passed went from 174 to 189, and the number of controls that failed went from 50 down to 35. Furthermore, an additional 8 controls are in the process of being fixed and should be available in the next DHIS2 release. This also

	Applicable	Pass	Fail	Conflict	Uknown
Old	258	174	50	5	29
New	258	189	35	5	29

Table 5.12: Updated results from the scoring phase

means that for level 1 of ASVS, the number of failed controls is now 14, down from 19. For the next release, the number will be 11.

An example of a planned functionality is control 2.1.7. The control deals with checking a user-entered password against a list of the most common passwords. This check can be against a list of one thousand or ten thousand of the most common passwords. This control is presently a fail, but the functionality should be coming soon.

5.2 Part 2: Ranking results

In this phase, the severity and importance of the failed controls was ranked. However, only a subset was ranked, 39 out of 50, due to there originally being a greater number of controls with conflicting results. These were later resolved by the security engineer, many as pass, most as fail, but this occurred after this phase had been completed.

First, the results are presented statistically, giving an overall overview of the results from each participant. Then, examples are hand-picked to illustrate instances of agreements and disagreements, and the implications of this.

5.2.1 Statistical results

Each ranked control was given a score from 1 to 5. Tables 5.13 and 5.14 show the security engineer and front-end developer's distribution of controls across these five rankings. As seen in table 5.13, the security engineer gave the majority of controls a rank 4. The frontend developer, on

the other hand, ranked the majority of issues as rank 3.

Rank 1	Rank 2	Rank 3	Rank 4	Rank 5
2	2	7	19	5

Table 5.13: Phase 3 results from the security engineer

Both participants placed the majority of controls in rank 3 and above. How each control is ranked is dependent on the participants' view and knowledge of the issue. As the next section illustrates, the results for several controls varied significantly.

Rank 1	Rank 2	Rank 3	Rank 4	Rank 5
2	3	16	7	7

Table 5.14: Phase 3 results from the frontend developer

5.2.2 Critical and non-critical controls

The key takeaway from the previous section is that there were a lot of differences in the rankings given by the participants. This section highlights three sets of examples: critical controls in which both participants agreed, non-critical controls in which both participants agree, and controls with radically different rankings.

Critical controls

Two instances where both participants gave a control a rank 5 was with controls 1.1.1 and 1.2.3, both in the Architecture chapter of ASVS. The first example is on the use of a secure software development life cycle, such that "security is addressed in all stages of development". The frontend developer looked at the entirety of the process, writing that he considers "security a process more than an end state". The security engineer focused on one particular activity in software development: code commitment.

Like many other open-source projects, the DHIS2 codebase is located in a (public available) Github repository. When a developer is ready to integrate their changes with the central repository, they commit their changes and can then choose to either push the changes directly or, create a pull request in which their work is peer-reviewed. The concern expressed by the engineer is an attacker gaining access to an account and committing malicious code to the DHIS2 repo. However, this is a general concern with all software that utilize such a service. In an email correspondence, the security engineer provided more details. No commit is merged into master without approval from at least two reviewers. Furthermore, many developers sign their commits with a digital signature, thus verifying the validity of the source. According to Github's help page, it is possible to set a "branch" as a protected resource and block all non-verified commits. It is unclear to me if that is the case with DHIS2.

The other control, 1.2.3, is described in ASVS as such: "Verify that the application uses a single vetted authentication mechanism that is known to be secure, can be extended to include strong authentication and has sufficient logging and monitoring to detect account abuse or breaches". The frontend developer stressed the severity of this vulnerability, writing that "it is only a matter of time until the first breach happens as a result of this [vulnerability]". The security engineer expressed that the control is broad in its description, and could be broken into smaller pieces. The basics of logging and monitoring, though not perfect, are in place. For example, the user is unable to do an unlimited number of password recoveries, thus limiting "account abuse".

With the April 2020 update, both controls are scored as pass by the security engineer. For 1.1.1, the security engineer expressed that for an open-source system, the development cycle is good enough, and on 2.2.3, a lot of improvements had been done in the last six months: DHIS2 now uses an industry-standard authentication system, like password algorithms and 2-factor authentication, and there have been improvements in logging and

monitoring as well.

Non-critical

Two instances where both participants gave a control a rank 1/2 are with controls 2.1.8 and 2.1.12. On the first, the control deals with the use of a password strength meter when setting or updating a password. Both participants expressed that while this feature is helpful for the user, it does not necessarily improve security. The second control also concerns the user experience, allowing the user to "temporarily view the entire masked password, or temporarily view the last typed character of the password on platforms that do not have this as native functionality". Both again agreed that this is a user interface concern, and not of the highest priority from a security standpoint.

Conflicting results

Two instances where the two participants gave vastly different rankings to controls occur with 4.3.1 and 12.4.2. The first control aims to test that "administrative interfaces use appropriate multi-factor authentication to prevent unauthorized use". This control was given a rank 2 by the security engineer, arguing that while two-factor is enabled, it is not enforced, because 2FA is more of a UI decision, rather than a technical one. The same control was given a rank 5 by the frontend developer, who remarked that DHIS2 is particularly vulnerable to it if the aforementioned CSRF vulnerability is not corrected (also ranked 5 by the participant).

The second control deals with ensuring that files obtained from untrusted sources are scanned and blocked if malicious. DHIS2 can be extended with additional applications developed by third-parties, and this is where the security engineer chose to focus on. Discovering a malicious application is difficult unless strict guidelines and measures are put in place. Apple, for instance, reviews every application before it is allowed on the App

Store [38]. For DHIS2, this is significantly harder to do. The frontend developer took a different approach, emphasizing that the implementors are themselves in charge of ensuring that no malicious application is uploaded onto their DHIS2 instance. The control was therefore given a rank 1.

Chapter 6

Discussion and analysis

In this chapter, I will first give a summary of the results from the audit, and then discuss their implications. This discussion is accomplished by comparing the results against what is expected from an HMIS, presented in section 6.2. Then I turn to the second objective of this thesis, to analyze the auditing process and the results to answer this research question:

> *What are the benefits and challenges of using ASVS to assess DHIS2's security?*

First, section 6.3 leverages the results from the audit by giving possible suggestions for addressing the discovered shortcomings. Then, by combining the results with the supporting literature, this thesis can broadly explore both the relevance of an ASVS audit and standards in general. This topic is presented in section 6.4. This is followed by a reflection on the challenges and limitations of the project in section 6.5. Finally, section 6.6 reflects on the conducted research.

6.1 Summary of the results

Chapter 5 presented the results from the audit. 258 out of 286 ASVS controls were applicable to DHIS2, either in part or semi-applicable to

implementation also. 178 out of those 258 were assessed as pass, whereas 50 controls failed (as seen in table 6.1). The main focus of this chapter will be on the failed controls, discussing what they tell us about DHIS2's security and how to address these shortcomings.

	Total	Level 1	Level 2	Level 3
Failed	50	19	27	4
Semi-implementation	12	1	10	1

Table 6.1: Failed ASVS controls

Note that while 12 of the 50 controls that failed also concern implementation, they do not necessarily fail on the implementation side – or vise-versa. This distinction in the results is difficult to make due to insufficient data - with the exception of a few controls that were commented on by the participant. Nonetheless, for the purpose of the discussion, these "exception" controls are sufficient to be able to draw examples from.

6.1.1 Factors in applicability

The non-applicability of ASVS controls was due to two reasons. The first are requirements that target services and components DHIS2 does not have. And the second leading factor was requirements applicable to implementation, either in part or wholly. Within the implementation category, four unique factors were identified.

- **Implementation due to local laws**. DHIS2 is used in many countries and contexts. Therefore, controls that deal with configuring DHIS2 to suit local laws and regulations to satisfy the different requirements on different districts are implementor responsibility.
- **Implementation due to configuration**. Several security features have to be configured and activated by the implementors.
- **Implementation due to sensitive data classification**. The developers do not know what is sensitive in each context DHIS2 is used in.

The core team can provide the functionality to classify data, but the ultimate responsibility to classify data correctly falls on the implementors.

- **Implementation due to web-server setup**. Several controls concern the web-server and the application of firewalls, proxies and the like to secure the hosted platform and the data server. This also includes logging and analyzing the data.

The coming sections will discuss how to address some of these categories.

6.2 DHIS2 and the expectations of security in an HMIS

High-tier security for an HMIS is critical, as established in section 3.3.1: Security in an HMIS. The results established a baseline, which we can utilize to evaluate the strength of DHIS2's security as compared against what is expected of general data security and the health sector. This section examines this, first by looking at the results compared to what ASVS says and then seeing what inspiration can be taken from the GDPR in building secure applications.

6.2.1 Auditing results compared to ASVS levels

OWASP describes a level 1 coverage as "bare minimum that all applications should strive for" [9, p. 10]. For software that handles health information and personal data, a level 2 coverage is recommended, where "failure could significantly impact the organization's operations, and even its survivability" [9, p. 11]. A level 3 is recommended for applications whose failure could jeopardize operations or even human life. DHIS2 falls into the level 2/3 category. As seen in table 6.1, DHIS2 falls short of a level 1 coverage with 19 failed controls, and 27 and 4 failed controls on levels 2 and 3 respectively.

However, not all security controls are equally important, and it can be argued that organizations are unlikely to view the results in such black and white colors. The purpose of adding a third phase to the assessment process: Ranking, was to register a risk assessment process of a sort, meant to give insight to how the participants ranked the importance and severity of each control. As seen in section 5.2, the ranking given to each control varied. For instance, a small subset of the controls from level 1 concern the frontend interface, as in the case of two controls: 2.1.8, on the use of password strength meters, ranked as a level 1 issue by both participants; and 2.2.3 on allowing the user to temporarily view the masked password (ranked as a level 2 issue). In a few instances, one participant gave a control a rank 5, another a rank 1.

With that said, each failed control is an opportunity for the attacker to penetrate the system and steal or corrupt the data. The majority of the issues in the ASVS failed list were ranked as level 3 or above. Vulnerabilities such as the CSRF attack were given a level 5 ranking, issues the development team have been aware of and are highly vulnerable against, but have yet to fix.

6.2.2 GDPR as a reference point for DHIS2

At the time when this audit was conducted, DHIS2 had not been used in Norway before. This changed with the outbreak of COVID-19. As of May 2020, DHIS2 now is being implemented by municipalities in Norway to track contact tracing between individuals, and DHIS2 is therefore now directly subject to GDPR. How HISP and the DHIS2 development team indents to handle this is immediately unknown to me.

A single control in ASVS makes a direct reference to GDPR: “Verify that regulated private data is stored encrypted while at rest, such as personally identifiable information (PII), sensitive personal information, or data assessed likely to be subject to EU’s GDPR” (6.1.1). Outside the

EU, DHIS2 is not subject to GDPR. Even before the recent development in Norway, the control was still selected as applicable, likely due to the GDPR's principles and guidelines on handling private data securely. In the international context, we do not need to be GDPR compliant, and specific articles of GDPR can only be accomplished at the implementation level, but developers can still take inspiration from its contents, specifically Article 25 on *data privacy by design* and *data privacy by default* (outlined in section 3.5.2) [10], the purpose of which is to protect the personal data of its citizens. As outlined in section 3.3.2, several African countries are taking inspiration from and implementing a similar set of laws as those found in GDPR. By using the GDPR as a point of reference, the DHIS2 development team can help ensure that the core software not only takes personal data protection to heart but is configurable enough to meet the different contexts it is used in.

6.3 Possible solutions to software and implementation

The separation between software and implementation is an important distinction to consider with DHIS2. Due to this, software security must be approached from two different angles. This section seeks to leverage the benefits of an ASVS audit and give context to the results, exploring possible solutions to addressing the discovered vulnerabilities, both from a software angle and an implementation angle. To this end, this discussion will take a broad approach, with solutions that address the issues at the root, rather than for each control.

6.3.1 Solving software related controls

Failed controls that are applicable to software are the responsibility of the DHIS2 development team. There are multiple, varied approaches to

addressing them, and the selection(s) depends on the available manpower and the team's willingness to invest in sophisticated processes. A relatively simple, straightforward approach is to raise the security requirements as tasks and fix the vulnerabilities. However, this is still reactive, rather than proactive. In the past, "it was common practice to perform security-related activities only as part of testing" [43]. This meant that security vulnerabilities were discovered either too late or not at all. The first control of ASVS recommends the use of a "secure software development lifecycle that addresses security in all stages of development". The idea is that by making security a continuous activity, you make security proactive, rather than reactive.

This topic brings up the **Secure Software Development Lifecycle** approach presented in section 3.1.1. Here the discussion turns to embedding security into the **Agile** development method (currently followed by the DHIS2 team), but other development models, such as DevOps [21], are also viable.

Implementing security activities into the SDLC can be categorized into these three levels [42].

- **Every Sprint (Most critical)**: These are the security activities that should be repeated in every sprint, such as code analysis, following secure coding checklists (relevant to control 1.1.7), and threat modeling (relevant to control 1.1.2).
- **One-time (non-repeating)**: These activities are performed at the start of a new project, like risk assessment and analyzing attack surfaces.
- **Bucket (all other)**: These activities are performed regularly, spread across multiple Sprints, such as random testing, attack surface review, and dynamic analysis.

A benefit of ASVS is that the requirements are highly testable, and the controls from the list can be incorporated into several of theses phases, either during every sprint, every couple of months, or once a year. This task of embedding security into the SDLC and approaching security with

a *Security by Design* mindset might be challenging. In fact, "security" and "agile-development" have often been referred to as incompatible, due to the short sprints and iterative deliveries [11]. However, the Agile method is known to be highly flexible and adaptable [42]. A proposed solution is to raise issues discovered in the audit as tasks in the product backlog. This, the OWASP team notes, "helps with prioritization of specific tasks (or grooming), and makes security visible in the agile process" [9, p. 13].

Security by Design and Default, Data Protection by Design and Default, all share similarities. All four need not be followed, and the latter two are only applicable to GDPR-compliant countries. Nonetheless, their core principles, which stress the importance of having security be a founding and continuous activity in a software's life cycle, can be useful to any organization seeking to become better at security.

6.3.2 Solving implementation related controls

Addressing software on the implementation side comes with a different set of challenges. First, there is the initial security configuration of a local DHIS2 instance, which if improperly set up will leave the platform vulnerable. Secondly, security-related activities must continue throughout the platform's lifecycle, in activities such as monitoring and updating to newer versions of DHIS2. Security at the implementation level is an ongoing issue, as laid out in section 2.2.6, with poor proxy setup, no firewalls, and often no monitoring, to name a few. Presently, local HISP groups provide guidance to address these shortcomings. This section will reaffirm the importance of the guidance and training that HISP provides. Also, it will propose *security by default* - addressing implementation through software - as a development strategy to ensure that DHIS2 is as secure as possible by default.

Note that during the scoring phase, only controls applicable to software and semi-applicable to implementation were scored. Non-applicable

controls that concerned implementation-only were not scored. The majority of participants were from the development team, and the intention was to keep the focus on controls that concerned them the most. The solutions proposed in this section, however, could, in theory, be applied to the implementation-only controls also.

Addressing implementation through guidance

According to Sæbø, Adu-Gyamfi and Nielsen, the extent of the expertise that HISP provides with supporting implementation includes the "training of users at different levels, system implementation, maintenance, integration with other systems and software development of extensions and apps" [6, p. 75]. This section will attempt to give context to this in light of the results, by selecting examples that could be addressed through guidance.

The first example is control 1.1.5, which asks to "verify definition and security analysis of the application's high-level architecture and all connected remote services". According to the frontend developer, this is an implementation issue, but that HISP should "provide clear guidance on what the sysadmins MUST and MUST NOT do in terms of maintaining a secure system". A similar sentiment is true with 8.1.4: "Verify the application can detect and alert on abnormal numbers of requests, such as by IP, user, total per hour or day, or whatever makes sense application". This control is handled at the server level, and it is the responsibility of the system administrators to monitor the API traffic, not DHIS2 itself.

During the audit, the participants noted several examples where a bad configuration could leave DHIS2 vulnerable. For instance, one control 1.12.1, which asks to verify that "user-uploaded files are stored outside of the web root", the release manager explained that this feature is configured by the implementors. The consequence of this is a system administrator that - unaware of the repercussions - alters the configuration to store files inside the webroot itself. So, for example, if a user or the attacker then

uploads a malicious file and that file is stored inside the webroot, the system could be compromised.

The danger is arguably higher with the next example: control 2.5.4, a requirement the implementor considers to be an ongoing issue. When an instance of the DHIS2 platform is initialized, there is a default "admin" account present. Alternatively, this can be referred as the "root". This account has higher privileges and is used to create other accounts. After the bootstrap is completed, the admin should be deleted, but often is not. It is unclear to me the full scope of what the DHIS2 admin accounts have access to, in terms of both read and write. In general, the presence of an admin account is dangerous. OWASP writes that, for a threat agent to accomplish their goal, they "have to gain access to only a few accounts, or just one admin account to compromise the system" [2]. Further, it is essential that software is not shipped or deployed with default credentials, such as admin/admin for username and password. It is unknown to me if that is the case with DHIS2. If it is, then the admin user must either be disabled, or the password has to be changed after the bootstrap and kept confidential.

DHIS2 supports multi-factor authentication to prevent unauthorized use, the topic of control 4.3.1, but, according to the release manager, it is configurable and is not enforced. There are multiple cases like these to find in ASVS with DHIS2. In these instances, the users and administrators themselves have to consider usability versus security and decide if it is worth implementing. These examples underline the importance of good on-site training and guidance. In a paper on digital health intelligence and security in HMIS's, the writers state that "there is an increasing concern that cloud server in general, and Amazon's in particular, have vulnerabilities poorly understood by regulators and users who rely on their security" [12, p. 16]. As the reach of the internet and DHIS2 expands, particularly in Africa, these vulnerabilities and many more discovered in the ASVS audit could be prime targets for an exploitation.

Addressing implementation through software

The concept of **Secure by Default** is important to this discussion, and was originally brought to my attention as a potential remedy by the implementor during a presentation of DHIS2's security landscape. As outlined in section 3.1.2, the philosophy of this approach is to build software that is secure by default. Of particular interest to DHIS2, in the context of implementation, is the following principle: "security should not require extensive configuration to work, and should just work reliably where implemented" [56]. A few 'by default' examples is setting the firewall by default, requiring authentication by default, enabling encryption by default and setting password complexity by default. By configuring the software to be secure out-of-the-box, we lessen the burden on the implementors. Security configuration is the sixth entry in OWASP's Top 10 list, addressing issues like "insecure default configurations, incomplete or ad hoc configurations, open cloud storage, misconfigured HTTP headers, or verbose error message containing sensitive information" [48]. Chapter 14 of ASVS is also dedicated to configuration.

An example of this configuration concern is the failed control 4.2.2 on **Cross-Site Request Forgery**, as explained at length in section 5.1.6, an issue that has been previously raised by external security groups. In short, the issue arises when an attacker uses the authenticated state of a user to commit fraudulent transactions. The solution is also addressed in control 3.4.3, whereby setting a SameSite attribute in the cookie prevents this form of attack [33]. As discussed previously, this control is a point of argument issue among the HISP developers and implementors. Setting in on by default would ensure that the instance of DHIS2 given to implementors is as secure as possible without needing to configure it. Presently, it is not set, the reason being that developers make apps which they test against online test/demo servers, and if the sameSite attribute was enabled, testing would be difficult. In an email exchange sometime after the assessment had been

completed, the implementor reaffirmed his stance on the issue, arguing that "better that devs have to do a bit of work to workaround its inclusion, than implementers have to do workaround its exclusion".

There is possibly a balance to find here, between what is best for the implementors and what is best for the developers. And if that balance is struck - even if all the details of the control are not accomplished, given the intricacies of addressing it, it could be safe to check the control as passed.

6.4 Relevance of an ASVS assessment

The audit needs to provide benefit to the organization. This section argues for those benefits and discusses ASVS' relevance, comparing it towards other methods of establishing security and looks at how developers can best utilize the standard.

6.4.1 OWASP's Top 10 as security standard

The ASVS is first and foremost not an awareness project. Its purpose is to provide a more proactive approach to application security, offering detailed requirements suited to organizations of different sizes. This is in stark contrast to OWASP's Top 10, which was specifically designed to be an awareness document, representing a "broad consensus about the most critical security risks to web applications" [48]. The authors of ASVS tailored the level 1 requirements of ASVS to reflect the Top 10 list, thus "making it easier for OWASP Top 10 adopters to step up to an actual security standard" [9, p. 10]. The question this section wants to answer is whether adapting the ASVS is worth the time and effort.

One important factor to answering this question is raised by OWASP themselves. The first is that the Top 10 is only reflected in the first level of ASVS, the bare minimum for entry-level awareness. For a smaller team new to security, the Top 10 or level 1 of ASVS is a good starting point.

DHIS2, on the other hand, needs to look beyond the simplicity of the Top 10. The Top 10 provides sufficient background on a particular issue, including possible mitigations and example attack scenarios. Leveraging and adopting the Top 10 remains good practice, but it does not help developers create a secure application, nor is addressing the current Top 10 vulnerabilities enough to cover all of the todays biggest vulnerabilities.

The current version of the Top 10, updated last in 2017, saw the removal of Cross-Site Request Forgery from the list, due to more applications and frameworks having anti-CSRF protection than when the entry was originally introduced [49]. As been previously discussed, CSRF is an ongoing issue with DHIS2. The same is true for other issues uncovered in the audit, some previously known to the DHIS2 developers, others not. Looking to the Top 10 is a good strategy, but arguably insufficient for DHIS2.

6.4.2 Relevance of ASVS in a mature organization

When no dedicated security team or role is present within the organization or team, the responsibility for driving or enforcing security related features and activities falls on the developers or team lead. In this case, these people must have or must acquire the necessary expertise to make the correct security decisions for their software. ASVS can be the guiding star that educates developers on the industry's best practices. However, the question this section wants to address is whether an ASVS audit is of worth to an arguably mature organization such as DHIS2, with a multitude of security features already in place, and whether it can provide value to developers who are already decently knowledgable in security.

Until the recent hiring of a security engineer, the DHIS2 team had no dedicated individual driving security within the organization. Therefore, security-related features received lower prioritization. Only in the closing months of 2019 had security become a focus. The auditing process

established that a significant number of controls fail. However, the qualitative data also showed that the participants collectively had a good overview of the many security vulnerabilities that are present in DHIS2, particularly within their areas of expertise, be it backend, frontend or implementation-related vulnerabilities.

It is difficult to say if the audit provided any benefit to the participants, and if it drew their attention to any particular vulnerability they were either not aware of or had dismissed as less-than critical. This is particularly difficult to evaluate from the author's perspective, as this form of data was not part of the official data gathering, and would require additional interviews. In an email correspondence on March 2020, however, the security engineer re-scored the previously failed controls, and exclaimed the learning experience – having recently been hired – of seeing how DHIS2 compares against ASVS. Since our first chat in early December 2019, the number of failed controls had been reduced significantly. While I am unable to draw a direct causality between my project and the improvements, I would argue that the new results highlight the relevance of ASVS's controls to DHIS2.

In general to all software teams, ASVS's relevance is dependent on the team's or individual's maturity level on the subject of security. Rather than do a systematic audit using ASVS, a developer is perhaps more likely to seek out individual solutions to the problems they are facing or have been reported in the field, or implement a security feature in response to a vulnerability they read in an article or paper. During a DHIS2 security seminar in April 2019, the implementor expressed that vulnerability management was done on an ad hoc (less than ideal) basis, and to address DHIS2's security vulnerabilities, there must be more "external" pressure from management and users. Using ASVS to inform the organization of its vulnerabilities as deemed important by the many experts involved in the making of ASVS is one approach to making the vulnerabilities more visible.

6.4.3 Time estimates for the audit

The previous section reflected on the need for ASVS in an organization with a mature security infrastructure already in place. This section builds on this by giving time estimates for the audit as conducted in this project. A significant factor in the willingness to invest in a standard is the time it takes to complete the exercise and is therefore important to explore. In general, there are three main time factors to consider:

- **ASVS level**. Doing an audit against level 1 takes significantly less time than doing all three, as the control complexity increases with each level, thus adding to the time it takes to possibly research and confirm a control's status.
- **Method of assessment**. The approach to the data gathering was split into two phases. This was done out of necessity, due to the challenges of finding participants. Potentially, these two phases would occur concurrently, which could drive down the time significantly.
- **Thoroughness of the assessment**. The participants scored each control on a best-guess basis. This is not necessarily ideal, depending on the risk level, as controls can be scored incorrectly if not properly researched and/or tested and verified.

These three factors impact the potential time estimates for the audit. This is a student project and is therefore not necessarily a reflection of an audit as it would have occurred had this been done by a member of HISP. Nonetheless, the following is a time estimate for the entire project.

- **Applicability phase**: Four participants were interviewed, each session taking approximately two hours each, for a total of eight hours.
- **Scoring phase**: Two participants were initially interviewed, for approximately two hours each. Adding up the contribution by the security engineer, this brings up the tally to four and a half hours.

To summarize, the auditing process (excluding the ranking phase) took 12,5 hours. This is excluding the organization of responses and analysis that occurred thereafter.

6.5 Challenges and limitations to the audit

Among the other issues listed throughout this thesis, the main challenges and limitations of the auditing process are examined below.

6.5.1 Developer participation and consulting with multiple people

The leading issue throughout the process was finding and engaging with the participants. The process could not be realized without the contribution of HISP employees and especially the DHIS2 development team, who have intimate, technical knowledge of the platform. There was an expectation on the author's part that there would be a larger engagement from the group. Several leading members of the DHIS2 team were contacted by email but never responded. The number of participants for each phase also dwindled, who where either too preoccupied or unresponsive when contacted by email. The consequence of this is that the results are inconclusive. 29 controls remain unresolved, and 5 have conflicting opinions. It is also entirely possible that some controls were scored incorrectly. Instances where one participant skipped on a control, and another gave it a pass or fail, received the latter's score, meaning their evaluation of said control is unchallenged and is possibly incorrect.

Luckily, the group that participated in this exercise all had vastly different jobs within HISP. This is important because a developer might have one perspective on an issue, a release manager, or implementor another. Had all participated in all phases - in the first two phases in particular - the final set of data would in all likelihood be different. The key takeaway from

this experience is that when doing an audit it is important to get people with different areas of expertise. DHIS2 consists of a lot of moving pieces: multiple modules, apps, API, and a frontend and a backend, and a single individual is unlikely to have a comprehensive overview over it all. An example: during the scoring phase, the frontend- and backend developer scored 122 and 146 controls as pass respectively. However, this left 104 and 69 controls unresolved for each. By combining their results, the number was brought down to 29, thus filling in the areas left blank by the other.

Finding and engaging with the relevant people is likely to be easier if the project is instigated by people within the company, be it an internal or external audit. In both cases, the lead in this endeavor is likely to have access to all documentation and better access to the developers (if needed).

6.5.2 Outdated data

The security engineer re-scored the previously failed controls, and the number of failed controls went from 50 to 34, with several more fixes planned for future releases. This new data shows that the results of this thesis were built on is now outdated. And this was to be expected. The last scoring sessions occurred approximately half a year before the planned publication of this paper. These new results reflect HISP's increased focus on DHIS2's security, which had never been of particular focus within the organization until now. This thesis is then an interesting time capsule that lets the reader see how DHIS2 was half a year ago, and the improvements that can be made within a relatively short amount of time.

6.6 Reflections upon the research conducted

6.6.1 Errors in the applicability spreadsheets

The initial spreadsheet downloaded from OWASP's ASVS site contained no checkboxes. These were added to register the applicability of a control to

software and to implementation. However, the single **Applicable** checkbox could only register if a control was applicable or not applicable to software; it could not register if the participant was unsure and skipped the control. This error was not discovered until late in the process. Where possible, the participants or the interviewer would note this in the comments field, but this was not always done due to negligence. The implication of this is that the applicability of some control might have been judged erroneously. However, if this happened, the errors would be far in between. The majority of controls are applicable, and for the most part, the non-applicable controls all have comments. The same mistake occurred in the second phase, but was discovered immediately during the first session and corrected by adding an **Uknown** column of checkboxes.

6.6.2 The author's involvement in selecting applicability

Chapter 4 presented a list of criteria for how the results from the participants would be merged. For example, during the applicability phase, a control is checked in the final list if a majority of the participants have checked it. If, however, an argument is provided arguing for the inclusion or exclusion of a control, that argument is evaluated and taken into consideration. This can call into question the validity of the audit results and the influence the author had over them. I will present two counter-arguments against this.

The first is that the author's influence over the results was inevitable. The data had to be combined for the next phase, and lest a member of HISP aided in this task, someone had to make these decisions. It is possible that controls were erroneously selected as applicable, and vise-versa. But this leads us to my next point. On the majority of the controls, in the case of uncertainty, a control was checked as applicable. Better a control was wrongly included, rather than excluded. The consequence of possibly including irrelevant controls is that it adds to the total number of applicable controls, which in turn lowers the percentage of passed controls given that

it is calculated from the total.

Chapter 7

Conclusion

By conducting an audit of DHIS2 using the Application Security Verification Standard, this thesis has explored the process, the results, and the merits of this exercise. The importance of a security standard has been much written about by security experts. Yet, to my knowledge, there has not been a study done that has explored the benefits and challenges to an ASVS assessment through first-hand experience.

The research project was based on an interview method as a tool for data collection. Three phases of data collection were utilized in this thesis, wherein members of HISP and the DHIS2 development team were interviewed. The first two phases concerned the audit, in which the participants would select security controls applicable to DHIS2 in the first phase and then score them in the second. The third phase was created to evaluate the relevance of ASVS's controls to DHIS2. For each phase, the data was collected in Excel spreadsheets and then analyzed for patterns and themes.

7.1 Research summary

This thesis aimed to answer the following research questions:

How does DHIS2's security compare against ASVS?

What are the benefits and challenges of using ASVS to assess DHIS2's security?

Both of these questions were answered by conducting a security audit of DHIS2. The first of these were answered by the quantitative data that was gathered by scoring the ASVS controls. The data revealed that DHIS2 fell short of what ASVS and the general literature recommends for security in a health information system. In the third phase, it was discovered that several of the controls that failed were critical. The second question was answered by analyzing all three sets of data and reflecting on the process of doing this exercize and the relevance of the ASVS standard to DHIS2. The results are key challenges and benefits which are as follows:

- Relevance of ASVS controls
- Leveraging the results to strengthen security
- The benefit of ASVS over OWASP's Top 10
- Relevance of ASVS in a mature organization
- Challenges to the ASVS assessment

The **relevance of ASVS controls** to DHIS2 is clear. The majority of controls applied to the platform. Many controls concern issues that the team is currently dealing with, and many have subsequently been fixed. In the context of DHIS2, implementation was the biggest decider of applicability. Even then, the relevance of these controls is not diminished, as HISP and other organizations strive to guide local governments and health facilities in the secure implementation of the DHIS2 platform.

An ASVS audit can be best utilized to align your security solutions with an internationally-recognized standard and **strengthen security**. The audit revealed vulnerabilities. In DHIS2, these vulnerabilities can be addressed from two different angles. For controls applicable to software, this thesis proposed the adoption of a Secure Software Development Life Cycle

(SDLC) approach, such that security activities are made proactive, rather than reactive. For controls applicable to implementation, there are two approaches: (1) the continuing of the training and guidance provided by HISP, such that the local instances of DHIS2 set up by the implementors are secure; (2) and address implementation at the software level, embracing the concept of Secure by Default to build software that is inherently secure and that does not require extensive configuration to work reliably.

The **benefit of ASVS** over security awareness documents like OWASP's Top 10 and other similar publications is that it provides a more proactive approach to application security, offering clear, detailed guidelines on how to best create a secure web application. For a smaller team and software of lesser significance, the Top 10 is a good starting point, but for DHIS2 a more complex set of guidelines is required.

The relevance of ASVS in a **mature organization** with a comprehensive security infrastructure was discussed. Collectively the participants were very knowledgable about the strengths and weaknesses present in DHIS2. If the exercise had any impact on the participants, and if it revealed anything new and of worth to them, is difficult to determine – with one exception. The exercise allowed the security engineer to see how DHIS2 compares against ASVS, and to get a birds-eye view of the platform's current security solutions. The relevance of ASVS is ultimately dependent on the individual's existing knowledge on the issues or the organization's willingness to embrace and invest in this process.

The assessment of DHIS2 came with a set of **challenges**. An audit can be done in many ways. Ideally, if one person does not have an overview of the entire system, multiple people with different areas of expertise should be consulted. Finding relevant participants proved to be a major challenge throughout the entire process and contributed to the inconclusiveness of the final results. Time is also a major factor to consider when investing in this process. The audit takes time. The time needed is dependent on the selected ASVS level, method of assessment, and the thoroughness of the

audit.

Bibliography

[1] *21.3. Exporting data and meta-data*. DHIS2. URL: https://docs.dhis2.org/2.22/en/user/html/ch21s03.html. [Accessed 15-April-2020].

[2] *A2-Broken Authentication*. OWASP. URL: https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A2-Broken_Authentication. [Accessed 15-April-2020].

[3] Ahmed Abdel-Aziz. *Scoping Security Assessments - A Project Management Approach*. May 2011. URL: https://www.sans.org/reading-room/whitepapers/leadership/scoping-security-assessments-project-management-approach-33673. SANS Institute.

[4] *About DHIS2*. DHIS2. URL: https://www.dhis2.org/about. [Accessed 15-April-2020].

[5] *About Microsoft SDLC*. Microsoft. URL: https://www.microsoft.com/en-us/securityengineering/sdl/about. [Accessed 15-April-2020].

[6] Eric Adu-Gyamfi, Petter Nielsen and Johan Ivar Sæbø. 'The Dynamics of a Global Health Information Systems Research and Implementation Project'. In: *Proceedings of the 17th Scandinavian Conference on Health Informatics*. Linköping Electronic Conference Proceedings 161. Linköping University Electronic Press, 2019, pp. 73–77. URL: https://www.researchgate.net/publication/337276393_The_Dynamics_of_a_Global_Health_Information_Systems_Research_and_Implementation_Project.

[7] Ali Alsaawi. 'A Critical Review of Qualitative Interviews'. In: *SSRN Electronic Journal* 3.4 (Jan. 2014), pp. 149–156. DOI: 10.2139/ssrn.2819536. URL: https://www.researchgate.net/publication/280155117_A_Critical_Review_of_Qualitative_Interviews.

[8] Adel Alshamrani1 and Abdullah Bahattab2. 'A Comparison Between Three SDLC Models Waterfall Model, Spiral Model, and Incremental/Iterative Model'. In: *IJCSI International Journal of Computer Science Issues* 12.1 (Jan. 2015), pp. 106–111. URL: https://www.ijcsi.org/papers/IJCSI-12-1-1-106-111.pdf.

[9] *Application Security Verification Standard 4.0*. OWASP. URL: https://github.com/OWASP/ASVS/blob/master/4.0/OWASP%20Application%20Security%20Verification%20Standard%204.0-en.pdf. [Accessed 15-April-2020].

[10] *Art. 25 GDPR Data protection by design and by default*. European Union. URL: https://gdpr-info.eu/art-25-gdpr/. [Accessed 15-April-2020].

[11] Charlie Belmer. *Integrating Security With Agile Development*. Apr. 2019. URL: https://nullsweep.com/integrating-security-with-agile-development. [Accessed 15-April-2020].

[12] Margaret Bourdeaux et al. *Weaponizing Digital Health Intelligence*. Jan. 2020. URL: https://www.belfercenter.org/publication/weaponizing-digital-health-intelligence#toc-9-0-0. [Accessed 15-April-2020].

[13] Jørn Braa and Sundeep Sahay. 'The DHIS2 open source software platform: evolution over time and space'. In: *Global Health Informatics. Principles of eHealth and mHealth to Improve Quality of Care*. The MIT Press, Apr. 2017. ISBN: 9780262338127. URL: https://www.researchgate.net/publication/316619278_The_DHIS2_Open_Source_Software_Platform_Evolution_Over_Time_and_Space.

[14] *CIS Controls*. Center for Internet Security (CIS). URL: https://www.cisecurity.org/blog/cis-controls-version-7-whats-old-whats-new/. [Accessed 15-April-2020].

[15] *Complete guide to GDPR compliance*. Gdpr.eu. URL: https://gdpr.eu. [Accessed 15-April-2020].

[16] *Coronavirus disease 2019*. Wikipedia. URL: https://en.wikipedia.org/wiki/Coronavirus_disease_2019. [Accessed 15-April-2020].

[17] Michael Coughlan. 'Interviewing in qualitative research'. In: *International Journal of Therapy and Rehabilitation* 16.6 (June 2009), pp. 309–314. DOI: 10.12968/ijtr.2009.16.6.42433. URL: https://www.researchgate.net/publication/261471599_Interviewing_in_qualitative_research.

[18] *Cross Site Request Forgery (CSRF)*. OWASP. URL: https://owasp.org/www-community/attacks/csrf. [Accessed 15-April-2020].

[19] *Cyber-security regulation*. Wikipedia. URL: https://en.wikipedia.org/wiki/Cyber-security_regulation. [Accessed 15-April-2020].

[20] *Definition of standard*. Webster. URL: https://www.merriam-webster.com/dictionary/standard. [Accessed 15-April-2020].

[21] *DevOps*. Wikipedia. URL: https://en.wikipedia.org/wiki/DevOps. [Accessed 15-April-2020].

[22] *DHIS2 Cloud hosting*. DHIS2. URL: https://www.dhis2.org/hosting. [Accessed 15-April-2020].

[23] *DHIS2 Jira*. DHIS2. URL: https://jira.dhis2.org/secure/Dashboard.jspa. [Accessed 27-April-2020].

[24] Eric Dosal. *4 Reasons to Run a Security Assessment*. June 2018. URL: https://www.compuquip.com/blog/4-reasons-to-run-a-security-assessment. [Accessed 15-April-2020].

[25] Jim Finkle and Jonathan Stempel. *Yahoo says all three billion accounts hacked in 2013 data theft*. Reuters. Oct. 2017. URL: https://www.reuters.com/article/us-yahoo-cyber/yahoo-says-all-three-billion-accounts-hacked-in-2013-data-theft-idUSKCN1C82O1. [Accessed 15-April-2020].

[26] *Framework and Standards for Country Health Information Systems Second Edition*. World Health Organization (WHO). 2012. URL: https://www.who.int/healthinfo/country_monitoring_evaluation/who-hmn-framework-standards-chi.pdf. [Accessed 15-April-2020].

[27] Piccoli Gabriele and Pigni Federico. 'Information systems for managers: with cases'. In: 4.0. Prospect Press, 2018, p. 28.

[28] *GDPR official page*. URL: https://eur-lex.europa.eu/eli/reg/2016/679/oj. [Accessed 7-Mail-2020].

[29] *General Data Protection Regulation (GDPR)*. European Union. URL: https://gdpr-info.eu. [Accessed 15-April-2020].

[30] Dieter Gollmann. 'Computer Security'. In: 3rd edition. John Wiley & Sons, Ltd, 2011.

[31] Peter Leo Gorski et al. 'Warn if Secure or How to Deal with Security by Default in Software Development?' In: *The Twelfth International Symposium on Human Aspects of Information Security & Assurance*. 2018. URL: https://www.researchgate.net/publication/327797790_Warn_if_Secure_or_How_to_Deal_with_Security_by_Default_in_Software_Development.

[32] Daniel H Grossoehme. 'Overview of Qualitative Research'. In: *Journal of Health Care Chaplaincy* 20.3 (July 2014), pp. 109–122. DOI: 10.1080/08854726.2014.925660. URL: https://www.researchgate.net/publication/263097154_Overview_of_Qualitative_Research.

[33] Scott Helme. *Cross-Site Request Forgery is dead!* Feb. 2017. URL: https://scotthelme.co.uk/csrf-is-dead/. [Accessed 15-April-2020].

[34] *Information Security Management*. Iso.org. URL: https://www.iso.org/isoiec-27001-information-security.html. [Accessed 15-April-2020].

[35] Trisha Jalan. *MEITY to brief Joint Committee on Personal Data Protection Bill on January 14*. URL: https://www.medianama.com/2020/01/223-personal-data-protection-bill-2019/. [Accessed 15-April-2020].

[36] Yomi Kazeem. *Kenya is stepping up its citizens' digital security with a new EU-inspired data protection law*. Nov. 2019. URL: https://qz.com/africa/1746202/kenya-has-passed-new-data-protection-laws-in-compliance-with-gdpr/. [Accessed 15-April-2020].

[37] Nir Kshetri. 'Cybercrime and Cybersecurity in Africa'. In: *Journal of Global Information Technology Management* 22.2 (2019), pp. 77–81.

[38] Kif Lesling. *Inside Apple's team that greenlights iPhone apps for the App Store*. CNBC. June 2019. URL: https://www.cnbc.com/2019/06/21/how-apples-app-review-process-for-the-app-store-works.html. [Accessed 15-April-2020].

[39] Magnus Li. *Development in Platform Ecosystems*. Lecture slides from course in platforms and DHIS2. 2019. URL: https://www.uio.no/studier/emner/matnat/ifi/IN5320/h19/timeplan/in5320-19-8-isintroduction.pdf. [Accessed 15-April-2020].

[40] Hogan Lovells. *Overview of data protection laws in Africa*. Oct. 2019. URL: https://www.lexology.com/library/detail.aspx?g=82196d1c-2faa-43c2-983b-be3b0f1747f2. [Accessed 15-April-2020].

[41] Jim Manico. *Secure Development Lifecycle*. Powerpoint by OWASP volunteer. URL: https://owasp.org/www-pdf-archive/Jim_Manico_(Hamburg)_-_Securiing_the_SDLC.pdf. [Accessed 15-April-2020].

[42] Somesh Mohanty. *Adopting Secure SDLC Practice for Agile*. DZONE. Apr. 2018. URL: https://dzone.com/articles/adopting-secure-sdlc-practice-for-agile. [Accessed 15-April-2020].

[43] Ernest Mougoue. *SSDLC 101: What Is the Secure Software Development Life Cycle?* July 2017. URL: https://dzone.com/articles/ssdlc-101-what-is-the-secure-software-development. [Accessed 15-April-2020].

[44] *NIST Special Publication 800-53 Revision 4*. NIST. Jan. 2015. URL: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53r4.pdf. [Accessed 15-April-2020].

[45] *Ny sikkerhetslov stiller strengere krav til IT-systemer [New security law places stricter requirements on IT systems]*. Standard Norge. 2020. URL: https://www.standard.no/nyheter/nyhetsarkiv/ikt/nyheter-2019/ny-sikkerhetslov-stiller-strengere-krav-til-it-systemer-/. [Accessed 15-April-2020].

[46] Shannon M Oltmann. 'Qualitative Interviews: A Methodological Discussion of the Interviewer and Respondent Contexts'. In: *Forum Qualitative Sozialforschung* 17.2 (May 2016). URL: https://www.researchgate.net/publication/304876962_Qualitative_Interviews_A_Methodological_Discussion_of_the_Interviewer_and_Respondent_Contexts.

[47] *OWASP Application Security Verification Standard*. OWASP. URL: https://owasp.org/www-project-application-security-verification-standard/. [Accessed 15-April-2020].

[48] *OWASP Top 10*. OWASP. URL: https://owasp.org/www-project-top-ten/. [Accessed 15-April-2020].

[49] *OWASP Top 10 Release Notes*. OWASP. URL: https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_Release_Notes. [Accessed 15-April-2020].

[50] *PCI Security Standards Council*. URL: https://www.pcisecuritystandards.org. [Accessed 10-Mail-2020].

[51] *Red Hat Enterprise Linux 4 Security Guide*. Red Hat Enterprise. Mar. 2020. URL: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/4/pdf/security_guide/Red_Hat_Enterprise_Linux-4-Security_Guide-en-US.pdf. [Accessed 15-April-2020].

[52] Duncan Riley. *Ransomware targets hospitals and medical providers during coronavirus pandemic*. Mar. 2020. URL: https://siliconangle.com/2020/03/16/ransomware-targets-hospitals-medical-providers-coronavirus-pandemic/. [Accessed 15-April-2020].

[53] Margaret Rouse. *Confidentiality, integrity, and availability (CIA triad)*. Apr. 2020. URL: https://whatis.techtarget.com/definition/Confidentiality-integrity-and-availability-CIA. [Accessed 16-April-2020].

[54] Karen Scarfone, Dan Benigni and Tim Grance. 'Cyber Security Standards'. In: *Wiley Handbook of Science and Technology for Homeland Security*. John Wiley & Sons, Inc, 2008.

[55] *SDLC - Overview*. Tutorialspoint. URL: https://www.tutorialspoint.com/sdlc/sdlc_overview.htm. [Accessed 16-April-2020].

[56] *Secure by Default*. National Cyber Security Centre. URL: https://www.ncsc.gov.uk/information/secure-default. [Accessed 15-April-2020].

[57] *Security-by-design Framework*. Cyber Security Agency of Singapore. Nov. 2017. URL: https://www.csa.gov.sg/legislation/supplementary-references. [Accessed 15-April-2020].

[58] *Sensitive information*. TechTarget. 2014. URL: https://whatis.techtarget.com/definition/sensitive-information. [Accessed 15-April-2020].

[59] Linda Shields and Alison Twycross. 'The difference between quantitative and qualitative research'. In: *Paediatric nursing* 15 (2003), p. 24. URL: https://www.researchgate.net/publication/8964072_The_difference_between_quantitative_and_qualitative_research.

[60] *Software Development Process*. Wikipedia. URL: https://en.wikipedia.org/wiki/Software_development_process. [Accessed 15-April-2020].

[61] *Software development with Data Protection by Design and by Default*. Datatilsynet. URL: https://www.datatilsynet.no/en/about-privacy/virksomhetenes-plikter/innebygd-personvern/data-protection-by-design-and-by-default/?print=true. [Accessed 15-April-2020].

[62] *Summary of the HIPAA Security Rule*. HIPAA. URL: https://www.hhs.gov/hipaa/for-professionals/security/laws-regulations/index.html. [Accessed 15-April-2020].

[63] *The 20 CIS Controls & Resources*. Center for Internet Secufrity (CIS). URL: https://www.cisecurity.org/controls/cis-controls-list/. [Accessed 15-April-2020].

[64] Amrit Tiwana. 'Platform Ecosystems: Aligning Architecture, Governance, and Strategy'. In: 1.0. Morgan Kaufmann, 2014. Chap. 1, p. 5. URL: https://www.researchgate.net/publication/286345261_Platform_Ecosystems_Aligning_Architecture_Governance_and_Strategy.

[65] Dan Constantin Tofan. 'Information Security Standards'. In: *Journal of Mobile, Embedded and Distributed Systems* III.3 (2011), pp. 128–129. URL: https://www.researchgate.net/publication/279679417_Information_Security_Standards.

[66] *What are the Software Development Models?* TRYQA. URL: http://tryqa.com/what-are-the-software-development-models/. [Accessed 15-April-2020].

[67] *What does data protection 'by design' and 'by default' mean?* European Comission. URL: https://ec.europa.eu/info/law/law-topic/data-protection/reform/rules-business-and-organisations/obligations/what-does-data-protection-design-and-default-mean_en. [Accessed 15-April-2020].

[68] *What is Cybersecurity?* CISCO. URL: https://www.cisco.com/c/en/us/products/security/what-is-cybersecurity.html. [Accessed 15-April-2020].

[69] *What is Security Requirement*. IGI Global. URL: https://www.igi-global.com/dictionary/discovering-core-security-requirements-drm/26121. [Accessed 15-April-2020].

[70] *Who is the OWASP Foundation?* OWASP. URL: https://owasp.org.

[71] Ian Willoughby. *Cyber-attack causes majoR disruptions to small-town CZECH hospital*. Dec. 2019. URL: https://www.radio.cz/en/section/curraffrs/cyber-attack-causes-major-disruptions-to-small-town-czech-hospital. [Accessed 15-April-2020].

Appendices

Phase 1: Applicability spreadsheet

This spreadsheet shows the results from the applicability phase. To explain the spreadsheet:

- A check for **Applicable** means the control is applicable to software and the DHIS2 development team.
- A check for **Implementation** means the control is applicable to the implementors.
- A check for both **Applicable** and **Implementation** means the control has to be implemented both by the DHIS2 development team and by the local implementors.

Four people participated in this phase. All four responses are shown side-by-side, both for applicability to software, and applicability to implementation. For every four sets of columns, the responses were combined into a new column.

For each participant, a counter shows the number of checked controls and the total number of ASVS controls. The frontend developer, for example, checked 242 out of 286 as applicable to DHIS2.

			C1-C10 Top 10 Proactive Controls: https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering)				242/286	132/286	196/286	256/286	258/286						Total: 46	Total: 21	Total: 45	Total: 80	Total: 79
			ASVS 4.0				Front-end	Release m.	Implementor	Backend	Combined		Front-end developer	Release manager	Implementor	Backend developer	Frontend	Release M.	Implementor	Backend	Combined
Section	**Name**	**Item**	**Description**	**L1**	**L2**	**L3**	**Applicable**	**Applicable**	**Applicable**	**Applicable**	**Applicable**		**Comments**	**Comments**	**Comments**	**Comments**	**Impl.**	**Impl.**	**Impl.**	**Impl.**	**Impl.**
V1	Architecture	1.1.1	Verify the use of a secure software development lifecycle that addresses security in all stages of development. [C1]		X	X	☑	☑	☑	☐	☑						☐	☐	☑	☑	☐
V1	Architecture	1.1.2	Verify the use of threat modeling for every design change or sprint planning to identify threats, plan for countermeasures, facilitate appropriate risk responses, and guide security testing.		X	X	☑	☑	☑	☑	☑						☐	☐	☐	☐	☐
V1	Architecture	1.1.3	Verify that all user stories and features contain functional security constraints, such as "As a user, I should be able to view and edit my profile. I should not be able to view or edit anyone else's profile"		X	X	☑	☑	☑	☑	☑						☐	☐	☐	☐	☐
V1	Architecture	1.1.4	Verify documentation and justification of all the application's trust boundaries, components, and significant data flows.		X	X	☑	☑	☑	☑	☑						☐	☐	☐	☐	☐
V1	Architecture	1.1.5	Verify definition and security analysis of the application's high-level architecture and all connected remote services. [C1]		X	X	☑	☑	☑	☑	☑					The user can opt to use external services such as Amazon cloud. It is part implementation, in that they are themselves in charge of securing it, but from DHIS2s standpoint, if they are abble to secure it, then they do.	☐	☐	☐	☑	☑
V1	Architecture	1.1.6	Verify implementation of centralized, simple (economy of design), vetted, secure, and reusable security controls to avoid duplicate, missing, ineffective, or insecure controls. [C10]		X	X	☑	☑	☑	☑	☑						☐	☐	☐	☐	☐
V1	Architecture	1.1.7	Verify availability of a secure coding checklist, security requirements, guideline, or policy to all developers and testers.		X	X	☑	☑	☑	☑	☑				There is an absence of requirements and guidelines during development		☐	☐	☐	☐	☐
V1	Architecture	1.2.1	Verify the use of unique or special low-privilege operating system accounts for all application components, services, and servers. [C3]		X	X	☐	☐	☐	☐	☐		DHIS2 is deployed "in-house" so servers, etc needs to be secured by the implementors.		Revelant for somebody who's hosting the platform (not developers)		☑	☑	☑	☑	☑
V1	Architecture	1.2.2	Verify that communications between application components, including APIs, middleware and data layers, are authenticated. Components should have the least necessary privileges needed. [C3]		X	X	☑	☑	☐	☑	☑			Last sentence is an implementation (configure issue). To ensure they are authenticated, is an application issue.	In implementation, every user should have an unique data base etc. Applicable to documentation.		☐	☑	☑	☑	☑
V1	Architecture	1.2.3	Verify that the application uses a single vetted authentication mechanism that is known to be secure, can be extended to include strong authentication, and has sufficient logging and monitoring to detect account abuse or breaches.		X	X	☑	☑	☑	☑	☑		Authentication needs to be implemented by the developers for the core software, but logging/monitoring is done on a server basis, and authentication for additional server software needs to be done by the implementors.				☑	☐	☐	☑	☑
V1	Architecture	1.2.4	Verify that all authentication pathways and identity management APIs implement consistent authentication security control strength, such that there are no weaker alternatives per the risk of the application.		X	X	☑	☑	☑	☑	☑						☐	☐	☐	☐	☐
V1	Architecture	1.4.1	Verify that trusted enforcement points such as at access control gateways, servers, and serverless functions enforce access controls. Never enforce access controls on the client.		X	X	☑	☑	☑	☐	☑				/portal gives away username and password, i.e. no access controls on the client (words case example).	Do not enforce access control on the client. Don't have any service functions, sounds like an implementation issue.	☐	☐	☐	☑	☑
V1	Architecture	1.4.2	Verify that the chosen access control solution is flexible enough to meet the application's needs.		X	X	☑	☑	☑	☑	☑						☐	☐	☐	☐	☐
V1	Architecture	1.4.3	Verify enforcement of the principle of least privilege in functions, data files, URLs, controllers, services, and other resources. This implies protection against spoofing and elevation of privilege.		X	X	☑	☑	☑	☑	☑					The system is in place, but the implementors configure it. Have the ability to configure the access to these things, but don't have any setup to hinder spoofing, for example. In the case of elevation, you cannot give more access than what the user has.	☐	☐	☐	☑	☑

			C1-C10 Top 10 Proactive Controls: https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering)				242/286	132/286	196/286	256/286	258/286					Total: 46	Total: 21	Total: 45	Total: 80	Total: 79
V1	Architecture	1.4.4	Verify the application uses a single and well-vetted access control mechanism for accessing protected data and resources. All requests must pass through this single mechanism to avoid copy and paste or insecure alternative paths. [C7]		X	X	☑	☑	☑	☑	☑	True for resources internal to DHIS2, but an application can request additional unprotected resources out of the security measures dhis2 provides.				☐	☐	☐	☐	☐
V1	Architecture	1.4.5	Verify that attribute or feature-based access control is used whereby the code checks the user's authorization for a feature/data item rather than just their role. Permissions should still be allocated using roles. [C7]		X	X	☑	☑	☑	☑	☑				We don't authorize when we check this type of access control; only check permissions.	☐	☐	☐	☐	☐
V1	Architecture	1.5.1	Verify that input and output requirements clearly define how to handle and process data based on type, content, and applicable laws, regulations, and other policy compliance.		X	X	☑	☑	☑	☑	☑	DHIS2 is a self-hosted software, and thus the need to comply with laws and regulations fall on the implementors. (kan ikke endre system til å være kompliant, men kan deaktivetet funksjonaliteten).		Applicable, but leans towards implementation. Designed for many environments, but it's an implementation issue. Depends on what is reported from the field.	Have support for this, but is an implementation issue	☑	☑	☑	☑	☑
V1	Architecture	1.5.2	Verify that serialization is not used when communicating with untrusted clients. If this is not possible, ensure that adequate integrity controls (and possibly encryption if sensitive data is sent) are enforced to prevent deserialization attacks including object injection.		X	X	☑	☑	☑	☑	☑			Related to implementation. At the moment, no way to know if a client should be trusted. In US regulations on patient system you are not allowed to access the resource if the device isn't trusted. Issue in Android; easily hacked.		☐	☐	☑	☑	☑
V1	Architecture	1.5.3	Verify that input validation is enforced on a trusted service layer. [C5]		X	X	☑	☑	☑	☑	☑					☐	☐	☐	☐	☐
V1	Architecture	1.5.4	Verify that output encoding occurs close to or by the interpreter for which it is intended. [C4]		X	X	☑	☑	☑	☑	☑					☐	☐	☐	☐	☐
V1	Architecture	1.6.1	Verify that there is an explicit policy for management of cryptographic keys and that a cryptographic key lifecycle follows a key management standard such as NIST SP 800-57.		X	X	☐	☐	☐	☐	☐	Unsure if we have crypto parts in the codebase, if so, is applicable (needs more research). If we have it, then it is applicable. If not, then it is not applicable.				☐	☑	☑	☑	☑
V1	Architecture	1.6.2	Verify that consumers of cryptographic services protect key material and other secrets by using key vaults or API based alternatives.		X	X	☑	☐	☐	☑	☑	Unsure if we have crypto parts in the codebase, if so, is applicable (needs more research). If we have it, then it is applicable. If not, then it is not applicable.			Pass on their end, but up to the implementors to handle it. Doesnt involve the backend.	☐	☑	☑	☑	☑
V1	Architecture	1.6.3	Verify that all keys and passwords are replaceable and are part of a well-defined process to re-encrypt sensitive data.		X	X	☐	☐	☐	☑	☑	Unsure if we have crypto parts in the codebase, if so, is applicable (needs more research). If we have it, then it is applicable. If not, then it is not applicable.				☐	☑	☑	☑	☑
V1	Architecture	1.6.4	Verify that symmetric keys, passwords, or API secrets generated by or shared with clients are used only in protecting low risk secrets, such as encrypting local storage, or temporary ephemeral uses such as parameter obfuscation. Sharing secrets with clients is clear-text equivalent and architecturally should be treated as such.		X	X	☐	☐	☐	☑	☑	Unsure if we have crypto parts in the codebase, if so, is applicable (needs more research). If we have it, then it is applicable. If not, then it is not applicable.		This is unrelated to software development.		☐	☑	☑	☑	☑
V1	Architecture	1.7.1	Verify that a common logging format and approach is used across the system. [C9]		X	X	☑	☑	☑	☑	☑	For DHIS2 this is applicable for the core team, for surrounding software such as nginx (web server), the implementor is responsible.				☑	☐	☐	☑	☑

			C1-C10 Top 10 Proactive Controls: https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering)				242/286	132/286	196/286	256/286	258/286						Total: 46	Total: 21	Total: 45	Total: 80	Total: 79
V1	Architecture	1.7.2	Verify that logs are securely transmitted to a preferably remote system for analysis, detection, alerting, and escalation. [C9]		X	X	☐	☐	☐	☑	☑						☑	☑	☑	☑	☑
V1	Architecture	1.8.1	Verify that all sensitive data is identified and classified into protection levels.		X	X	☑	☐	☐	☑	☑	For non-aggregated data, such as data related to individuals in Tracker, there should be some form of protection levels.	Don't know what's sensitive	Not possible for developments to classify data; up to local governments. Up to developers to allow the application to classify data.			☐	☑	☑	☑	☑
V1	Architecture	1.8.2	Verify that all protection levels have an associated set of protection requirements, such as encryption requirements, integrity requirements, retention, privacy and other confidentiality requirements, and that these are applied in the architecture.		X	X	☑	☑	☑	☑	☑						☐	☐	☐	☑	☑
V1	Architecture	1.9.1	Verify the application encrypts communications between components, particularly when these components are in different containers, systems, sites, or cloud providers. [C3]		X	X	☑	☑	☐	☑	☑	The core DHIS2 software needs to guarantee use of HTTPS, and on the implementation side, SSL termination must be done according to secure standards.			Support for using https and ssl, but also implementation		☑	☐	☑	☑	☑
V1	Architecture	1.9.2	Verify that application components verify the authenticity of each side in a communication link to prevent person-in-the-middle attacks. For example, application components should validate TLS certificates and chains.		X	X	☑	☑	☐	☑	☑		Valid for desktop users, but android is another dimension.	Android related, using PKI, VPN.	Have support for TLS, but unsure if they validate certificates		☑	☑	☑	☑	☑
V1	Architecture	1.10.1	Verify that a source code control system is in use, with procedures to ensure that check-ins are accompanied by issues or change tickets. The source code control system should have access control and identifiable users to allow traceability of any changes.		X	X	☑	☑	☑	☑	☑						☐	☐	☐	☐	☐
V1	Architecture	1.11.1	Verify the definition and documentation of all application components in terms of the business or security functions they provide.		X	X	☑	☑	☑	☑	☑						☐	☐	☐	☐	☐
V1	Architecture	1.11.2	Verify that all high-value business logic flows, including authentication, session management and access control, do not share unsynchronized state.		X	X	☑	☑	☑	☑	☑						☐	☐	☐	☐	☐
V1	Architecture	1.11.3	Verify that all high-value business logic flows, including authentication, session management and access control are thread safe and resistant to time-of-check and time-of-use race conditions.			X	☑	☑	☑	☑	☑				Dont have race conditions posiblities in access contorls. Everything is done in one thread.		☐	☐	☐	☐	☐
V1	Architecture	1.12.1	Verify that user-uploaded files are stored outside of the web root.		X	X	☑	☑	☑	☑	☑		It has to be a configuration. If the system stores the files inside the root, someone could configure the directory where the data is stored inside the web root				☑	☑	☑	☑	☑
V1	Architecture	1.12.2	Verify that user-uploaded files - if required to be displayed or downloaded from the application - are served by either octet stream downloads, or from an unrelated domain, such as a cloud file storage bucket. Implement a suitable content security policy to reduce the risk from XSS vectors or other attacks from the uploaded file.		X	X	☑	☑	☑	☑	☑	One use case is installing web apps from the DHIS2 App Store.					☐	☐	☐	☐	☐
V1	Architecture	1.14.1	Verify the segregation of components of differing trust levels through well-defined security controls, firewall rules, API gateways, reverse proxies, cloud-based security groups, or similar mechanisms.		X	X	☐	☐	☐	☐	☐		Securiy firewalls and proxy are all dependending on how you implement it.				☑	☑	☑	☑	☑
V1	Architecture	1.14.2	Verify that if deploying binaries to untrusted devices makes use of binary signatures, trusted connections, and verified endpoints.		X	X	☐	☐	☐	☐	☐						☐	☑	☑	☑	☑
V1	Architecture	1.14.3	Verify that the build pipeline warns of out-of-date or insecure components and takes appropriate actions.		X	X	☑	☑	☑	☑	☑						☐	☐	☐	☐	☐
V1	Architecture	1.14.4	Verify that the build pipeline contains a build step to automatically build and verify the secure deployment of the application, particularly if the application infrastructure is software defined, such as cloud environment build scripts.		X	X	☑	☑	☑	☑	☑	Done for e.g. the App Store backend.					☐	☐	☐	☐	☐
V1	Architecture	1.14.5	Verify that application deployments adequately sandbox, containerize and/or isolate at the network level to delay and deter attackers from attacking other applications, especially when they are performing sensitive or dangerous actions such as deserialization. [C5]		X	X	☐	☐	☐	☐	☐		DHIS2 helps with implementation				☑	☑	☑	☑	☑
V1	Architecture	1.14.6	Verify the application does not use unsupported, insecure, or deprecated client-side technologies such as NSAPI plugins, Flash, Shockwave, ActiveX, Silverlight, NACL, or client-side Java applets.		X	X	☑	☑	☑	☑	☑		People can create their own clients for the API, which DHIS2 can't control. Have to make sure the API is secure. Applicable for the core apps which should be secure.		Do not use any of these things, but have client technologies that they know are insequece, but that they are working on securing		☐	☐	☐	☐	☐
V2	Authentication	2.1.1	Verify that user set passwords are at least 12 characters in length. [C6]	X	X	X	☑	☑	☑	☑	☑				You can configure it to be 12 characters.		☐	☐	☐	☐	☐
V2	Authentication	2.1.2	Verify that passwords 64 characters or longer are permitted. [C6]	X	X	X	☑	☑	☑	☑	☑						☐	☐	☐	☐	☐

			C1-C10 Top 10 Proactive Controls: https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering)				242/286	132/286	196/286	256/286	258/286						Total: 46	Total: 21	Total: 45	Total: 80	Total: 79
V2	Authentication	2.1.3	Verify that passwords can contain spaces and truncation is not performed. Consecutive multiple spaces MAY optionally be coalesced. [C6]	X	X	X	☑	☑	☑	☑	☑						☐	☐	☐	☐	☐
V2	Authentication	2.1.4	Verify that Unicode characters are permitted in passwords. A single Unicode code point is considered a character, so 12 emoji or 64 kanji characters should be valid and permitted.	X	X	X	☑	☑	☑	☑	☑						☐	☐	☐	☐	☐
V2	Authentication	2.1.5	Verify users can change their password.	X	X	X	☑	☑	☑	☑	☑						☐	☐	☐	☐	☐
V2	Authentication	2.1.6	Verify that password change functionality requires the user's current and new password.	X	X	X	☑	☑	☑	☑	☑						☐	☐	☐	☐	☐
V2	Authentication	2.1.7	Verify that passwords submitted during account registration, login, and password change are checked against a set of breached passwords either locally (such as the top 1,000 or 10,000 most common passwords which match the system's password policy) or using an external API. If using an API a zero knowledge proof or other mechanism should be used to ensure that the plain text password is not sent or used in verifying the breach status of the password. If the password is breached, the application must require the user to set a new non-breached password. [C6]	X	X	X	☑	☑	☑	☑	☑						☐	☐	☐	☐	☐
V2	Authentication	2.1.8	Verify that a password strength meter is provided to help users set a stronger password.	X	X	X	☑	☑	☑	☑	☑						☐	☐	☐	☐	☐
V2	Authentication	2.1.9	Verify that there are no password composition rules limiting the type of characters permitted. There should be no requirement for upper or lower case or numbers or special characters. [C6]	X	X	X	☑	☑	☑	☑	☑				Should have the possiblity to remove the configuration in the future		☐	☐	☐	☐	☐
V2	Authentication	2.1.10	Verify that there are no periodic credential rotation or password history requirements.	X	X	X	☑	☑	☑	☑	☑			Different requirements on different districts.	Can opt out		☑	☐	☐	☑	☑
V2	Authentication	2.1.11	Verify that "paste" functionality, browser password helpers, and external password managers are permitted.	X	X	X	☑	☑	☑	☑	☑						☐	☐	☐	☐	☐
V2	Authentication	2.1.12	Verify that the user can choose to either temporarily view the entire masked password, or temporarily view the last typed character of the password on platforms that do not have this as native functionality.	X	X	X	☐	☑	☑	☑	☑						☐	☐	☐	☐	☐
V2	Authentication	2.2.1	Verify that anti-automation controls are effective at mitigating breached credential testing, brute force, and account lockout attacks. Such controls include blocking the most common breached passwords, soft lockouts, rate limiting, CAPTCHA, ever increasing delays between attempts, IP address restrictions, or risk-based restrictions such as location, first login on a device, recent attempts to unlock the account, or similar. Verify that no more than 100 failed attempts per hour is possible on a single account.	X	X	X	☑	☑	☑	☑	☑				Have possiblity to opt-in and use CAPTCHA		☐	☐	☐	☐	☐
V2	Authentication	2.2.2	Verify that the use of weak authenticators (such as SMS and email) is limited to secondary verification and transaction approval and not as a replacement for more secure authentication methods. Verify that stronger methods are offered before weak methods, users are aware of the risks, or that proper measures are in place to limit the risks of account compromise.	X	X	X	☑	☑	☑	☑	☑						☐	☐	☐	☐	☐
V2	Authentication	2.2.3	Verify that secure notifications are sent to users after updates to authentication details, such as credential resets, email or address changes, logging in from unknown or risky locations. The use of push notifications - rather than SMS or email - is preferred, but in the absence of push notifications, SMS or email is acceptable as long as no sensitive information is disclosed in the notification.	X	X	X	☑	☑	☑	☑	☑						☐	☐	☐	☐	☐
V2	Authentication	2.2.4	Verify impersonation resistance against phishing, such as the use of multi-factor authentication, cryptographic devices with intent (such as connected keys with a push to authenticate), or at higher AAL levels, client-side certificates.			X	☑	☑	☑	☑	☑	2FA is a setting, not mandatory, so up to the implementor to enable/require it			Have the possiblity to enale multi-factor.		☑	☐	☐	☑	☑
V2	Authentication	2.2.5	Verify that where a credential service provider (CSP) and the application verifying authentication are separated, mutually authenticated TLS is in place between the two endpoints.			X	☑	☑	☐	☑	☑		External service to configure 2FA				☑	☑	☑	☑	☑
V2	Authentication	2.2.6	Verify replay resistance through the mandated use of OTP devices, cryptographic authenticators, or lookup codes.			X	☐	☐	☐	☐	☐		"Mandated" is imporant. But the system has to support it.	Don't provide those services. Imlementation issue.	Don't use OTP. Have tokens when you reset password, which are temporariy until you have altered your password		☐	☑	☑	☑	☑
V2	Authentication	2.2.7	Verify intent to authenticate by requiring the entry of an OTP token or user-initiated action such as a button press on a FIDO hardware key.			X	☐	☑	☑	☑	☑						☐	☐	☐	☐	☐
V2	Authentication	2.3.1	Verify system generated initial passwords or activation codes SHOULD be securely randomly generated, SHOULD be at least 6 characters long, and MAY contain letters and numbers, and expire after a short period of time. These initial secrets must not be permitted to become the long term password.	X	X	X	☑	☑	☑	☑	☑						☐	☐	☐	☐	☐
V2	Authentication	2.3.2	Verify that enrollment and use of subscriber-provided authentication devices are supported, such as a U2F or FIDO tokens.		X	X	☐	☑	☐	☑	☑		Have to support it.	Hard to do overseas.			☐	☐	☑	☐	☐
V2	Authentication	2.3.3	Verify that renewal instructions are sent with sufficient time to renew time bound authenticators.		X	X	☑	☑	☑	☑	☑						☐	☐	☐	☐	☐
V2	Authentication	2.4.1	Verify that passwords are stored in a form that is resistant to offline attacks. Passwords SHALL be salted and hashed using an approved one-way key derivation or password hashing function. Key derivation and password hashing functions take a password, a salt, and a cost factor as inputs when generating a password hash. [C6]		X	X	☑	☑	☑	☑	☑						☐	☐	☐	☐	☐
V2	Authentication	2.4.2	Verify that the salt is at least 32 bits in length and be chosen arbitrarily to minimize salt value collisions among stored hashes. For each credential, a unique salt value and the resulting hash SHALL be stored. [C6]		X	X	☑	☑	☑	☑	☑				We use a different framework, Sprint Security. Don't do the last bit.		☐	☐	☐	☐	☐

			C1-C10 Top 10 Proactive Controls: https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering)				242/286	132/286	196/286	256/286	258/286						Total: 46	Total: 21	Total: 45	Total: 80	Total: 79
V2	Authentication	2.4.3	Verify that if PBKDF2 is used, the iteration count SHOULD be as large as verification server performance will allow, typically at least 100,000 iterations. [C6]		X	X	☑	☐	☐	☑	☑						☐	☐	☐	☐	☐
V2	Authentication	2.4.4	Verify that if bcrypt is used, the work factor SHOULD be as large as verification server performance will allow, typically at least 13. [C6]		X	X	☑	☐	☐	☑	☑						☐	☐	☐	☑	☑
V2	Authentication	2.4.5	Verify that an additional iteration of a key derivation function is performed, using a salt value that is secret and known only to the verifier. Generate the salt value using an approved random bit generator [SP 800-90Ar1] and provide at least the minimum security strength specified in the latest revision of SP 800-131A. The secret salt value SHALL be stored separately from the hashed passwords (e.g., in a specialized device like a hardware security module).		X	X	☐	☐	☐	☐	☐						☐	☐	☐	☐	☐
V2	Authentication	2.5.1	Verify that a system generated initial activation or recovery secret is not sent in clear text to the user. [C6]	X	X	X	☑	☑	☑	☑	☑						☐	☐	☐	☐	☐
V2	Authentication	2.5.2	Verify password hints or knowledge-based authentication (so-called "secret questions") are not present.	X	X	X	☑	☑	☑	☑	☑						☐	☐	☐	☐	☐
V2	Authentication	2.5.3	Verify password credential recovery does not reveal the current password in any way. [C6]	X	X	X	☑	☑	☑	☑	☑						☐	☐	☐	☐	☐
V2	Authentication	2.5.4	Verify shared or default accounts are not present (e.g. "root", "admin", or "sa").	X	X	X	☑	☐	☑	☑	☑		The accounts is something implementation configures themselves. There is a built in admin account when you bootstrap the system that is used to create other accounts This should be disabled later on. DHIS2 can't do anything about it	Admin should be disabled in implementation. People in other countries don't disable it themselves and leave it present.			☑	☑	☑	☑	☑
V2	Authentication	2.5.5	Verify that if an authentication factor is changed or replaced, that the user is notified of this event.	X	X	X	☑	☑	☑	☑	☑						☐	☐	☐	☐	☐
V2	Authentication	2.5.6	Verify forgotten password, and other recovery paths use a secure recovery mechanism, such as TOTP or other soft token, mobile push, or another offline recovery mechanism. [C6]	X	X	X	☐	☑	☑	☑	☑						☐	☐	☐	☐	☐
V2	Authentication	2.5.7	Verify that if OTP or multi-factor authentication factors are lost, that evidence of identity proofing is performed at the same level as during enrollment.		X	X	☑	☑	☐	☑	☑			Dont use much multi authentication	Implemenation because the way you recover is that you get in touch with the system adminstrator.		☐	☐	☐	☑	☑
V2	Authentication	2.6.1	Verify that lookup secrets can be used only once.		X	X	☐	☐	☐	☐	☐	Unknown to me if we use lookup secrets in this form					☐	☐	☐	☐	☐
V2	Authentication	2.6.2	Verify that lookup secrets have sufficient randomness (112 bits of entropy), or if less than 112 bits of entropy, salted with a unique and random 32-bit salt and hashed with an approved one-way hash.		X	X	☐	☐	☐	☐	☐						☐	☐	☐	☐	☐
V2	Authentication	2.6.3	Verify that lookup secrets are resistant to offline attacks, such as predictable values.		X	X	☐	☐	☐	☐	☐						☐	☐	☐	☐	☐
V2	Authentication	2.7.1	Verify that clear text out of band (NIST "restricted") authenticators, such as SMS or PSTN, are not offered by default, and stronger alternatives such as push notifications are offered first.	X	X	X	☑	☑	☑	☑	☑	I believe the 2FA implementation uses SMS/E-mail that may count as out of band authentication, so may be applicable unless I am mistaken.					☐	☐	☐	☑	☑
V2	Authentication	2.7.2	Verify that the out of band verifier expires out of band authentication requests, codes, or tokens after 10 minutes.	X	X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V2	Authentication	2.7.3	Verify that the out of band verifier authentication requests, codes, or tokens are only usable once, and only for the original authentication request.	X	X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V2	Authentication	2.7.4	Verify that the out of band authenticator and verifier communicates over a secure independent channel.	X	X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V2	Authentication	2.7.5	Verify that the out of band verifier retains only a hashed version of the authentication code.		X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V2	Authentication	2.7.6	Verify that the initial authentication code is generated by a secure random number generator, containing at least 20 bits of entropy (typically a six digital random number is sufficient).		X	X	☑	☐	☐	☑	☑						☐	☐	☑	☐	☐
V2	Authentication	2.8.1	Verify that time-based OTPs have a defined lifetime before expiring.	X	X	X	☐	☐	☑	☐	☐		Not sure if they have OTP's		Confident there is a lifetime, but unsure if there are OTP's.		☐	☐	☐	☐	☐
V2	Authentication	2.8.2	Verify that symmetric keys used to verify submitted OTPs are highly protected, such as by using a hardware security module or secure operating system based key storage.		X	X	☐	☐	☐	☐	☐			Most of time-based is relevant if googles 2FU is used.			☐	☐	☐	☐	☐
V2	Authentication	2.8.3	Verify that approved cryptographic algorithms are used in the generation, seeding, and verification.		X	X	☐	☐	☐	☑	☑						☐	☐	☐	☐	☐
V2	Authentication	2.8.4	Verify that time-based OTP can be used only once within the validity period.		X	X	☐	☐	☐	☐	☐						☐	☐	☐	☐	☐

			C1-C10 Top 10 Proactive Controls: https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering)				242/286	132/286	196/286	256/286	258/286						Total: 46	Total: 21	Total: 45	Total: 80	Total: 79
V2	Authentication	2.8.5	Verify that if a time-based multi factor OTP token is re-used during the validity period, it is logged and rejected with secure notifications being sent to the holder of the device.		X	X	☐	☐	☐	☐	☐						☐	☐	☐	☐	☐
V2	Authentication	2.8.6	Verify physical single factor OTP generator can be revoked in case of theft or other loss. Ensure that revocation is immediately effective across logged in sessions, regardless of location.		X	X	☐	☐	☐	☐	☐						☐	☐	☐	☐	☐
V2	Authentication	2.8.7	Verify that biometric authenticators are limited to use only as secondary factors in conjunction with either something you have and something you know.		o	X	☐	☐	☐	☐	☐	We don't have biometric auth yet, but it is something we are looking into and will become applicable at that point			Possibly for android app, but the primary goal with biometrics is verification of people, patients etc.		☐	☐	☐	☐	☐
V2	Authentication	2.9.1	Verify that cryptographic keys used in verification are stored securely and protected against disclosure, such as using a TPM or HSM, or an OS service that can use this secure storage.		X	X	☐	☐	☐	☐	☐						☐	☐	☐	☐	☐
V2	Authentication	2.9.2	Verify that the challenge nonce is at least 64 bits in length, and statistically unique or unique over the lifetime of the cryptographic device.		X	X	☐	☑	☐	☐	☐						☐	☐	☐	☐	☐
V2	Authentication	2.9.3	Verify that approved cryptographic algorithms are used in the generation, seeding, and verification.		X	X	☐	☑	☑	☑	☑						☐	☐	☐	☐	☐
V2	Authentication	2.10.1	Verify that integration secrets do not rely on unchanging passwords, such as API keys or shared privileged accounts.		OS assisted	HSM	☑	☐	☑	☑	☑						☑	☐	☐	☑	☑
V2	Authentication	2.10.2	Verify that if passwords are required, the credentials are not a default account.		OS assisted	HSM	☑	☑	☑	☑	☑						☑	☐	☐	☑	☑
V2	Authentication	2.10.3	Verify that passwords are stored with sufficient protection to prevent offline recovery attacks, including local system access.		OS assisted	HSM	☑	☑	☑	☑	☑						☑	☐	☐	☑	☑
V2	Authentication	2.10.4	Verify passwords, integrations with databases and third-party systems, seeds and internal secrets, and API keys are managed securely and not included in the source code or stored within source code repositories. Such storage SHOULD resist offline attacks. The use of a secure software key store (L1), hardware trusted platform module (TPM), or a hardware security module (L3) is recommended for password storage.		OS assisted	HSM	☑	☑	☑	☑	☑	DHIS2 core team is responsible for not putting credentials in the source code, but implementors for securely storing the information on the server.		Passwords stored in source code.	Have an external config for this, but implementors are in charge of using it correctly.		☑	☐	☐	☑	☑
V3	Session	3.1.1	Verify the application never reveals session tokens in URL parameters or error messages.	X	X	X	☑	☑	☑	☑	☑						☐	☐	☐	☐	☐
V3	Session	3.2.1	Verify the application generates a new session token on user authentication. [C6]	X	X	X	☑	☑	☑	☑	☑						☐	☐	☐	☐	☐
V3	Session	3.2.2	Verify that session tokens possess at least 64 bits of entropy. [C6]	X	X	X	☑	☑	☑	☑	☑						☐	☐	☐	☐	☐
V3	Session	3.2.3	Verify the application only stores session tokens in the browser using secure methods such as appropriately secured cookies (see section 3.4) or HTML 5 session storage.	X	X	X	☑	☑	☑	☑	☑	SameSite cookies need to be enabled on the nginx configuration					☑	☐	☐	☑	☑
V3	Session	3.2.4	Verify that session token are generated using approved cryptographic algorithms. [C6]		X	X	☑	☑	☑	☑	☑						☐	☐	☐	☐	☐
V3	Session	3.3.1	Verify that logout and expiration invalidate the session token, such that the back button or a downstream relying party does not resume an authenticated session, including across relying parties. [C6]	X	X	X	☑	☑	☑	☑	☑						☐	☐	☐	☐	☐
V3	Session	3.3.2	If authenticators permit users to remain logged in, verify that re-authentication occurs periodically both when actively used or after an idle period. [C6]	30 days	12 hours or 30 minutes of inactivity, 2FA optional	12 hours or 15 minutes of inactivity, with 2FA	☑	☑	☑	☑	☑			DHIS2 has a mechanism for detecting if its online or not. Is used to send periodic pings, so the client would know if online or not. The problem was that the user was never offline, was always logged in. The default is 1 hour. Banking is 10 minutes.			☐	☐	☐	☐	☐
V3	Session	3.3.3	Verify that the application terminates all other active sessions after a successful password change, and that this is effective across the application, federated login (if present), and any relying parties.		X	X	☑	☑	☑	☑	☑						☐	☐	☐	☐	☐
V3	Session	3.3.4	Verify that users are able to view and log out of any or all currently active sessions and devices.		X	X	☑	☑	☑	☑	☑						☐	☐	☐	☐	☐
V3	Session	3.4.1	Verify that cookie-based session tokens have the 'Secure' attribute set. [C6]	X	X	X	☑	☑	☑	☑	☑			Do it on the implementation side with the web proxy. Developers want to run it without HTTPS.			☐	☐	☐	☐	☐
V3	Session	3.4.2	Verify that cookie-based session tokens have the 'HttpOnly' attribute set. [C6]	X	X	X	☑	☑	☑	☑	☑						☐	☐	☐	☐	☐
V3	Session	3.4.3	Verify that cookie-based session tokens utilize the 'SameSite' attribute to limit exposure to cross-site request forgery attacks. [C6]	X	X	X	☑	☑	☑	☑	☑	This is not part of the Java Servlet specs, so needs to be configured through nginx (see m106)					☑	☐	☐	☑	☑

			C1-C10 Top 10 Proactive Controls: https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering)				242/286	132/286	196/286	256/286	258/286						Total: 46	Total: 21	Total: 45	Total: 80	Total: 79
V3	Session	3.4.4	Verify that cookie-based session tokens use "__Host-" prefix (see references) to provide session cookie confidentiality.	X	X	X	☑	☑	☑	☑	☑						☐	☐	☐	☐	☐
V3	Session	3.4.5	Verify that if the application is published under a domain name with other applications that set or use session cookies that might override or disclose the session cookies, set the path attribute in cookie-based session tokens using the most precise path possible. [C6]	X	X	X	☑	☑	☑	☑	☑		Possibly implementation. The impelmentors must configure domains, network aspects.				☑	☑	☑	☑	☑
V3	Session	3.5.1	Verify the application does not treat OAuth and refresh tokens — on their own — as the presence of the subscriber and allows users to terminate trust relationships with linked applications.		X	X	☑	☑	☑	☑	☑						☐	☐	☐	☐	☐
V3	Session	3.5.2	Verify the application uses session tokens rather than static API secrets and keys, except with legacy implementations.		X	X	☑	☑	☑	☑	☑						☐	☐	☐	☐	☐
V3	Session	3.5.3	Verify that stateless session tokens use digital signatures, encryption, and other countermeasures to protect against tampering, enveloping, replay, null cipher, and key substitution attacks.		X	X	☐	☑	☑	☐	☐	Not applicable right now, but we are working on stateless session tokens, so it will be at some point.					☐	☐	☐	☐	☐
V3	Session	3.6.1	Verify that relying parties specify the maximum authentication time to CSPs and that CSPs re-authenticate the subscriber if they haven't used a session within that period.			X	☑	☑	☑	☑	☑						☑	☐	☐	☑	☑
V3	Session	3.6.2	Verify that CSPs inform relying parties of the last authentication event, to allow RPs to determine if they need to re-authenticate the user.			X	☑	☑	☑	☑	☑						☑	☐	☐	☑	☑
V3	Session	3.7.1	Verify the application ensures a valid login session or requires re-authentication or secondary verification before allowing any sensitive transactions or account modifications.	X	X	X	☑	☑	☑	☑	☑						☐	☐	☐	☐	☐
V4	Access	4.1.1	Verify that the application enforces access control rules on a trusted service layer, especially if client-side access control is present and could be bypassed.	X	X	X	☑	☑	☑	☑	☑						☐	☐	☐	☐	☐
V4	Access	4.1.2	Verify that all user and data attributes and policy information used by access controls cannot be manipulated by end users unless specifically authorized.	X	X	X	☑	☑	☑	☑	☑						☐	☐	☐	☐	☐
V4	Access	4.1.3	Verify that the principle of least privilege exists - users should only be able to access functions, data files, URLs, controllers, services, and other resources, for which they possess specific authorization. This implies protection against spoofing and elevation of privilege. [C7]	X	X	X	☑	☑	☑	☑	☑		Depends on how you define your roles and privlidges. Mostly application.				☐	☐	☐	☑	☑
V4	Access	4.1.4	Verify that the principle of deny by default exists whereby new users/roles start with minimal or no permissions and users/roles do not receive access to new features until access is explicitly assigned. [C7]	X	X	X	☑	☑	☑	☑	☑						☐	☐	☐	☐	☐
V4	Access	4.1.5	Verify that access controls fail securely including when an exception occurs. [C10]	X	X	X	☑	☑	☑	☑	☑						☐	☐	☐	☐	☐
V4	Access	4.2.1	Verify that sensitive data and APIs are protected against direct object attacks targeting creation, reading, updating and deletion of records, such as creating or updating someone else's record, viewing everyone's records, or deleting all records.	X	X	X	☑	☑	☑	☑	☑						☐	☐	☐	☐	☐
V4	Access	4.2.2	Verify that the application or framework enforces a strong anti-CSRF mechanism to protect authenticated functionality, and effective anti-automation or anti-CSRF protects unauthenticated functionality.	X	X	X	☑	☑	☑	☑	☑			Controversial: the standard protection is to use nonse tokens. Not implemented in DHIS2, for a variety of legacty reasons and due to how the API is used. Only protection is server-side cookies, currently. Wants server-side cookie to be on default			☐	☐	☐	☐	☐
V4	Access	4.3.1	Verify administrative interfaces use appropriate multi-factor authentication to prevent unauthorized use.	X	X	X	☑	☑	☑	☑	☑		We dont enforce multi-factor, but we make it available.	Design choice			☐	☐	☐	☐	☐
V4	Access	4.3.2	Verify that directory browsing is disabled unless deliberately desired. Additionally, applications should not allow discovery or disclosure of file or directory metadata, such as Thumbs.db, .DS_Store, .git or .svn folders.	X	X	X	☑	☑	☑	☑	☑			Depends on how you set up your apache server			☑	☐	☑	☑	☑
V4	Access	4.3.3	Verify the application has additional authorization (such as step up or adaptive authentication) for lower value systems, and / or segregation of duties for high value applications to enforce anti-fraud controls as per the risk of application and past fraud.		X	X	☐	☐	☑	☑	☑						☐	☐	☐	☐	☐
V5	Validation	5.1.1	Verify that the application has defenses against HTTP parameter pollution attacks, particularly if the application framework makes no distinction about the source of request parameters (GET, POST, cookies, headers, or environment variables).	X	X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V5	Validation	5.1.2	Verify that frameworks protect against mass parameter assignment attacks, or that the application has countermeasures to protect against unsafe parameter assignment, such as marking fields private or similar. [C5]	X	X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V5	Validation	5.1.3	Verify that all input (HTML form fields, REST requests, URL parameters, HTTP headers, cookies, batch files, RSS feeds, etc) is validated using positive validation (whitelisting). [C5]	X	X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐

			C1-C10 Top 10 Proactive Controls: https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering)				242/286	132/286	196/286	256/286	258/286						Total: 46	Total: 21	Total: 45	Total: 80	Total: 79
V5	Validation	5.1.4	Verify that structured data is strongly typed and validated against a defined schema including allowed characters, length and pattern (e.g. credit card numbers or telephone, or validating that two related fields are reasonable, such as checking that suburb and zip/postcode match). [C5]	X	X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V5	Validation	5.1.5	Verify that URL redirects and forwards only allow whitelisted destinations, or show a warning when redirecting to potentially untrusted content.	X	X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V5	Validation	5.2.1	Verify that all untrusted HTML input from WYSIWYG editors or similar is properly sanitized with an HTML sanitizer library or framework feature. [C5]	X	X	X	☑	☐	☑	☑	☑			Currently a problem. A feature called Interpretations, which has a rich text editor where people can write anything. Has been a problem with XSS before, due to satinazing.			☐	☐	☐	☐	☐
V5	Validation	5.2.2	Verify that unstructured data is sanitized to enforce safety measures such as allowed characters and length.	X	X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V5	Validation	5.2.3	Verify that the application sanitizes user input before passing to mail systems to protect against SMTP or IMAP injection.	X	X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V5	Validation	5.2.4	Verify that the application avoids the use of eval() or other dynamic code execution features. Where there is no alternative, any user input being included must be sanitized or sandboxed before being executed.	X	X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V5	Validation	5.2.5	Verify that the application protects against template injection attacks by ensuring that any user input being included is sanitized or sandboxed.	X	X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V5	Validation	5.2.6	Verify that the application protects against SSRF attacks, by validating or sanitizing untrusted data or HTTP file metadata, such as filenames and URL input fields, use whitelisting of protocols, domains, paths and ports.	X	X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V5	Validation	5.2.7	Verify that the application sanitizes, disables, or sandboxes user-supplied SVG scriptable content, especially as they relate to XSS resulting from inline scripts, and foreignObject.	X	X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V5	Validation	5.2.8	Verify that the application sanitizes, disables, or sandboxes user-supplied scriptable or expression template language content, such as Markdown, CSS or XSL stylesheets, BBCode, or similar.	X	X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V5	Validation	5.3.1	Verify that output encoding is relevant for the interpreter and context required. For example, use encoders specifically for HTML values, HTML attributes, JavaScript, URL Parameters, HTTP headers, SMTP, and others as the context requires, especially from untrusted inputs (e.g. names with Unicode or apostrophes, such as ねこ or O'Hara). [C4]	X	X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V5	Validation	5.3.2	Verify that output encoding preserves the user's chosen character set and locale, such that any Unicode character point is valid and safely handled. [C4]	X	X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V5	Validation	5.3.3	Verify that context-aware, preferably automated - or at worst, manual - output escaping protects against reflected, stored, and DOM based XSS. [C4]	X	X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V5	Validation	5.3.4	Verify that data selection or database queries (e.g. SQL, HQL, ORM, NoSQL) use parameterized queries, ORMs, entity frameworks, or are otherwise protected from database injection attacks. [C3]	X	X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V5	Validation	5.3.5	Verify that where parameterized or safer mechanisms are not present, context-specific output encoding is used to protect against injection attacks, such as the use of SQL escaping to protect against SQL injection. [C3, C4]	X	X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V5	Validation	5.3.6	Verify that the application projects against JavaScript or JSON injection attacks, including for eval attacks, remote JavaScript includes, CSP bypasses, DOM XSS, and JavaScript expression evaluation. [C4]	X	X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V5	Validation	5.3.7	Verify that the application protects against LDAP Injection vulnerabilities, or that specific security controls to prevent LDAP Injection have been implemented. [C4]	X	X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V5	Validation	5.3.8	Verify that the application protects against OS command injection and that operating system calls use parameterized OS queries or use contextual command line output encoding. [C4]	X	X	X	☑	☐	☑	☑	☑		We do not use straight syscalls to the OS and rely on the JVM to do so, unsure if we should handle known syscalls security risks between the JVM and OS.				☐	☐	☐	☐	☐
V5	Validation	5.3.9	Verify that the application protects against Local File Inclusion (LFI) or Remote File Inclusion (RFI) attacks.	X	X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V5	Validation	5.3.10	Verify that the application protects against XPath injection or XML injection attacks. [C4]	X	X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V5	Validation	5.4.1	Verify that the application uses memory-safe string, safer memory copy and pointer arithmetic to detect or prevent stack, buffer, or heap overflows.		X	X	☑	☐	☐	☑	☑		Assuming normal use of Java this should be an easy pass, though custom StringBuffer implementations could cause JVM heap size issues.		Not a problem with Java		☐	☐	☐	☐	☐
V5	Validation	5.4.2	Verify that format strings do not take potentially hostile input, and are constant.		X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V5	Validation	5.4.3	Verify that sign, range, and input validation techniques are used to prevent integer overflows.		X	X	☑	☐	☐	☑	☑				Not a java issue	Possibly not relevant for	☐	☐	☐	☐	☐

			C1-C10 Top 10 Proactive Controls: https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering)				242/286	132/286	196/286	256/286	258/286						Total: 46	Total: 21	Total: 45	Total: 80	Total: 79
V5	Validation	5.5.1	Verify that serialized objects use integrity checks or are encrypted to prevent hostile object creation or data tampering. [C5]	X	X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V5	Validation	5.5.2	Verify that the application correctly restricts XML parsers to only use the most restrictive configuration possible and to ensure that unsafe features such as resolving external entities are disabled to prevent XXE.	X	X	X	☑	☐	☑	☑	☑			Had problem with XXE entity injection before.			☐	☐	☐	☐	☐
V5	Validation	5.5.3	Verify that deserialization of untrusted data is avoided or is protected in both custom code and third-party libraries (such as JSON, XML and YAML parsers).	X	X	X	☑	☐	☑	☑	☑				All that is user supplied, so in that sense it is trusted since users need to log in		☐	☐	☐	☐	☐
V5	Validation	5.5.4	Verify that when parsing JSON in browsers or JavaScript-based backends, JSON.parse is used to parse the JSON document. Do not use eval() to parse JSON.	X	X	X	☑	☐	☑	☑	☑	Relevant for the App Store backend.					☐	☐	☐	☐	☐
V6	Cryptography	6.1.1	Verify that regulated private data is stored encrypted while at rest, such as personally identifiable information (PII), sensitive personal information, or data assessed likely to be subject to EU's GDPR.		X	X	☑	☐	☑	☑	☑			Reommended to make sure the database is on an encrypted disk			☑	☐	☑	☑	☑
V6	Cryptography	6.1.2	Verify that regulated health data is stored encrypted while at rest, such as medical records, medical device details, or de-anonymized research records.		X	X	☑	☐	☑	☑	☑	Could be relevant for Tracker					☑	☐	☑	☑	☑
V6	Cryptography	6.1.3	Verify that regulated financial data is stored encrypted while at rest, such as financial accounts, defaults or credit history, tax records, pay history, beneficiaries, or de-anonymized market or research records.		X	X	☐	☐	☐	☑	☑			Dont deal with financial data.	Don't work with just health data, can be other forms of data. Implementation issue		☐	☐	☐	☑	☑
V6	Cryptography	6.2.1	Verify that all cryptographic modules fail securely, and errors are handled in a way that does not enable Padding Oracle attacks.	X	X	X	☑	☐	☐	☑	☑						☐	☐	☐	☐	☐
V6	Cryptography	6.2.2	Verify that industry proven or government approved cryptographic algorithms, modes, and libraries are used, instead of custom coded cryptography. [C8]		X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V6	Cryptography	6.2.3	Verify that encryption initialization vector, cipher configuration, and block modes are configured securely using the latest advice.		X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V6	Cryptography	6.2.4	Verify that random number, encryption or hashing algorithms, key lengths, rounds, ciphers or modes, can be reconfigured, upgraded, or swapped at any time, to protect against cryptographic breaks. [C8]		X	X	☑	☐	☑	☑	☑				Have the ability to reconfigure and swap		☐	☐	☐	☐	☐
V6	Cryptography	6.2.5	Verify that known insecure block modes (i.e. ECB, etc.), padding modes (i.e. PKCS#1 v1.5, etc.), ciphers with small block sizes (i.e. Triple-DES, Blowfish, etc.), and weak hashing algorithms (i.e. MD5, SHA1, etc.) are not used unless required for backwards compatibility.		X	X	☐	☐	☑	☑	☑						☐	☐	☐	☐	☐
V6	Cryptography	6.2.6	Verify that nonces, initialization vectors, and other single use numbers must not be used more than once with a given encryption key. The method of generation must be appropriate for the algorithm being used.		X	X	☐	☐	☑	☑	☑						☐	☐	☐	☐	☐
V6	Cryptography	6.2.7	Verify that encrypted data is authenticated via signatures, authenticated cipher modes, or HMAC to ensure that ciphertext is not altered by an unauthorized party.			X	☐	☐	☐	☐	☐			Dont do digital signatures. Dont do a lot of encryption inside DHIS.			☐	☐	☐	☐	☐
V6	Cryptography	6.2.8	Verify that all cryptographic operations are constant-time, with no 'short-circuit' operations in comparisons, calculations, or returns, to avoid leaking information.			X	☐	☐	☑	☑	☑						☐	☐	☐	☐	☐
V6	Cryptography	6.3.1	Verify that all random numbers, random file names, random GUIDs, and random strings are generated using the cryptographic module's approved cryptographically secure random number generator when these random values are intended to be not guessable by an attacker.		X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V6	Cryptography	6.3.2	Verify that random GUIDs are created using the GUID v4 algorithm, and a cryptographically-secure pseudo-random number generator (CSPRNG). GUIDs created using other pseudo-random number generators may be predictable.		X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V6	Cryptography	6.3.3	Verify that random numbers are created with proper entropy even when the application is under heavy load, or that the application degrades gracefully in such circumstances.			X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V6	Cryptography	6.4.1	Verify that a secrets management solution such as a key vault is used to securely create, store, control access to and destroy secrets. [C8]		X	X	☐	☐	☑	☑	☑						☐	☐	☐	☑	☑
V6	Cryptography	6.4.2	Verify that key material is not exposed to the application but instead uses an isolated security module like a vault for cryptographic operations. [C8]		X	X	☐	☐	☑	☑	☑			We dont do a lot of cryptograhpics inside the DHIS2 itself. A bit.	Implementation issue. Don't use keys.		☐	☐	☐	☑	☑
V7	Error	7.1.1	Verify that the application does not log credentials or payment details. Session tokens should only be stored in logs in an irreversible, hashed form. [C9, C10]	X	X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V7	Error	7.1.2	Verify that the application does not log other sensitive data as defined under local privacy laws or relevant security policy. [C9]	X	X	X	☑	☐	☑	☑	☑	Implemented to suit Norwegian laws, not international.					☑	☐	☐	☑	☑
V7	Error	7.1.3	Verify that the application logs security relevant events including successful and failed authentication events, access control failures, deserialization failures and input validation failures. [C5, C7]		X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V7	Error	7.1.4	Verify that each log event includes necessary information that would allow for a detailed investigation of the timeline when an event happens. [C9]		X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V7	Error	7.2.1	Verify that all authentication decisions are logged, without storing sensitive session identifiers or passwords. This should include requests with relevant metadata needed for security investigations.		X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V7	Error	7.2.2	Verify that all access control decisions can be logged and all failed decisions are logged. This should include requests with relevant metadata needed for security investigations.		X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐

			C1-C10 Top 10 Proactive Controls: https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering)				242/286	132/286	196/286	256/286	258/286						Total: 46	Total: 21	Total: 45	Total: 80	Total: 79
V7	Error	7.3.1	Verify that the application appropriately encodes user-supplied data to prevent log injection. [C9]		X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V7	Error	7.3.2	Verify that all events are protected from injection when viewed in log viewing software. [C9]		X	X	☑	☐	☐	☑	☑						☑	☐	☑	☑	☑
V7	Error	7.3.3	Verify that security logs are protected from unauthorized access and modification. [C9]		X	X	☑	☐	☐	☑	☑						☑	☐	☑	☑	☑
V7	Error	7.3.4	Verify that time sources are synchronized to the correct time and time zone. Strongly consider logging only in UTC if systems are global to assist with post-incident forensic analysis. [C9]		X	X	☑	☐	☑	☑	☑	The application in our case should log in UTC, as the server time sync is out of our control, which moves this to the implementors side if we do not log in UTC					☑	☐	☑	☑	☑
V7	Error	7.4.1	Verify that a generic message is shown when an unexpected or security sensitive error occurs, potentially with a unique ID which support personnel can use to investigate. [C10]	X	X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V7	Error	7.4.2	Verify that exception handling (or a functional equivalent) is used across the codebase to account for expected and unexpected error conditions. [C10]		X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V7	Error	7.4.3	Verify that a "last resort" error handler is defined which will catch all unhandled exceptions. [C10]		X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V8	Data	8.1.1	Verify the application protects sensitive data from being cached in server components such as load balancers and application caches.		X	X	☑	☐	☑	☑	☑				An implementation responsibility also. Don't have a way to specify things as sensitive.		☐	☐	☐	☑	☑
V8	Data	8.1.2	Verify that all cached or temporary copies of sensitive data stored on the server are protected from unauthorized access or purged/invalidated after the authorized user accesses the sensitive data.		X	X	☐	☐	☐	☐	☐						☑	☐	☑	☑	☑
V8	Data	8.1.3	Verify the application minimizes the number of parameters in a request, such as hidden fields, Ajax variables, cookies and header values.		X	X	☑	☐	☑	☑	☑						☑	☐	☐	☑	☑
V8	Data	8.1.4	Verify the application can detect and alert on abnormal numbers of requests, such as by IP, user, total per hour or day, or whatever makes sense for the application.		X	X	☑	☐	☑	☑	☑	This can be handled on the server level by implementors, but ideally the application should also be aware		Use proxy servers			☑	☐	☑	☑	☑
V8	Data	8.1.5	Verify that regular backups of important data are performed and that test restoration of data is performed.			X	☐	☐	☐	☐	☐						☑	☐	☑	☑	☑
V8	Data	8.1.6	Verify that backups are stored securely to prevent data from being stolen or corrupted.			X	☐	☐	☐	☐	☐						☑	☐	☑	☑	☑
V8	Data	8.2.1	Verify the application sets sufficient anti-caching headers so that sensitive data is not cached in modern browsers.	X	X	X	☑	☐	☑	☑	☑	If implementor is using a reverse proxy, it must allow the application to set these headers and not overwrite them.					☑	☐	☐	☑	☑
V8	Data	8.2.2	Verify that data stored in client side storage (such as HTML5 local storage, session storage, IndexedDB, regular cookies or Flash cookies) does not contain sensitive data or PII.	X	X	X	☑	☐	☑	☑	☑	Sensitive data depends on the national laws, so the core team can do a "best effort", but ultimate responsibility falls on the implementor.					☑	☐	☐	☑	☑
V8	Data	8.2.3	Verify that authenticated data is cleared from client storage, such as the browser DOM, after the client or session is terminated.	X	X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V8	Data	8.3.1	Verify that sensitive data is sent to the server in the HTTP message body or headers, and that query string parameters from any HTTP verb do not contain sensitive data.	X	X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V8	Data	8.3.2	Verify that users have a method to remove or export their data on demand.	X	X	X	☑	☐	☑	☑	☑	Diff in aggregated data and individual data					☐	☐	☐	☐	☐
V8	Data	8.3.3	Verify that users are provided clear language regarding collection and use of supplied personal information and that users have provided opt-in consent for the use of that data before it is used in any way.	X	X	X	☑	☐	☐	☑	☑						☐	☐	☑	☑	☑
V8	Data	8.3.4	Verify that all sensitive data created and processed by the application has been identified, and ensure that a policy is in place on how to deal with sensitive data. [C8]	X	X	X	☐	☐	☐	☐	☐	Policy depends on where the data is used, different per district/ministry/country.		Up to the juristiction on how to implement it			☑	☐	☑	☑	☑
V8	Data	8.3.5	Verify accessing sensitive data is audited (without logging the sensitive data itself), if the data is collected under relevant data protection directives or where logging of access is required.		X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V8	Data	8.3.6	Verify that sensitive information contained in memory is overwritten as soon as it is no longer required to mitigate memory dumping attacks, using zeroes or random data.		X	X	☑	☐	☑	☑	☑			Not a java thing			☐	☐	☐	☐	☐
V8	Data	8.3.7	Verify that sensitive or private information that is required to be encrypted, is encrypted using approved algorithms that provide both confidentiality and integrity. [C8]		X	X	☑	☐	☑	☑	☑						☐	☐	☑	☐	☐

			C1-C10 Top 10 Proactive Controls: https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering)				242/286	132/286	196/286	256/286	258/286						Total: 46	Total: 21	Total: 45	Total: 80	Total: 79
V8	Data	8.3.8	Verify that sensitive personal information is subject to data retention classification, such that old or out of date data is deleted automatically, on a schedule, or as the situation requires.		X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V9	Communications	9.1.1	Verify that secured TLS is used for all client connectivity, and does not fall back to insecure or unencrypted protocols. [C8]	X	X	X	☑	☐	☐	☑	☑						☐	☐	☑	☑	☑
V9	Communications	9.1.2	Verify using online or up to date TLS testing tools that only strong algorithms, ciphers, and protocols are enabled, with the strongest algorithms and ciphers set as preferred.	X	X	X	☑	☐	☐	☑	☑						☐	☐	☑	☑	☑
V9	Communications	9.1.3	Verify that old versions of SSL and TLS protocols, algorithms, ciphers, and configuration are disabled, such as SSLv2, SSLv3, or TLS 1.0 and TLS 1.1. The latest version of TLS should be the preferred cipher suite.	X	X	X	☑	☐	☐	☑	☑						☐	☐	☐	☐	☐
V9	Communications	9.2.1	Verify that connections to and from the server use trusted TLS certificates. Where internally generated or self-signed certificates are used, the server must be configured to only trust specific internal CAs and specific self-signed certificates. All others should be rejected.		X	X	☑	☐	☐	☑	☑						☐	☐	☑	☑	☑
V9	Communications	9.2.2	Verify that encrypted communications such as TLS is used for all inbound and outbound connections, including for management ports, monitoring, authentication, API, or web service calls, database, cloud, serverless, mainframe, external, and partner connections. The server must not fall back to insecure or unencrypted protocols.		X	X	☑	☐	☐	☑	☑						☐	☐	☑	☑	☑
V9	Communications	9.2.3	Verify that all encrypted connections to external systems that involve sensitive information or functions are authenticated.		X	X	☑	☐	☐	☑	☑						☐	☐	☑	☑	☑
V9	Communications	9.2.4	Verify that proper certification revocation, such as Online Certificate Status Protocol (OCSP) Stapling, is enabled and configured.		X	X	☐	☐	☐	☐	☐	Unsure, but I think this is done by implementors on a server level					☑	☐	☑	☑	☑
V9	Communications	9.2.5	Verify that backend TLS connection failures are logged.			X	☑	☐	☐	☑	☑						☐	☐	☑	☑	☑
V10	Malicious	10.1.1	Verify that a code analysis tool is in use that can detect potentially malicious code, such as time functions, unsafe file operations and network connections.			X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V10	Malicious	10.2.1	Verify that the application source code and third party libraries do not contain unauthorized phone home or data collection capabilities. Where such functionality exists, obtain the user's permission for it to operate before collecting any data.		X	X	☑	☐	☑	☑	☑						☐	☐	☐	☑	☑
V10	Malicious	10.2.2	Verify that the application does not ask for unnecessary or excessive permissions to privacy related features or sensors, such as contacts, cameras, microphones, or location.		X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V10	Malicious	10.2.3	Verify that the application source code and third party libraries do not contain back doors, such as hard-coded or additional undocumented accounts or keys, code obfuscation, undocumented binary blobs, rootkits, or anti-debugging, insecure debugging features, or otherwise out of date, insecure, or hidden functionality that could be used maliciously if discovered.			X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V10	Malicious	10.2.4	Verify that the application source code and third party libraries does not contain time bombs by searching for date and time related functions.			X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V10	Malicious	10.2.5	Verify that the application source code and third party libraries does not contain malicious code, such as salami attacks, logic bypasses, or logic bombs.			X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V10	Malicious	10.2.6	Verify that the application source code and third party libraries do not contain Easter eggs or any other potentially unwanted functionality.			X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V10	Malicious	10.3.1	Verify that if the application has a client or server auto-update feature, updates should be obtained over secure channels and digitally signed. The update code must validate the digital signature of the update before installing or executing the update.	X	X	X	☑	☐	☑	☑	☑			When you download the DHIS2 .war file, there is no hash code on it, to verifiy is the code is actually valid. Could cause a MITM attack.			☐	☐	☐	☐	☐
V10	Malicious	10.3.2	Verify that the application employs integrity protections, such as code signing or sub-resource integrity. The application must not load or execute code from untrusted sources, such as loading includes, modules, plugins, code, or libraries from untrusted sources or the Internet.	X	X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V10	Malicious	10.3.3	Verify that the application has protection from sub-domain takeovers if the application relies upon DNS entries or DNS sub-domains, such as expired domain names, out of date DNS pointers or CNAMEs, expired projects at public source code repos, or transient cloud APIs, serverless functions, or storage buckets (autogen-bucket-id.cloud.example.com) or similar. Protections can include ensuring that DNS names used by applications are regularly checked for expiry or change.	X	X	X	☑	☐	☑	☑	☑						☐	☐	☐	☑	☑
V11	BusLogic	11.1.1	Verify the application will only process business logic flows for the same user in sequential step order and without skipping steps.	X	X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V11	BusLogic	11.1.2	Verify the application will only process business logic flows with all steps being processed in realistic human time, i.e. transactions are not submitted too quickly.	X	X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V11	BusLogic	11.1.3	Verify the application has appropriate limits for specific business actions or transactions which are correctly enforced on a per user basis.	X	X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V11	BusLogic	11.1.4	Verify the application has sufficient anti-automation controls to detect and protect against data exfiltration, excessive business logic requests, excessive file uploads or denial of service attacks.	X	X	X	☑	☐	☑	☑	☑	E.g. file uploads are restricted through the reverse proxy.					☑	☐	☐	☑	☑
V11	BusLogic	11.1.5	Verify the application has business logic limits or validation to protect against likely business risks or threats, identified using threat modelling or similar methodologies.	X	X	X	☐	☐	☑	☑	☑						☐	☐	☐	☐	☐

			C1-C10 Top 10 Proactive Controls: https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering)				242/286	132/286	196/286	256/286	258/286						Total: 46	Total: 21	Total: 45	Total: 80	Total: 79
V11	BusLogic	11.1.6	Verify the application does not suffer from "time of check to time of use" (TOCTOU) issues or other race conditions for sensitive operations.		X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V11	BusLogic	11.1.7	Verify the application monitors for unusual events or activity from a business logic perspective. For example, attempts to perform actions out of order or actions which a normal user would never attempt. [C9]		X	X	☑	☐	☑	☐	☐	Could be interesting for e.g. ProgamStages and Tracker programs.					☐	☐	☐	☐	☐
V11	BusLogic	11.1.8	Verify the application has configurable alerting when automated attacks or unusual activity is detected.		X	X	☑	☐	☑	☑	☑						☐	☐	☑	☑	☑
V12	Files	12.1.1	Verify that the application will not accept large files that could fill up storage or cause a denial of service attack.	X	X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V12	Files	12.1.2	Verify that compressed files are checked for "zip bombs" - small input files that will decompress into huge files thus exhausting file storage limits.		X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V12	Files	12.1.3	Verify that a file size quota and maximum number of files per user is enforced to ensure that a single user cannot fill up the storage with too many files, or excessively large files.		X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V12	Files	12.2.1	Verify that files obtained from untrusted sources are validated to be of expected type based on the file's content.		X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V12	Files	12.3.1	Verify that user-submitted filename metadata is not used directly with system or framework file and URL API to protect against path traversal.	X	X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V12	Files	12.3.2	Verify that user-submitted filename metadata is validated or ignored to prevent the disclosure, creation, updating or removal of local files (LFI).	X	X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V12	Files	12.3.3	Verify that user-submitted filename metadata is validated or ignored to prevent the disclosure or execution of remote files (RFI), which may also lead to SSRF.	X	X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V12	Files	12.3.4	Verify that the application protects against reflective file download (RFD) by validating or ignoring user-submitted filenames in a JSON, JSONP, or URL parameter, the response Content-Type header should be set to text/plain, and the Content-Disposition header should have a fixed filename.	X	X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V12	Files	12.3.5	Verify that untrusted file metadata is not used directly with system API or libraries, to protect against OS command injection.	X	X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V12	Files	12.3.6	Verify that the application does not include and execute functionality from untrusted sources, such as unverified content distribution networks, JavaScript libraries, node npm libraries, or server-side DLLs.		X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V12	Files	12.4.1	Verify that files obtained from untrusted sources are stored outside the web root, with limited permissions, preferably with strong validation.	X	X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V12	Files	12.4.2	Verify that files obtained from untrusted sources are scanned by antivirus scanners to prevent upload of known malicious content.	X	X	X	☑	☐	☑	☑	☑	Implementor would have to run a virus scan on the files		Can send a word document with malicious content in it and other people can download it.			☑	☐	☑	☑	☑
V12	Files	12.5.1	Verify that the web tier is configured to serve only files with specific file extensions to prevent unintentional information and source code leakage. For example, backup files (e.g. .bak), temporary working files (e.g. .swp), compressed files (.zip, .tar.gz, etc) and other extensions commonly used by editors should be blocked unless required.	X	X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V12	Files	12.5.2	Verify that direct requests to uploaded files will never be executed as HTML/JavaScript content.	X	X	X	☑	☐	☑	☑	☑						☐	☐	☐	☐	☐
V12	Files	12.6.1	Verify that the web or application server is configured with a whitelist of resources or systems to which the server can send requests or load data/files from.	X	X	X	☑	☐	☑	☑	☑				impl. server configuration		☐	☐	☐	☑	☑
V13	API	13.1.1	Verify that all application components use the same encodings and parsers to avoid parsing attacks that exploit different URI or file parsing behavior that could be used in SSRF and RFI attacks.	X	X	X	☑	☑	☑	☑	☑						☐	☐	☐	☐	☐
V13	API	13.1.2	Verify that access to administration and management functions is limited to authorized administrators.	X	X	X	☑	☑	☑	☑	☑						☑	☐	☐	☑	☑
V13	API	13.1.3	Verify API URLs do not expose sensitive information, such as the API key, session tokens etc.	X	X	X	☑	☑	☐	☑	☑						☐	☐	☐	☐	☐
V13	API	13.1.4	Verify that authorization decisions are made at both the URI, enforced by programmatic or declarative security at the controller or router, and at the resource level, enforced by model-based permissions.		X	X	☑	☑	☐	☑	☑						☐	☐	☐	☐	☐
V13	API	13.1.5	Verify that requests containing unexpected or missing content types are rejected with appropriate headers (HTTP response status 406 Unacceptable or 415 Unsupported Media Type).		X	X	☑	☑	☐	☑	☑						☐	☐	☐	☐	☐
V13	API	13.2.1	Verify that enabled RESTful HTTP methods are a valid choice for the user or action, such as preventing normal users using DELETE or PUT on protected API or resources.	X	X	X	☑	☑	☐	☑	☑						☐	☐	☐	☐	☐
V13	API	13.2.2	Verify that JSON schema validation is in place and verified before accepting input.	X	X	X	☑	☑	☐	☑	☑						☐	☐	☐	☐	☐
V13	API	13.2.3	Verify that RESTful web services that utilize cookies are protected from Cross-Site Request Forgery via the use of at least one or more of the following: triple or double submit cookie pattern (see https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet), CSRF nonces, or ORIGIN request header checks.	X	X	X	☑	☑	☐	☑	☑		Very applicable, hot topic, raised by several security groups.				☐	☐	☐	☐	☐
V13	API	13.2.4	Verify that REST services have anti-automation controls to protect against excessive calls, especially if the API is unauthenticated.		X	X	☑	☑	☐	☑	☑	Especially important for the App Store backend API as it is publicly available without auth.					☐	☐	☐	☐	☐

			C1-C10 Top 10 Proactive Controls: https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering)				242/286	132/286	196/286	256/286	258/286						Total: 46	Total: 21	Total: 45	Total: 80	Total: 79
V13	API	13.2.5	Verify that REST services explicitly check the incoming Content-Type to be the expected one, such as application/xml or application/JSON.		X	X	☑	☑	☐	☑	☑						☐	☐	☐	☐	☐
V13	API	13.2.6	Verify that the message headers and payload are trustworthy and not modified in transit. Requiring strong encryption for transport (TLS only) may be sufficient in many cases as it provides both confidentiality and integrity protection. Per-message digital signatures can provide additional assurance on top of the transport protections for high-security applications but bring with them additional complexity and risks to weigh against the benefits.		X	X	☑	☑	☐	☑	☑						☐	☐	☐	☐	☐
V13	API	13.3.1	Verify that XSD schema validation takes place to ensure a properly formed XML document, followed by validation of each input field before any processing of that data takes place.	X	X	X	☑	☑	☐	☑	☑						☐	☐	☐	☐	☐
V13	API	13.3.2	Verify that the message payload is signed using WS-Security to ensure reliable transport between client and service.		X	X	☑	☐	☐	☑	☑		Not applicable if SOAP isn't used.				☐	☐	☐	☐	☐
V13	API	13.4.1	Verify that query whitelisting or a combination of depth limiting and amount limiting should be used to prevent GraphQL or data layer expression denial of service (DoS) as a result of expensive, nested queries. For more advanced scenarios, query cost analysis should be used.		X	X	☑	☑	☐	☑	☑	We don't use GraphQL but DHIS2 has a custom dynamic filter language similar to it which can lead to heavy queries so this applies.					☐	☐	☐	☐	☐
V13	API	13.4.2	Verify that GraphQL or other data layer authorization logic should be implemented at the business logic layer instead of the GraphQL layer.		X	X	☑	☐	☐	☑	☑						☐	☐	☐	☐	☐
V14	Config	14.1.1	Verify that the application build and deployment processes are performed in a secure and repeatable way, such as CI / CD automation, automated configuration management, and automated deployment scripts.		X	X	☑	☑	☐	☑	☑		DHIS is not responsible how it is deployed within the environment, only responsible for how the process should be done. Need to be deployed securely by the implementors.				☐	☐	☐	☐	☐
V14	Config	14.1.2	Verify that compiler flags are configured to enable all available buffer overflow protections and warnings, including stack randomization, data execution prevention, and to break the build if an unsafe pointer, memory, format string, integer, or string operations are found.		X	X	☑	☑	☐	☑	☑						☐	☐	☐	☐	☐
V14	Config	14.1.3	Verify that server configuration is hardened as per the recommendations of the application server and frameworks in use.		X	X	☐	☐	☐	☐	☐						☑	☑	☐	☑	☑
V14	Config	14.1.4	Verify that the application, configuration, and all dependencies can be re-deployed using automated deployment scripts, built from a documented and tested runbook in a reasonable time, or restored from backups in a timely fashion.		X	X	☑	☐	☐	☑	☑		Only do deyployment scripts for tesitng. Just provide the . war file and they can deploy it themselves and do what they want.				☑	☑	☐	☑	☑
V14	Config	14.1.5	Verify that authorized administrators can verify the integrity of all security-relevant configurations to detect tampering.			X	☑	☑	☐	☑	☑		Some is not internally, some if some externally. Some auditing is done by the system.				☑	☑	☐	☑	☑
V14	Config	14.2.1	Verify that all components are up to date, preferably using a dependency checker during build or compile time. [C2]	X	X	X	☑	☑	☐	☑	☑						☐	☐	☐	☐	☐
V14	Config	14.2.2	Verify that all unneeded features, documentation, samples, configurations are removed, such as sample applications, platform documentation, and default or example users.	X	X	X	☑	☑	☐	☑	☑						☐	☐	☐	☐	☐
V14	Config	14.2.3	Verify that if application assets, such as JavaScript libraries, CSS stylesheets or web fonts, are hosted externally on a content delivery network (CDN) or external provider, Subresource Integrity (SRI) is used to validate the integrity of the asset.	X	X	X	☐	☑	☐	☐	☐	Currently hosted by the application itself, but will be relevant in the future.					☐	☐	☐	☐	☐
V14	Config	14.2.4	Verify that third party components come from pre-defined, trusted and continually maintained repositories. [C2]		X	X	☑	☑	☐	☑	☑						☐	☐	☐	☐	☐
V14	Config	14.2.5	Verify that an inventory catalog is maintained of all third party libraries in use. [C2]		X	X	☑	☑	☐	☑	☑						☐	☐	☐	☐	☐
V14	Config	14.2.6	Verify that the attack surface is reduced by sandboxing or encapsulating third party libraries to expose only the required behaviour into the application. ([C2] (https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering))		X	X	☑	☑	☐	☑	☑						☐	☐	☐	☐	☐
V14	Config	14.3.1	Verify that web or application server and framework error messages are configured to deliver user actionable, customized responses to eliminate any unintended security disclosures.	X	X	X	☑	☑	☐	☑	☑						☐	☐	☐	☐	☐
V14	Config	14.3.2	Verify that web or application server and application framework debug modes are disabled in production to eliminate debug features, developer consoles, and unintended security disclosures.	X	X	X	☑	☑	☐	☑	☑					Implementation responsbility. They have the posiblity to run debug opeations, but its not activated by default.	☐	☐	☐	☑	☑
V14	Config	14.3.3	Verify that the HTTP headers or any part of the HTTP response do not expose detailed version information of system components.	X	X	X	☑	☑	☐	☑	☑						☐	☐	☐	☐	☐

			C1-C10 Top 10 Proactive Controls: https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering)				242/286	132/286	196/286	256/286	258/286						Total: 46	Total: 21	Total: 45	Total: 80	Total: 79
V14	Config	14.4.1	Verify that every HTTP response contains a content type header specifying a safe character set (e.g., UTF-8, ISO 8859-1).	X	X	X	☑	☑	☐	☑	☑						☐	☐	☐	☐	☐
V14	Config	14.4.2	Verify that all API responses contain Content-Disposition: attachment; filename="api.json" (or other appropriate filename for the content type).	X	X	X	☑	☑	☐	☑	☑						☐	☐	☐	☐	☐
V14	Config	14.4.3	Verify that a content security policy (CSPv2) is in place that helps mitigate impact for XSS attacks like HTML, DOM, JSON, and JavaScript injection vulnerabilities.	X	X	X	☑	☑	☐	☑	☑						☐	☐	☐	☐	☐
V14	Config	14.4.4	Verify that all responses contain X-Content-Type-Options: nosniff.	X	X	X	☑	☑	☐	☑	☑						☐	☐	☐	☐	☐
V14	Config	14.4.5	Verify that HTTP Strict Transport Security headers are included on all responses and for all subdomains, such as Strict-Transport-Security: max-age=15724800; includeSubdomains.	X	X	X	☑	☑	☐	☑	☑						☐	☐	☐	☐	☐
V14	Config	14.4.6	Verify that a suitable "Referrer-Policy" header is included, such as "no-referrer" or "same-origin".	X	X	X	☑	☑	☐	☑	☑						☐	☐	☐	☐	☐
V14	Config	14.4.7	Verify that a suitable X-Frame-Options or Content-Security-Policy: frame-ancestors header is in use for sites where content should not be embedded in a third-party site.	X	X	X	☑	☑	☐	☑	☑						☐	☐	☐	☐	☐
V14	Config	14.5.1	Verify that the application server only accepts the HTTP methods in use by the application or API, including pre-flight OPTIONS.	X	X	X	☑	☑	☐	☑	☑						☐	☐	☐	☐	☐
V14	Config	14.5.2	Verify that the supplied Origin header is not used for authentication or access control decisions, as the Origin header can easily be changed by an attacker.	X	X	X	☑	☑	☐	☑	☑						☐	☐	☐	☐	☐
V14	Config	14.5.3	Verify that the cross-domain resource sharing (CORS) Access-Control-Allow-Origin header uses a strict white-list of trusted domains to match against and does not support the "null" origin.	X	X	X	☑	☑	☐	☑	☑	The CORS whitelist is controlled by the implementor or admin of the instance.					☑	☐	☐	☑	☑
V14	Config	14.5.4	Verify that HTTP headers added by a trusted proxy or SSO devices, such as a bearer token, are authenticated by the application.		X	X	☑	☑	☐	☑	☑						☐	☐	☐	☐	☐

Phase 2: Scoring spreadsheet

This spreadsheet shows the results from the scoring phase. To explain the spreadsheet:

- A failed control is marked red.
- A passed control is marked green.
- A control with conflicting results (a pass and fail) from the participants is marked yellow.
- A control that was skipped by the participants is marked pink.

Three people participated in this phase. All three responses are shown side-by-side for comparisson. To reiterate from the Methods chapter: a control is a pass or a fail if selected as such by the majority, or if passed by the other participants.

For each participant, a counter shows the number of passed controls and the total number of applicable controls. The frontend developer, for example, gave 122 out of 258 controls a pass.

			C1-C10 Top 10 Proactive Controls: https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering)						258/286	122/258	146/258	4/258	174/258					79
			ASVS 4.0							Frontend	Backend	Security E.	Combined		Frontend	Backend	Security E.	
Section	**Name**	**Item**	**Description**	**L1**	**L2**	**L3**	**CWE**	**NIST**	**Applicable**	**Pass**	**Pass**	**Pass**	**Pass**		**Comments**	**Comments**	**Comments**	**Impl.**
V1	Architecture	1.1.1	Verify the use of a secure software development lifecycle that addresses security in all stages of development. [C1]		X	X			☑	☐	☐	☐	☐			The control is failed from the point of view of a standarized process. There is a peer-review process however, and certain security considerations taken into in before development.		☐
V1	Architecture	1.1.2	Verify the use of threat modeling for every design change or sprint planning to identify threats, plan for countermeasures, facilitate appropriate risk responses, and guide security testing.		X	X	1053		☑	☐	☐	☐	☐			No formal process for this.		☐
V1	Architecture	1.1.3	Verify that all user stories and features contain functional security constraints, such as "As a user, I should be able to view and edit my profile. I should not be able to view or edit anyone else's profile"		X	X	1110		☑	☐	☑	☐	☑					☐
V1	Architecture	1.1.4	Verify documentation and justification of all the application's trust boundaries, components, and significant data flows.		X	X	1059		☑	☐	☑	☐	☑			Continuesly updated as things are altered in the system.		☐
V1	Architecture	1.1.5	Verify definition and security analysis of the application's high-level architecture and all connected remote services. [C1]		X	X	1059		☑	☐	☐	☐	☐			The user can opt to use external services such as Amazon cloud. It is part implementation, in that they are themselves in charge of securing it, but from DHIS2s standpoint, if they are aple to secure it, then they do		☑
V1	Architecture	1.1.6	Verify implementation of centralized, simple (economy of design), vetted, secure, and reusable security controls to avoid duplicate, missing, ineffective, or insecure controls. [C10]		X	X	637		☑	☑	☑	☐	☑					☐
V1	Architecture	1.1.7	Verify availability of a secure coding checklist, security requirements, guideline, or policy to all developers and testers.		X	X	637		☑	☐	☐	☐	☐			No formalized guidelines or checklist		☐
V1	Architecture	1.2.2	Verify that communications between application components, including APIs, middleware and data layers, are authenticated. Components should have the least necessary privileges needed. [C3]		X	X	306		☑	☐	☑	☐	☑			Have more API tests that test authentication and access control for the users, based on how the system is configured.		☑
V1	Architecture	1.2.3	Verify that the application uses a single vetted authentication mechanism that is known to be secure, can be extended to include strong authentication, and has sufficient logging and monitoring to detect account abuse or breaches.		X	X	306		☑	☐	☐	☐	☐			Also have an oauth setup in addition to username/password. Have protection against flooding of authentiation, i.e. brute force attack.		☑
V1	Architecture	1.2.4	Verify that all authentication pathways and identity management APIs implement consistent authentication security control strength, such that there are no weaker alternatives per the risk of the application.		X	X	306		☑	☑	☑	☐	☑					☐
V1	Architecture	1.4.1	Verify that trusted enforcement points such as at access control gateways, servers, and serverless functions enforce access controls. Never enforce access controls on the client.		X	X	602		☑	☑	☐	☑	☑				This is covered on the server-side.	☑
V1	Architecture	1.4.2	Verify that the chosen access control solution is flexible enough to meet the application's needs.		X	X	284		☑	☑	☑	☐	☑					☐
V1	Architecture	1.4.3	Verify enforcement of the principle of least privilege in functions, data files, URLs, controllers, services, and other resources. This implies protection against spoofing and elevation of privilege.		X	X	272		☑	☑	☑	☐	☑			The system is in place, but the implementors configure it. Have the ability to configure the access to these things, but don't have any setup to hinder spoofing, for example. In the case of elevation of priviledge, you cannot give more access than what the user has.		☑
V1	Architecture	1.4.4	Verify the application uses a single and well-vetted access control mechanism for accessing protected data and resources. All requests must pass through this single mechanism to avoid copy and paste or insecure alternative paths. [C7]		X	X	284		☑	☑	☑	☐	☑					☐
V1	Architecture	1.4.5	Verify that attribute or feature-based access control is used whereby the code checks the user's authorization for a feature/data item rather than just their role. Permissions should still be allocated using roles. [C7]		X	X	275		☑	☑	☐	☐	☐			We don't authorize when we check this type of access control; only check permissions.		☐
V1	Architecture	1.5.1	Verify that input and output requirements clearly define how to handle and process data based on type, content, and applicable laws, regulations, and other policy compliance.		X	X	1029		☑	☐	☑	☐	☑			Have support for this, but is an implementation issue		☑
V1	Architecture	1.5.2	Verify that serialization is not used when communicating with untrusted clients. If this is not possible, ensure that adequate integrity controls (and possibly encryption if sensitive data is sent) are enforced to prevent deserialization attacks including object injection.		X	X	502		☑	☐	☑	☐	☑					☑
V1	Architecture	1.5.3	Verify that input validation is enforced on a trusted service layer. [C5]		X	X	602		☑	☑	☑	☐	☑					☐
V1	Architecture	1.5.4	Verify that output encoding occurs close to or by the interpreter for which it is intended. [C4]		X	X	116		☑	☑	☑	☐	☑					☐
V1	Architecture	1.6.2	Verify that consumers of cryptographic services protect key material and other secrets by using key vaults or API based alternatives.		X	X	320		☑	☐	☐	☐	☐			Pass on their end, but up to the implementors to handle it. Doesnt involve the backend.		☑
V1	Architecture	1.6.3	Verify that all keys and passwords are replaceable and are part of a well-defined process to re-encrypt sensitive data.		X	X	320		☑	☐	☐	☐	☐				Task to fix this is on my todo list. Issue is created and tracked.	☑

			C1-C10 Top 10 Proactive Controls: https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering)						258/286	122/258	146/258	4/258	174/258				79
V1	Architecture	1.6.4	Verify that symmetric keys, passwords, or API secrets generated by or shared with clients are used only in protecting low risk secrets, such as encrypting local storage, or temporary ephemeral uses such as parameter obfuscation. Sharing secrets with clients is clear-text equivalent and architecturally should be treated as such.		X	X	320		☑	☐	☑	☐	☐				☑
V1	Architecture	1.7.1	Verify that a common logging format and approach is used across the system. [C9]		X	X	1009		☑	☐	☑	☐	☑				☑
V1	Architecture	1.7.2	Verify that logs are securely transmitted to a preferably remote system for analysis, detection, alerting, and escalation. [C9]		X	X			☑	☐	☑	☐	☐			Lacking tools/config to send logs to remote storage	☑
V1	Architecture	1.8.1	Verify that all sensitive data is identified and classified into protection levels.		X	X			☑	☐	☑	☐	☑				☑
V1	Architecture	1.8.2	Verify that all protection levels have an associated set of protection requirements, such as encryption requirements, integrity requirements, retention, privacy and other confidentiality requirements, and that these are applied in the architecture.		X	X			☑	☐	☑	☐	☑				☑
V1	Architecture	1.9.1	Verify the application encrypts communications between components, particularly when these components are in different containers, systems, sites, or cloud providers. [C3]		X	X	319		☑	☐	☑	☐	☑		Support for using https and ssl, but also implementation		☑
V1	Architecture	1.9.2	Verify that application components verify the authenticity of each side in a communication link to prevent person-in-the-middle attacks. For example, application components should validate TLS certificates and chains.		X	X	295		☑	☐	☐	☐	☐		Have support for TLS, but unsure if they validate certificates		☑
V1	Architecture	1.10.1	Verify that a source code control system is in use, with procedures to ensure that check-ins are accompanied by issues or change tickets. The source code control system should have access control and identifiable users to allow traceability of any changes.		X	X	284		☑	☑	☑	☐	☑				☐
V1	Architecture	1.11.1	Verify the definition and documentation of all application components in terms of the business or security functions they provide.		X	X	1059		☑	☑	☑	☐	☑				☐
V1	Architecture	1.11.2	Verify that all high-value business logic flows, including authentication, session management and access control, do not share unsynchronized state.		X	X	362		☑	☑	☑	☐	☑				☐
V1	Architecture	1.11.3	Verify that all high-value business logic flows, including authentication, session management and access control are thread safe and resistant to time-of-check and time-of-use race conditions.			X	367		☑	☑	☑	☐	☑		Dont have race conditions posiblities in access controls. Everything is done in one thread.		☐
V1	Architecture	1.12.1	Verify that user-uploaded files are stored outside of the web root.		X	X	552		☑	☐	☑	☐	☑				☑
V1	Architecture	1.12.2	Verify that user-uploaded files - if required to be displayed or downloaded from the application - are served by either octet stream downloads, or from an unrelated domain, such as a cloud file storage bucket. Implement a suitable content security policy to reduce the risk from XSS vectors or other attacks from the uploaded file.		X	X	646		☑	☑	☐	☐	☐		Part pass, part fail		☐
V1	Architecture	1.14.3	Verify that the build pipeline warns of out-of-date or insecure components and takes appropriate actions.		X	X	1104		☑	☑	☑	☐	☑				☐
V1	Architecture	1.14.4	Verify that the build pipeline contains a build step to automatically build and verify the secure deployment of the application, particularly if the application infrastructure is software defined, such as cloud environment build scripts.		X	X			☑	☑	☑	☐	☑				☐
V1	Architecture	1.14.6	Verify the application does not use unsupported, insecure, or deprecated client-side technologies such as NSAPI plugins, Flash, Shockwave, ActiveX, Silverlight, NACL, or client-side Java applets.		X	X	477		☑	☑	☐	☐	☐		Do not use any of these things, but have client technologies that they know are insequece, but that they are working on securing	Not sure, front-end only issue	☐
V2	Authentication	2.1.1	Verify that user set passwords are at least 12 characters in length. [C6]	X	X	X	521	5.1.1.2	☑	☐	☑	☑	☑		You can configure it to be 12 characters.		☐
V2	Authentication	2.1.2	Verify that passwords 64 characters or longer are permitted. [C6]	X	X	X	521	5.1.1.2	☑	☑	☑	☐	☑				☐
V2	Authentication	2.1.3	Verify that passwords can contain spaces and truncation is not performed. Consecutive multiple spaces MAY optionally be coalesced. [C6]	X	X	X	521	5.1.1.2	☑	☑	☑	☐	☑				☐
V2	Authentication	2.1.4	Verify that Unicode characters are permitted in passwords. A single Unicode code point is considered a character, so 12 emoji or 64 kanji characters should be valid and permitted.	X	X	X	521	5.1.1.2	☑	☑	☑	☐	☑				☐
V2	Authentication	2.1.5	Verify users can change their password.	X	X	X	620	5.1.1.2	☑	☑	☑	☐	☑				☐
V2	Authentication	2.1.6	Verify that password change functionality requires the user's current and new password.	X	X	X	620	5.1.1.2	☑	☑	☑	☐	☑				☐
V2	Authentication	2.1.7	Verify that passwords submitted during account registration, login, and password change are checked against a set of breached passwords either locally (such as the top 1,000 or 10,000 most common passwords which match the system's password policy) or using an external API. If using an API a zero knowledge proof or other mechanism should be used to ensure that the plain text password is not sent or used in verifying the breach status of the password. If the password is breached, the application must require the user to set a new non-breached password. [C6]	X	X	X	521	5.1.1.2	☑	☐	☐	☐	☐				☐
V2	Authentication	2.1.8	Verify that a password strength meter is provided to help users set a stronger password.	X	X	X	521	5.1.1.2	☑	☐	☐	☐	☐				☐

			C1-C10 Top 10 Proactive Controls: https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering)						258/286	122/258	146/258	4/258	174/258				79
V2	Authentication	2.1.9	Verify that there are no password composition rules limiting the type of characters permitted. There should be no requirement for upper or lower case or numbers or special characters. [C6]	X	X	X	521	5.1.1.2	☑	☐	☐	☐	☐		Should have the possiblity to remove the configuration in the future		☐
V2	Authentication	2.1.10	Verify that there are no periodic credential rotation or password history requirements.	X	X	X	263	5.1.1.2	☑	☐	☑	☐	☑	Should be fail according to: https://docs.dhis2.org/master/en/user/html/dhis2_user_manual_en_full.html#password-requirements	Can opt out		☑
V2	Authentication	2.1.11	Verify that "paste" functionality, browser password helpers, and external password managers are permitted.	X	X	X	521	5.1.1.2	☑	☑	☐	☐	☑				☐
V2	Authentication	2.1.12	Verify that the user can choose to either temporarily view the entire masked password, or temporarily view the last typed character of the password on platforms that do not have this as native functionality.	X	X	X	521	5.1.1.2	☑	☐	☐	☐	☐				☐
V2	Authentication	2.2.1	Verify that anti-automation controls are effective at mitigating breached credential testing, brute force, and account lockout attacks. Such controls include blocking the most common breached passwords, soft lockouts, rate limiting, CAPTCHA, ever increasing delays between attempts, IP address restrictions, or risk-based restrictions such as location, first login on a device, recent attempts to unlock the account, or similar. Verify that no more than 100 failed attempts per hour is possible on a single account.	X	X	X	307	5.2.2 / 5.1.1.2 / 5.1.4.2 / 5.1.5.2	☑	☐	☑	☐	☑		Have possiblity to opt-in and use CAPTCHA		☐
V2	Authentication	2.2.2	Verify that the use of weak authenticators (such as SMS and email) is limited to secondary verification and transaction approval and not as a replacement for more secure authentication methods. Verify that stronger methods are offered before weak methods, users are aware of the risks, or that proper measures are in place to limit the risks of account compromise.	X	X	X	304	5.2.10	☑	☐	☑	☐	☑				☐
V2	Authentication	2.2.3	Verify that secure notifications are sent to users after updates to authentication details, such as credential resets, email or address changes, logging in from unknown or risky locations. The use of push notifications - rather than SMS or email - is preferred, but in the absence of push notifications, SMS or email is acceptable as long as no sensitive information is disclosed in the notification.	X	X	X	620		☑	☐	☐	☐	☐				☐
V2	Authentication	2.2.4	Verify impersonation resistance against phishing, such as the use of multi-factor authentication, cryptographic devices with intent (such as connected keys with a push to authenticate), or at higher AAL levels, client-side certificates.			X	308	5.2.5	☑	☐	☑	☐	☑		Have the possiblity to enale multi-factor.		☑
V2	Authentication	2.2.5	Verify that where a credential service provider (CSP) and the application verifying authentication are separated, mutually authenticated TLS is in place between the two endpoints.			X	319	5.2.6	☑	☐	☑	☐	☑				☑
V2	Authentication	2.2.7	Verify intent to authenticate by requiring the entry of an OTP token or user-initiated action such as a button press on a FIDO hardware key.			X	308	5.2.9	☑	☐	☑	☐	☑				☐
V2	Authentication	2.3.1	Verify system generated initial passwords or activation codes SHOULD be securely randomly generated, SHOULD be at least 6 characters long, and MAY contain letters and numbers, and expire after a short period of time. These initial secrets must not be permitted to become the long term password.	X	X	X	330	5.1.1.2 / A.3	☑	☐	☐	☐	☐		Don't have system generated passwords, but unsure		☐
V2	Authentication	2.3.2	Verify that enrollment and use of subscriber-provided authentication devices are supported, such as a U2F or FIDO tokens.		X	X	308	6.1.3	☑	☐	☐	☐	☐				☐
V2	Authentication	2.3.3	Verify that renewal instructions are sent with sufficient time to renew time bound authenticators.		X	X	287	6.1.4	☑	☐	☐	☐	☐				☐
V2	Authentication	2.4.1	Verify that passwords are stored in a form that is resistant to offline attacks. Passwords SHALL be salted and hashed using an approved one-way key derivation or password hashing function. Key derivation and password hashing functions take a password, a salt, and a cost factor as inputs when generating a password hash. [C6]		X	X	916	5.1.1.2	☑	☑	☑	☐	☑				☐
V2	Authentication	2.4.2	Verify that the salt is at least 32 bits in length and be chosen arbitrarily to minimize salt value collisions among stored hashes. For each credential, a unique salt value and the resulting hash SHALL be stored. [C6]		X	X	916	5.1.1.2	☑	☑	☑	☐	☑		We use a different framework, Sprint Security. Don't do the last bit.		☐
V2	Authentication	2.4.3	Verify that if PBKDF2 is used, the iteration count SHOULD be as large as verification server performance will allow, typically at least 100,000 iterations. [C6]		X	X	916	5.1.1.2	☑	☑	☐	☐	☐				☐
V2	Authentication	2.4.4	Verify that if bcrypt is used, the work factor SHOULD be as large as verification server performance will allow, typically at least 13. [C6]		X	X	916	5.1.1.2	☑	☑	☐	☐	☐			Fail, uses default that is 10. Can't currently be configured either.	☑

			C1-C10 Top 10 Proactive Controls: https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering)						258/286	122/258	146/258	4/258	174/258					79
V2	Authentication	2.5.1	Verify that a system generated initial activation or recovery secret is not sent in clear text to the user. [C6]	X	X	X	640	5.1.1.2	☑	☐	☐	☐	☐		Sent in clear text	Sent it in clear text by mail.		☐
V2	Authentication	2.5.2	Verify password hints or knowledge-based authentication (so-called "secret questions") are not present.	X	X	X	640	5.1.1.2	☑	☑	☑	☐	☑					☐
V2	Authentication	2.5.3	Verify password credential recovery does not reveal the current password in any way. [C6]	X	X	X	640	5.1.1.2	☑	☐	☑	☑	☑					☐
V2	Authentication	2.5.4	Verify shared or default accounts are not present (e.g. "root", "admin", or "sa").	X	X	X	16	5.1.1.2 / A.3	☑	☑	☑	☐	☑					☑
V2	Authentication	2.5.5	Verify that if an authentication factor is changed or replaced, that the user is notified of this event.	X	X	X	304	6.1.2.3	☑	☐	☐	☐	☐			Dont notify when things change.		☐
V2	Authentication	2.5.6	Verify forgotten password, and other recovery paths use a secure recovery mechanism, such as TOTP or other soft token, mobile push, or another offline recovery mechanism. [C6]	X	X	X	640	5.1.1.2	☑	☐	☑	☐	☑					☐
V2	Authentication	2.5.7	Verify that if OTP or multi-factor authentication factors are lost, that evidence of identity proofing is performed at the same level as during enrollment.		X	X	308	6.1.2.3	☑	☐	☑	☐	☑			Implementation because the way you recover is that you get in touch with the system adminstrator.		☑
V2	Authentication	2.7.1	Verify that clear text out of band (NIST "restricted") authenticators, such as SMS or PSTN, are not offered by default, and stronger alternatives such as push notifications are offered first.	X	X	X	287	5.1.3.2	☑	☐	☑	☐	☑					☑
V2	Authentication	2.7.2	Verify that the out of band verifier expires out of band authentication requests, codes, or tokens after 10 minutes.	X	X	X	287	5.1.3.2	☑	☐	☐	☐	☐					☐
V2	Authentication	2.7.3	Verify that the out of band verifier authentication requests, codes, or tokens are only usable once, and only for the original authentication request.	X	X	X	287	5.1.3.2	☑	☐	☐	☐	☐					☐
V2	Authentication	2.7.4	Verify that the out of band authenticator and verifier communicates over a secure independent channel.	X	X	X	523	5.1.3.2	☑	☐	☐	☐	☐					☐
V2	Authentication	2.7.5	Verify that the out of band verifier retains only a hashed version of the authentication code.		X	X	256	5.1.3.2	☑	☐	☐	☐	☐					☐
V2	Authentication	2.7.6	Verify that the initial authentication code is generated by a secure random number generator, containing at least 20 bits of entropy (typically a six digital random number is sufficient).		X	X	310	5.1.3.2	☑	☐	☐	☐	☐					☐
V2	Authentication	2.8.3	Verify that approved cryptographic algorithms are used in the generation, seeding, and verification.		X	X	326	5.1.4.2 / 5.1.5.2	☑	☐	☑	☐	☑					☐
V2	Authentication	2.9.3	Verify that approved cryptographic algorithms are used in the generation, seeding, and verification.		X	X	327	5.1.7.2	☑	☑	☑	☐	☑					☐
V2	Authentication	2.10.1	Verify that integration secrets do not rely on unchanging passwords, such as API keys or shared privileged accounts.		OS assisted	HSM	287	5.1.1.1	☑	☐	☑	☐	☑					☑
V2	Authentication	2.10.2	Verify that if passwords are required, the credentials are not a default account.		OS assisted	HSM	255	5.1.1.1	☑	☐	☑	☐	☑					☑
V2	Authentication	2.10.3	Verify that passwords are stored with sufficient protection to prevent offline recovery attacks, including local system access.		OS assisted	HSM	522	5.1.1.1	☑	☐	☑	☐	☑			Hashed.		☑
V2	Authentication	2.10.4	Verify passwords, integrations with databases and third-party systems, seeds and internal secrets, and API keys are managed securely and not included in the source code or stored within source code repositories. Such storage SHOULD resist offline attacks. The use of a secure software key store (L1), hardware trusted platform module (TPM), or a hardware security module (L3) is recommended for password storage.		OS assisted	HSM	798		☑	☐	☑	☐	☑			Have an external config for this, but implementors are in charge of using it correctly.		☑
V3	Session	3.1.1	Verify the application never reveals session tokens in URL parameters or error messages.	X	X	X	598		☑	☑	☑	☐	☑					☐
V3	Session	3.2.1	Verify the application generates a new session token on user authentication. [C6]	X	X	X	384	7.1	☑	☑	☐	☐	☑					☐
V3	Session	3.2.2	Verify that session tokens possess at least 64 bits of entropy. [C6]	X	X	X	331	7.1	☑	☐	☐	☐	☐					☐
V3	Session	3.2.3	Verify the application only stores session tokens in the browser using secure methods such as appropriately secured cookies (see section 3.4) or HTML 5 session storage.	X	X	X	539	7.1	☑	☐	☑	☐	☑		Not done in core, but only in the imlementation side. Provide guidelines.			☑
V3	Session	3.2.4	Verify that session token are generated using approved cryptographic algorithms. [C6]		X	X	331	7.1	☑	☑	☐	☐	☑					☐
V3	Session	3.3.1	Verify that logout and expiration invalidate the session token, such that the back button or a downstream relying party does not resume an authenticated session, including across relying parties. [C6]	X	X	X	613	7.1	☑	☑	☑	☐	☑					☐

			C1-C10 Top 10 Proactive Controls: https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering)						258/286	122/258	146/258	4/258	174/258					79
V3	Session	3.3.2	If authenticators permit users to remain logged in, verify that re-authentication occurs periodically both when actively used or after an idle period. [C6]	30 days	12 hours or 30 minutes of inactivity, 2FA optional	12 hours or 15 minutes of inactivity, with 2FA	613	7.2	☑	☑	☐	☐	☑					☐
V3	Session	3.3.3	Verify that the application terminates all other active sessions after a successful password change, and that this is effective across the application, federated login (if present), and any relying parties.		X	X	613		☑	☐	☑	☐	☑					☐
V3	Session	3.3.4	Verify that users are able to view and log out of any or all currently active sessions and devices.		X	X	613	7.1	☑	☐	☐	☐	☐		Only when you terminate a session, not selective.			☐
V3	Session	3.4.1	Verify that cookie-based session tokens have the 'Secure' attribute set. [C6]	X	X	X	614	7.1.1	☑	☑	☑	☐	☑					☐
V3	Session	3.4.2	Verify that cookie-based session tokens have the 'HttpOnly' attribute set. [C6]	X	X	X	1004	7.1.1	☑	☑	☐	☐	☑					☐
V3	Session	3.4.3	Verify that cookie-based session tokens utilize the 'SameSite' attribute to limit exposure to cross-site request forgery attacks. [C6]	X	X	X	16	7.1.1	☑	☐	☐	☐	☐		Not done in core, but only in the imlementation side. Provide guidelines.			☑
V3	Session	3.4.4	Verify that cookie-based session tokens use "__Host-" prefix (see references) to provide session cookie confidentiality.	X	X	X	16	7.1.1	☑	☐	☐	☐	☐					☐
V3	Session	3.4.5	Verify that if the application is published under a domain name with other applications that set or use session cookies that might override or disclose the session cookies, set the path attribute in cookie-based session tokens using the most precise path possible. [C6]	X	X	X	16	7.1.1	☑	☐	☐	☐	☐					☑
V3	Session	3.5.1	Verify the application does not treat OAuth and refresh tokens — on their own — as the presence of the subscriber and allows users to terminate trust relationships with linked applications.		X	X	290	7.1.2	☑	☑	☐	☐	☑					☐
V3	Session	3.5.2	Verify the application uses session tokens rather than static API secrets and keys, except with legacy implementations.		X	X	798		☑	☐	☑	☐	☑					☐
V3	Session	3.6.1	Verify that relying parties specify the maximum authentication time to CSPs and that CSPs re-authenticate the subscriber if they haven't used a session within that period.			X	613	7.2.1	☑	☐	☐	☐	☐					☑
V3	Session	3.6.2	Verify that CSPs inform relying parties of the last authentication event, to allow RPs to determine if they need to re-authenticate the user.			X	613	7.2.1	☑	☐	☐	☐	☐					☑
V3	Session	3.7.1	Verify the application ensures a valid login session or requires re-authentication or secondary verification before allowing any sensitive transactions or account modifications.	X	X	X	778		☑	☐	☐	☐	☐		Fail (being discussed).			☐
V4	Access	4.1.1	Verify that the application enforces access control rules on a trusted service layer, especially if client-side access control is present and could be bypassed.	X	X	X	602		☑	☑	☑	☐	☑					☐
V4	Access	4.1.2	Verify that all user and data attributes and policy information used by access controls cannot be manipulated by end users unless specifically authorized.	X	X	X	639		☑	☑	☑	☐	☑					☐
V4	Access	4.1.3	Verify that the principle of least privilege exists - users should only be able to access functions, data files, URLs, controllers, services, and other resources, for which they possess specific authorization. This implies protection against spoofing and elevation of privilege. [C7]	X	X	X	285		☑	☑	☑	☐	☑					☑
V4	Access	4.1.4	Verify that the principle of deny by default exists whereby new users/roles start with minimal or no permissions and users/roles do not receive access to new features until access is explicitly assigned. [C7]	X	X	X	276		☑	☐	☑	☐	☑					☐
V4	Access	4.1.5	Verify that access controls fail securely including when an exception occurs. [C10]	X	X	X	285		☑	☑	☑	☐	☑					☐
V4	Access	4.2.1	Verify that sensitive data and APIs are protected against direct object attacks targeting creation, reading, updating and deletion of records, such as creating or updating someone else's record, viewing everyone's records, or deleting all records.	X	X	X	639		☑	☑	☑	☐	☑					☐
V4	Access	4.2.2	Verify that the application or framework enforces a strong anti-CSRF mechanism to protect authenticated functionality, and effective anti-automation or anti-CSRF protects unauthenticated functionality.	X	X	X	352		☑	☐	☐	☐	☐					☐
V4	Access	4.3.1	Verify administrative interfaces use appropriate multi-factor authentication to prevent unauthorized use.	X	X	X	419		☑	☐	☐	☐	☐					☐

			C1-C10 Top 10 Proactive Controls: https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering)						258/286	122/258	146/258	4/258	174/258				79
V4	Access	4.3.2	Verify that directory browsing is disabled unless deliberately desired. Additionally, applications should not allow discovery or disclosure of file or directory metadata, such as Thumbs.db, .DS_Store, .git or .svn folders.	X	X	X	548		☑	☑	☑	☐	☑				☑
V4	Access	4.3.3	Verify the application has additional authorization (such as step up or adaptive authentication) for lower value systems, and / or segregation of duties for high value applications to enforce anti-fraud controls as per the risk of application and past fraud.		X	X	732		☑	☐	☑	☐	☑				☐
V5	Validation	5.1.1	Verify that the application has defenses against HTTP parameter pollution attacks, particularly if the application framework makes no distinction about the source of request parameters (GET, POST, cookies, headers, or environment variables).	X	X	X	235		☑	☑	☐	☐	☑				☐
V5	Validation	5.1.2	Verify that frameworks protect against mass parameter assignment attacks, or that the application has countermeasures to protect against unsafe parameter assignment, such as marking fields private or similar. [C5]	X	X	X	915		☑	☑	☑	☐	☑				☐
V5	Validation	5.1.3	Verify that all input (HTML form fields, REST requests, URL parameters, HTTP headers, cookies, batch files, RSS feeds, etc) is validated using positive validation (whitelisting). [C5]	X	X	X	20		☑	☑	☐	☐	☑				☐
V5	Validation	5.1.4	Verify that structured data is strongly typed and validated against a defined schema including allowed characters, length and pattern (e.g. credit card numbers or telephone, or validating that two related fields are reasonable, such as checking that suburb and zip/postcode match). [C5]	X	X	X	20		☑	☑	☑	☐	☑				☐
V5	Validation	5.1.5	Verify that URL redirects and forwards only allow whitelisted destinations, or show a warning when redirecting to potentially untrusted content.	X	X	X	601		☑	☐	☐	☐	☐				☐
V5	Validation	5.2.1	Verify that all untrusted HTML input from WYSIWYG editors or similar is properly sanitized with an HTML sanitizer library or framework feature. [C5]	X	X	X	116		☑	☐	☐	☐	☐				☐
V5	Validation	5.2.2	Verify that unstructured data is sanitized to enforce safety measures such as allowed characters and length.	X	X	X	138		☑	☐	☐	☐	☐				☐
V5	Validation	5.2.3	Verify that the application sanitizes user input before passing to mail systems to protect against SMTP or IMAP injection.	X	X	X	147		☑	☐	☐	☐	☐		Probably don't have sanitization.		☐
V5	Validation	5.2.4	Verify that the application avoids the use of eval() or other dynamic code execution features. Where there is no alternative, any user input being included must be sanitized or sandboxed before being executed.	X	X	X	95		☑	☑	☑	☐	☑		Limited in what you can do with the dynamics code execution.		☐
V5	Validation	5.2.5	Verify that the application protects against template injection attacks by ensuring that any user input being included is sanitized or sandboxed.	X	X	X	94		☑	☑	☐	☐	☐				☐
V5	Validation	5.2.6	Verify that the application protects against SSRF attacks, by validating or sanitizing untrusted data or HTTP file metadata, such as filenames and URL input fields, use whitelisting of protocols, domains, paths and ports.	X	X	X	918		☑	☐	☐	☐	☐				☐
V5	Validation	5.2.7	Verify that the application sanitizes, disables, or sandboxes user-supplied SVG scriptable content, especially as they relate to XSS resulting from inline scripts, and foreignObject.	X	X	X	159		☑	☑	☑	☐	☑				☐
V5	Validation	5.2.8	Verify that the application sanitizes, disables, or sandboxes user-supplied scriptable or expression template language content, such as Markdown, CSS or XSL stylesheets, BBCode, or similar.	X	X	X	94		☑	☑	☐	☐	☑		Don't do satinization / frontend		☐
V5	Validation	5.3.1	Verify that output encoding is relevant for the interpreter and context required. For example, use encoders specifically for HTML values, HTML attributes, JavaScript, URL Parameters, HTTP headers, SMTP, and others as the context requires, especially from untrusted inputs (e.g. names with Unicode or apostrophes, such as ねこ or O'Hara). [C4]	X	X	X	116		☑	☑	☐	☐	☑				☐
V5	Validation	5.3.2	Verify that output encoding preserves the user's chosen character set and locale, such that any Unicode character point is valid and safely handled. [C4]	X	X	X	176		☑	☑	☐	☐	☑				☐
V5	Validation	5.3.3	Verify that context-aware, preferably automated - or at worst, manual - output escaping protects against reflected, stored, and DOM based XSS. [C4]	X	X	X	79		☑	☑	☐	☐	☑				☐
V5	Validation	5.3.4	Verify that data selection or database queries (e.g. SQL, HQL, ORM, NoSQL) use parameterized queries, ORMs, entity frameworks, or are otherwise protected from database injection attacks. [C3]	X	X	X	89		☑	☑	☑	☐	☑				☐
V5	Validation	5.3.5	Verify that where parameterized or safer mechanisms are not present, context-specific output encoding is used to protect against injection attacks, such as the use of SQL escaping to protect against SQL injection. [C3, C4]	X	X	X	89		☑	☑	☑	☐	☑				☐
V5	Validation	5.3.6	Verify that the application projects against JavaScript or JSON injection attacks, including for eval attacks, remote JavaScript includes, CSP bypasses, DOM XSS, and JavaScript expression evaluation. [C4]	X	X	X	830		☑	☑	☐	☐	☑				☐
V5	Validation	5.3.7	Verify that the application protects against LDAP Injection vulnerabilities, or that specific security controls to prevent LDAP Injection have been implemented. [C4]	X	X	X	943		☑	☐	☐	☐	☐				☐
V5	Validation	5.3.8	Verify that the application protects against OS command injection and that operating system calls use parameterized OS queries or use contextual command line output encoding. [C4]	X	X	X	78		☑	☐	☑	☐	☑				☐
V5	Validation	5.3.9	Verify that the application protects against Local File Inclusion (LFI) or Remote File Inclusion (RFI) attacks.	X	X	X	829		☑	☐	☑	☐	☑				☐

			C1-C10 Top 10 Proactive Controls: https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering)						258/286	122/258	146/258	4/258	174/258				79
V5	Validation	5.3.10	Verify that the application protects against XPath injection or XML injection attacks. [C4]	X	X	X	643		☑	☑	☐	☐	☑				☐
V5	Validation	5.4.1	Verify that the application uses memory-safe string, safer memory copy and pointer arithmetic to detect or prevent stack, buffer, or heap overflows.		X	X	120		☑	☑	☐	☐	☑				☐
V5	Validation	5.4.2	Verify that format strings do not take potentially hostile input, and are constant.		X	X	134		☑	☑	☐	☐	☑				☐
V5	Validation	5.4.3	Verify that sign, range, and input validation techniques are used to prevent integer overflows.		X	X	190		☑	☑	☐	☐	☑		Possibly not relevant for Java.		☐
V5	Validation	5.5.1	Verify that serialized objects use integrity checks or are encrypted to prevent hostile object creation or data tampering. [C5]	X	X	X	502		☑	☐	☑	☐	☑				☐
V5	Validation	5.5.2	Verify that the application correctly restricts XML parsers to only use the most restrictive configuration possible and to ensure that unsafe features such as resolving external entities are disabled to prevent XXE.	X	X	X	611		☑	☑	☑	☐	☑				☐
V5	Validation	5.5.3	Verify that deserialization of untrusted data is avoided or is protected in both custom code and third-party libraries (such as JSON, XML and YAML parsers).	X	X	X	502		☑	☑	☑	☐	☑		All that is user supplied, so in that sense it is trusted since users need to log in		☐
V5	Validation	5.5.4	Verify that when parsing JSON in browsers or JavaScript-based backends, JSON.parse is used to parse the JSON document. Do not use eval() to parse JSON.	X	X	X	95		☑	☑	☐	☐	☑				☐
V6	Cryptography	6.1.1	Verify that regulated private data is stored encrypted while at rest, such as personally identifiable information (PII), sensitive personal information, or data assessed likely to be subject to EU's GDPR.		X	X	311		☑	☐	☑	☐	☑				☑
V6	Cryptography	6.1.2	Verify that regulated health data is stored encrypted while at rest, such as medical records, medical device details, or de-anonymized research records.		X	X	311		☑	☐	☑	☐	☑		Have the ability to encrypt the data by an attributes basis		☑
V6	Cryptography	6.1.3	Verify that regulated financial data is stored encrypted while at rest, such as financial accounts, defaults or credit history, tax records, pay history, beneficiaries, or de-anonymized market or research records.		X	X	311		☑	☐	☑	☐	☐		Don't work with just health data, can be other forms of data. Implementation issue		☑
V6	Cryptography	6.2.1	Verify that all cryptographic modules fail securely, and errors are handled in a way that does not enable Padding Oracle attacks.	X	X	X	310		☑	☐	☐	☐	☐				☐
V6	Cryptography	6.2.2	Verify that industry proven or government approved cryptographic algorithms, modes, and libraries are used, instead of custom coded cryptography. [C8]		X	X	327		☑	☑	☑	☐	☑				☐
V6	Cryptography	6.2.3	Verify that encryption initialization vector, cipher configuration, and block modes are configured securely using the latest advice.		X	X	326		☑	☑	☐	☐	☑				☐
V6	Cryptography	6.2.4	Verify that random number, encryption or hashing algorithms, key lengths, rounds, ciphers or modes, can be reconfigured, upgraded, or swapped at any time, to protect against cryptographic breaks. [C8]		X	X	326		☑	☑	☑	☐	☑		Have the ability to reconfigure and swap		☐
V6	Cryptography	6.2.5	Verify that known insecure block modes (i.e. ECB, etc.), padding modes (i.e. PKCS#1 v1.5, etc.), ciphers with small block sizes (i.e. Triple-DES, Blowfish, etc.), and weak hashing algorithms (i.e. MD5, SHA1, etc.) are not used unless required for backwards compatibility.		X	X	326		☑	☐	☑	☐	☑		Use AES		☐
V6	Cryptography	6.2.6	Verify that nonces, initialization vectors, and other single use numbers must not be used more than once with a given encryption key. The method of generation must be appropriate for the algorithm being used.		X	X	326		☑	☐	☐	☐	☐				☐
V6	Cryptography	6.2.8	Verify that all cryptographic operations are constant-time, with no 'short-circuit' operations in comparisons, calculations, or returns, to avoid leaking information.			X	385		☑	☐	☐	☐	☐				☐
V6	Cryptography	6.3.1	Verify that all random numbers, random file names, random GUIDs, and random strings are generated using the cryptographic module's approved cryptographically secure random number generator when these random values are intended to be not guessable by an attacker.		X	X	338		☑	☐	☐	☐	☐		Fail for no specific crypto module is used to generate these things		☐
V6	Cryptography	6.3.2	Verify that random GUIDs are created using the GUID v4 algorithm, and a cryptographically-secure pseudo-random number generator (CSPRNG). GUIDs created using other pseudo-random number generators may be predictable.		X	X	338		☑	☐	☐	☐	☐				☐
V6	Cryptography	6.3.3	Verify that random numbers are created with proper entropy even when the application is under heavy load, or that the application degrades gracefully in such circumstances.			X	338		☑	☐	☐	☐	☐				☐
V6	Cryptography	6.4.1	Verify that a secrets management solution such as a key vault is used to securely create, store, control access to and destroy secrets. [C8]		X	X	798		☑	☐	☑	☐	☑				☑
V6	Cryptography	6.4.2	Verify that key material is not exposed to the application but instead uses an isolated security module like a vault for cryptographic operations. [C8]		X	X	320		☑	☐	☑	☐	☑				☑
V7	Error	7.1.1	Verify that the application does not log credentials or payment details. Session tokens should only be stored in logs in an irreversible, hashed form. [C9, C10]	X	X	X	532		☑	☑	☑	☐	☑				☐
V7	Error	7.1.2	Verify that the application does not log other sensitive data as defined under local privacy laws or relevant security policy. [C9]	X	X	X	532		☑	☐	☐	☐	☐				☑
V7	Error	7.1.3	Verify that the application logs security relevant events including successful and failed authentication events, access control failures, deserialization failures and input validation failures. [C5, C7]		X	X	778		☑	☐	☑	☐	☑				☐
V7	Error	7.1.4	Verify that each log event includes necessary information that would allow for a detailed investigation of the timeline when an event happens. [C9]		X	X	778		☑	☐	☐	☐	☐		Unsure on how complete the information is		☐
V7	Error	7.2.1	Verify that all authentication decisions are logged, without storing sensitive session identifiers or passwords. This should include requests with relevant metadata needed for security investigations.		X	X	778		☑	☑	☑	☐	☑				☐

			C1-C10 Top 10 Proactive Controls: https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering)						258/286	122/258	146/258	4/258	174/258				79
V7	Error	7.2.2	Verify that all access control decisions can be logged and all failed decisions are logged. This should include requests with relevant metadata needed for security investigations.		X	X	285		☑	☑	☐	☐	☑				☐
V7	Error	7.3.1	Verify that the application appropriately encodes user-supplied data to prevent log injection. [C9]		X	X	117		☑	☐	☑	☐	☑				☐
V7	Error	7.3.2	Verify that all events are protected from injection when viewed in log viewing software. [C9]		X	X	117		☑	☐	☐	☐	☐				☑
V7	Error	7.3.3	Verify that security logs are protected from unauthorized access and modification. [C9]		X	X	200		☑	☐	☑	☐	☑				☑
V7	Error	7.3.4	Verify that time sources are synchronized to the correct time and time zone. Strongly consider logging only in UTC if systems are global to assist with post-incident forensic analysis. [C9]		X	X			☑	☐	☑	☐	☑				☑
V7	Error	7.4.1	Verify that a generic message is shown when an unexpected or security sensitive error occurs, potentially with a unique ID which support personnel can use to investigate. [C10]	X	X	X	210		☑	☐	☑	☐	☑				☐
V7	Error	7.4.2	Verify that exception handling (or a functional equivalent) is used across the codebase to account for expected and unexpected error conditions. [C10]		X	X	544		☑	☑	☑	☐	☑				☐
V7	Error	7.4.3	Verify that a "last resort" error handler is defined which will catch all unhandled exceptions. [C10]		X	X	460		☑	☑	☐	☐	☑				☐
V8	Data	8.1.1	Verify the application protects sensitive data from being cached in server components such as load balancers and application caches.		X	X	524		☑	☑	☐	☐	☐		Fail, but is also an implementation responsbility. Don't have a way specify things are sensitive.		☑
V8	Data	8.1.3	Verify the application minimizes the number of parameters in a request, such as hidden fields, Ajax variables, cookies and header values.		X	X	233		☑	☑	☐	☐	☑				☑
V8	Data	8.1.4	Verify the application can detect and alert on abnormal numbers of requests, such as by IP, user, total per hour or day, or whatever makes sense for the application.		X	X	770		☑	☐	☐	☐	☐		Don't have this for the API, only for login and password recovery.		☑
V8	Data	8.2.1	Verify the application sets sufficient anti-caching headers so that sensitive data is not cached in modern browsers.	X	X	X	525		☑	☑	☐	☐	☑				☑
V8	Data	8.2.2	Verify that data stored in client side storage (such as HTML5 local storage, session storage, IndexedDB, regular cookies or Flash cookies) does not contain sensitive data or PII.	X	X	X	922		☑	☑	☑	☐	☑				☑
V8	Data	8.2.3	Verify that authenticated data is cleared from client storage, such as the browser DOM, after the client or session is terminated.	X	X	X	922		☑	☑	☐	☐	☑				☐
V8	Data	8.3.1	Verify that sensitive data is sent to the server in the HTTP message body or headers, and that query string parameters from any HTTP verb do not contain sensitive data.	X	X	X	319		☑	☑	☑	☐	☑				☐
V8	Data	8.3.2	Verify that users have a method to remove or export their data on demand.	X	X	X	212		☑	☐	☑	☐	☐		Can export all data on demand.		☐
V8	Data	8.3.3	Verify that users are provided clear language regarding collection and use of supplied personal information and that users have provided opt-in consent for the use of that data before it is used in any way.	X	X	X	285		☑	☐	☑	☐	☐				☑
V8	Data	8.3.5	Verify accessing sensitive data is audited (without logging the sensitive data itself), if the data is collected under relevant data protection directives or where logging of access is required.		X	X	532		☑	☐	☐	☐	☐				☐
V8	Data	8.3.6	Verify that sensitive information contained in memory is overwritten as soon as it is no longer required to mitigate memory dumping attacks, using zeroes or random data.		X	X	226		☑	☐	☐	☐	☐		Difficult for you can't know how to classify sensitive data.		☐
V8	Data	8.3.7	Verify that sensitive or private information that is required to be encrypted, is encrypted using approved algorithms that provide both confidentiality and integrity. [C8]		X	X	327		☑	☐	☑	☐	☑				☐
V8	Data	8.3.8	Verify that sensitive personal information is subject to data retention classification, such that old or out of date data is deleted automatically, on a schedule, or as the situation requires.		X	X	285		☑	☐	☑	☐	☑				☐
V9	Communications	9.1.1	Verify that secured TLS is used for all client connectivity, and does not fall back to insecure or unencrypted protocols. [C8]	X	X	X	319		☑	☑	☑	☐	☑		Have support to use TLS and SSL		☑
V9	Communications	9.1.2	Verify using online or up to date TLS testing tools that only strong algorithms, ciphers, and protocols are enabled, with the strongest algorithms and ciphers set as preferred.	X	X	X	326		☑	☑	☑	☐	☑				☑
V9	Communications	9.1.3	Verify that old versions of SSL and TLS protocols, algorithms, ciphers, and configuration are disabled, such as SSLv2, SSLv3, or TLS 1.0 and TLS 1.1. The latest version of TLS should be the preferred cipher suite.	X	X	X	326		☑	☑	☑	☐	☑				☐
V9	Communications	9.2.1	Verify that connections to and from the server use trusted TLS certificates. Where internally generated or self-signed certificates are used, the server must be configured to only trust specific internal CAs and specific self-signed certificates. All others should be rejected.		X	X	295		☑	☑	☑	☐	☑				☑
V9	Communications	9.2.2	Verify that encrypted communications such as TLS is used for all inbound and outbound connections, including for management ports, monitoring, authentication, API, or web service calls, database, cloud, serverless, mainframe, external, and partner connections. The server must not fall back to insecure or unencrypted protocols.		X	X	319		☑	☑	☑	☐	☑				☑

			C1-C10 Top 10 Proactive Controls: https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering)					258/286	122/258	146/258	4/258	174/258				79
V9	Communications	9.2.3	Verify that all encrypted connections to external systems that involve sensitive information or functions are authenticated.		X	X	287	☑	☑	☑	☐	☑				☑
V9	Communications	9.2.5	Verify that backend TLS connection failures are logged.			X	544	☑	☐	☑	☐	☑				☑
V10	Malicious	10.1.1	Verify that a code analysis tool is in use that can detect potentially malicious code, such as time functions, unsafe file operations and network connections.			X	749	☑	☐	☑	☐	☑				☐
V10	Malicious	10.2.1	Verify that the application source code and third party libraries do not contain unauthorized phone home or data collection capabilities. Where such functionality exists, obtain the user's permission for it to operate before collecting any data.		X	X	359	☑	☐	☑	☐	☑		Up to implementation for we have the possiblity to integrated google analytics, but its up to them who must get the user permissions for that part.		☑
V10	Malicious	10.2.2	Verify that the application does not ask for unnecessary or excessive permissions to privacy related features or sensors, such as contacts, cameras, microphones, or location.		X	X	272	☑	☑	☑	☐	☑				☐
V10	Malicious	10.2.3	Verify that the application source code and third party libraries do not contain back doors, such as hard-coded or additional undocumented accounts or keys, code obfuscation, undocumented binary blobs, rootkits, or anti-debugging, insecure debugging features, or otherwise out of date, insecure, or hidden functionality that could be used maliciously if discovered.			X	507	☑	☐	☑	☐	☑				☐
V10	Malicious	10.2.4	Verify that the application source code and third party libraries does not contain time bombs by searching for date and time related functions.			X	511	☑	☐	☐	☐	☐		Don't check if the libraries we check outside of what is automatically is available on the internet. If there are generated reports from the libraries, then then there is no fail.		☐
V10	Malicious	10.2.5	Verify that the application source code and third party libraries does not contain malicious code, such as salami attacks, logic bypasses, or logic bombs.			X	511	☑	☐	☐	☐	☐		Don't check if the libraries we check outside of what is automatically is available on the internet. If there are generated reports from the libraries, then then there is no fail.		☐
V10	Malicious	10.2.6	Verify that the application source code and third party libraries do not contain Easter eggs or any other potentially unwanted functionality.			X	507	☑	☐	☐	☐	☐		Don't check if the libraries we check outside of what is automatically is available on the internet. If there are generated reports from the libraries, then then there is no fail.		☐
V10	Malicious	10.3.1	Verify that if the application has a client or server auto-update feature, updates should be obtained over secure channels and digitally signed. The update code must validate the digital signature of the update before installing or executing the update.	X	X	X	16	☑	☐	☑	☐	☑				☐
V10	Malicious	10.3.2	Verify that the application employs integrity protections, such as code signing or sub-resource integrity. The application must not load or execute code from untrusted sources, such as loading includes, modules, plugins, code, or libraries from untrusted sources or the Internet.	X	X	X	353	☑	☑	☑	☐	☑		Pass for the back end, but for the custom apps this may be an issue.		☐
V10	Malicious	10.3.3	Verify that the application has protection from sub-domain takeovers if the application relies upon DNS entries or DNS sub-domains, such as expired domain names, out of date DNS pointers or CNAMEs, expired projects at public source code repos, or transient cloud APIs, serverless functions, or storage buckets (autogen-bucket-id.cloud.example.com) or similar. Protections can include ensuring that DNS names used by applications are regularly checked for expiry or change.	X	X	X	350	☑	☑	☑	☐	☑				☑
V11	BusLogic	11.1.1	Verify the application will only process business logic flows for the same user in sequential step order and without skipping steps.	X	X	X	841	☑	☑	☑	☐	☑				☐
V11	BusLogic	11.1.2	Verify the application will only process business logic flows with all steps being processed in realistic human time, i.e. transactions are not submitted too quickly.	X	X	X	779	☑	☑	☑	☐	☑				☐
V11	BusLogic	11.1.3	Verify the application has appropriate limits for specific business actions or transactions which are correctly enforced on a per user basis.	X	X	X	770	☑	☐	☑	☐	☑				☐
V11	BusLogic	11.1.4	Verify the application has sufficient anti-automation controls to detect and protect against data exfiltration, excessive business logic requests, excessive file uploads or denial of service attacks.	X	X	X	770	☑	☐	☑	☐	☑				☑
V11	BusLogic	11.1.5	Verify the application has business logic limits or validation to protect against likely business risks or threats, identified using threat modelling or similar methodologies.	X	X	X	841	☑	☐	☐	☐	☐				☐
V11	BusLogic	11.1.6	Verify the application does not suffer from "time of check to time of use" (TOCTOU) issues or other race conditions for sensitive operations.		X	X	367	☑	☐	☑	☐	☑		Could be relevant for password recovery. Had an issue recently that was fixed.		☐
V11	BusLogic	11.1.8	Verify the application has configurable alerting when automated attacks or unusual activity is detected.		X	X	390	☑	☐	☑	☐	☑				☑
V12	Files	12.1.1	Verify that the application will not accept large files that could fill up storage or cause a denial of service attack.	X	X	X	400	☑	☐	☐	☐	☐				☐
V12	Files	12.1.2	Verify that compressed files are checked for "zip bombs" - small input files that will decompress into huge files thus exhausting file storage limits.		X	X	409	☑	☐	☐	☐	☐		Have no check on it.		☐
V12	Files	12.1.3	Verify that a file size quota and maximum number of files per user is enforced to ensure that a single user cannot fill up the storage with too many files, or excessively large files.		X	X	770	☑	☐	☐	☐	☐				☐
V12	Files	12.2.1	Verify that files obtained from untrusted sources are validated to be of expected type based on the file's content.		X	X	434	☑	☑	☐	☐	☐		Don't validate all files. For pictures, we verify that it's a picture etc.		☐

			C1-C10 Top 10 Proactive Controls: https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering)						258/286	122/258	146/258	4/258	174/258					79
V12	Files	12.3.1	Verify that user-submitted filename metadata is not used directly with system or framework file and URL API to protect against path traversal.	X	X	X	22		☑	☑	☑	☐	☑			Handled by the external framework.		☐
V12	Files	12.3.2	Verify that user-submitted filename metadata is validated or ignored to prevent the disclosure, creation, updating or removal of local files (LFI).	X	X	X	73		☑	☑	☑	☐	☑					☐
V12	Files	12.3.3	Verify that user-submitted filename metadata is validated or ignored to prevent the disclosure or execution of remote files (RFI), which may also lead to SSRF.	X	X	X	98		☑	☑	☑	☐	☑					☐
V12	Files	12.3.4	Verify that the application protects against reflective file download (RFD) by validating or ignoring user-submitted filenames in a JSON, JSONP, or URL parameter, the response Content-Type header should be set to text/plain, and the Content-Disposition header should have a fixed filename.	X	X	X	641		☑	☑	☑	☐	☑					☐
V12	Files	12.3.5	Verify that untrusted file metadata is not used directly with system API or libraries, to protect against OS command injection.	X	X	X	78		☑	☐	☑	☐	☑					☐
V12	Files	12.3.6	Verify that the application does not include and execute functionality from untrusted sources, such as unverified content distribution networks, JavaScript libraries, node npm libraries, or server-side DLLs.		X	X	829		☑	☐	☑	☑	☑		We use a lot of NPM libraries, though we attempt to code review the ones we rely on.			☐
V12	Files	12.4.1	Verify that files obtained from untrusted sources are stored outside the web root, with limited permissions, preferably with strong validation.	X	X	X	922		☑	☑	☑	☐	☑					☐
V12	Files	12.4.2	Verify that files obtained from untrusted sources are scanned by antivirus scanners to prevent upload of known malicious content.	X	X	X	509		☑	☐	☐	☐	☐					☑
V12	Files	12.5.1	Verify that the web tier is configured to serve only files with specific file extensions to prevent unintentional information and source code leakage. For example, backup files (e.g. .bak), temporary working files (e.g. .swp), compressed files (.zip, .tar.gz, etc) and other extensions commonly used by editors should be blocked unless required.	X	X	X	552		☑	☑	☐	☐	☑					☐
V12	Files	12.5.2	Verify that direct requests to uploaded files will never be executed as HTML/JavaScript content.	X	X	X	434		☑	☑	☐	☐	☑					☐
V12	Files	12.6.1	Verify that the web or application server is configured with a whitelist of resources or systems to which the server can send requests or load data/files from.	X	X	X	918		☑	☑	☑	☐	☑			impl. server configuration		☑
V13	API	13.1.1	Verify that all application components use the same encodings and parsers to avoid parsing attacks that exploit different URI or file parsing behavior that could be used in SSRF and RFI attacks.	X	X	X	116		☑	☑	☑	☐	☑					☐
V13	API	13.1.2	Verify that access to administration and management functions is limited to authorized administrators.	X	X	X	419		☑	☐	☑	☐	☑					☑
V13	API	13.1.3	Verify API URLs do not expose sensitive information, such as the API key, session tokens etc.	X	X	X	598		☑	☑	☑	☐	☑					☐
V13	API	13.1.4	Verify that authorization decisions are made at both the URI, enforced by programmatic or declarative security at the controller or router, and at the resource level, enforced by model-based permissions.		X	X	285		☑	☑	☑	☐	☑					☐
V13	API	13.1.5	Verify that requests containing unexpected or missing content types are rejected with appropriate headers (HTTP response status 406 Unacceptable or 415 Unsupported Media Type).		X	X	434		☑	☑	☑	☐	☑					☐
V13	API	13.2.1	Verify that enabled RESTful HTTP methods are a valid choice for the user or action, such as preventing normal users using DELETE or PUT on protected API or resources.	X	X	X	650		☑	☑	☑	☐	☑					☐
V13	API	13.2.2	Verify that JSON schema validation is in place and verified before accepting input.	X	X	X	20		☑	☑	☑	☐	☑					☐
V13	API	13.2.3	Verify that RESTful web services that utilize cookies are protected from Cross-Site Request Forgery via the use of at least one or more of the following: triple or double submit cookie pattern (see https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet), CSRF nonces, or ORIGIN request header checks.	X	X	X	352		☑	☐	☐	☐	☐					☐
V13	API	13.2.4	Verify that REST services have anti-automation controls to protect against excessive calls, especially if the API is unauthenticated.		X	X	779		☑	☐	☐	☐	☐					☐
V13	API	13.2.5	Verify that REST services explicitly check the incoming Content-Type to be the expected one, such as application/xml or application/JSON.		X	X	436		☑	☑	☑	☐	☑					☐
V13	API	13.2.6	Verify that the message headers and payload are trustworthy and not modified in transit. Requiring strong encryption for transport (TLS only) may be sufficient in many cases as it provides both confidentiality and integrity protection. Per-message digital signatures can provide additional assurance on top of the transport protections for high-security applications but bring with them additional complexity and risks to weigh against the benefits.		X	X	345		☑	☑	☐	☐	☑					☐
V13	API	13.3.1	Verify that XSD schema validation takes place to ensure a properly formed XML document, followed by validation of each input field before any processing of that data takes place.	X	X	X	20		☑	☑	☐	☐	☑			Possibly handled by another library		☐

			C1-C10 Top 10 Proactive Controls: https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering)						258/286	122/258	146/258	4/258	174/258					79
V13	API	13.3.2	Verify that the message payload is signed using WS-Security to ensure reliable transport between client and service.		X	X	345		☑	☐	☐	☐	☐					☐
V13	API	13.4.1	Verify that query whitelisting or a combination of depth limiting and amount limiting should be used to prevent GraphQL or data layer expression denial of service (DoS) as a result of expensive, nested queries. For more advanced scenarios, query cost analysis should be used.		X	X	770		☑	☐	☑	☐	☐					☐
V13	API	13.4.2	Verify that GraphQL or other data layer authorization logic should be implemented at the business logic layer instead of the GraphQL layer.		X	X	285		☑	☐	☑	☐	☐					☐
V14	Config	14.1.1	Verify that the application build and deployment processes are performed in a secure and repeatable way, such as CI / CD automation, automated configuration management, and automated deployment scripts.		X	X			☑	☑	☑	☐	☑					☐
V14	Config	14.1.2	Verify that compiler flags are configured to enable all available buffer overflow protections and warnings, including stack randomization, data execution prevention, and to break the build if an unsafe pointer, memory, format string, integer, or string operations are found.		X	X	120		☑	☐	☐	☐	☐					☐
V14	Config	14.1.4	Verify that the application, configuration, and all dependencies can be re-deployed using automated deployment scripts, built from a documented and tested runbook in a reasonable time, or restored from backups in a timely fashion.		X	X			☑	☐	☑	☐	☑					☑
V14	Config	14.1.5	Verify that authorized administrators can verify the integrity of all security-relevant configurations to detect tampering.			X			☑	☐	☑	☐	☑					☑
V14	Config	14.2.1	Verify that all components are up to date, preferably using a dependency checker during build or compile time. [C2]	X	X	X	1026		☑	☑	☑	☐	☑		Do it manually.			☐
V14	Config	14.2.2	Verify that all unneeded features, documentation, samples, configurations are removed, such as sample applications, platform documentation, and default or example users.	X	X	X	1002		☑	☑	☑	☐	☑		Doing manually on ocassion.			☐
V14	Config	14.2.4	Verify that third party components come from pre-defined, trusted and continually maintained repositories. [C2]		X	X	829		☑	☑	☑	☐	☑					☐
V14	Config	14.2.5	Verify that an inventory catalog is maintained of all third party libraries in use. [C2]		X	X			☑	☐	☑	☐	☐			Fail, maybe could be auto generated		☐
V14	Config	14.2.6	Verify that the attack surface is reduced by sandboxing or encapsulating third party libraries to expose only the required behaviour into the application. ([C2] (https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering))		X	X	265		☑	☐	☐	☐	☐					☐
V14	Config	14.3.1	Verify that web or application server and framework error messages are configured to deliver user actionable, customized responses to eliminate any unintended security disclosures.	X	X	X	209		☑	☑	☑	☐	☑					☐
V14	Config	14.3.2	Verify that web or application server and application framework debug modes are disabled in production to eliminate debug features, developer consoles, and unintended security disclosures.	X	X	X	497		☑	☑	☑	☐	☑		Implementation responsbility. They have the posiblity to run debug opeations, but its not activated by default.			☑
V14	Config	14.3.3	Verify that the HTTP headers or any part of the HTTP response do not expose detailed version information of system components.	X	X	X	200		☑	☑	☑	☐	☑					☐
V14	Config	14.4.1	Verify that every HTTP response contains a content type header specifying a safe character set (e.g., UTF-8, ISO 8859-1).	X	X	X	173		☑	☑	☐	☐	☑					☐
V14	Config	14.4.2	Verify that all API responses contain Content-Disposition: attachment; filename="api.json" (or other appropriate filename for the content type).	X	X	X	116		☑	☐	☐	☐	☐					☐
V14	Config	14.4.3	Verify that a content security policy (CSPv2) is in place that helps mitigate impact for XSS attacks like HTML, DOM, JSON, and JavaScript injection vulnerabilities.	X	X	X	1021		☑	☑	☐	☐	☑					☐
V14	Config	14.4.4	Verify that all responses contain X-Content-Type-Options: nosniff.	X	X	X	116		☑	☐	☐	☐	☐					☐
V14	Config	14.4.5	Verify that HTTP Strict Transport Security headers are included on all responses and for all subdomains, such as Strict-Transport-Security: max-age=15724800; includeSubdomains.	X	X	X	523		☑	☐	☐	☐	☐					☐
V14	Config	14.4.6	Verify that a suitable "Referrer-Policy" header is included, such as "no-referrer" or "same-origin".	X	X	X	116		☑	☑	☐	☐	☑					☐
V14	Config	14.4.7	Verify that a suitable X-Frame-Options or Content-Security-Policy: frame-ancestors header is in use for sites where content should not be embedded in a third-party site.	X	X	X	346		☑	☑	☐	☐	☑					☐
V14	Config	14.5.1	Verify that the application server only accepts the HTTP methods in use by the application or API, including pre-flight OPTIONS.	X	X	X	749		☑	☑	☐	☐	☑					☐
V14	Config	14.5.2	Verify that the supplied Origin header is not used for authentication or access control decisions, as the Origin header can easily be changed by an attacker.	X	X	X	346		☑	☑	☐	☐	☑					☐
V14	Config	14.5.3	Verify that the cross-domain resource sharing (CORS) Access-Control-Allow-Origin header uses a strict white-list of trusted domains to match against and does not support the "null" origin.	X	X	X	346		☑	☑	☐	☐	☑		They do have a whitelist.			☑
V14	Config	14.5.4	Verify that HTTP headers added by a trusted proxy or SSO devices, such as a bearer token, are authenticated by the application.		X	X	306		☑	☐	☐	☐	☐					☐

B.1 Updated scoring

This spreadsheet shows the results from the scoring conducted by the security engineer in early April 2020. The security engineer re-scored the 50 controls that had previously been a fail.

The old and new results are presented side-by-side, with additional comments. Controls that will be passing soon are marked blue.

			C1-C10 Top 10 Proactive Controls: https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering)						258/286	174/258	189/258		
			ASVS 4.0							Old results	New results	Security Engineer updated	Side comments
Section	**Name**	**Item**	**Description**	**L1**	**L2**	**L3**	**CWE**	**NIST**	**Applicable**				
V1	Architecture	1.1.1	Verify the use of a secure software development lifecycle that addresses security in all stages of development. [C1]		X	X			☑	☐	☑	DHIS2 uses Github for most of the dev. cycle including a CI pipeline. We have a commit review process that requires two persons to review each commit before it can be merged. As per 2 April, 2020 Github still don't make signatures for releases available automatically. This can be added. Overall DHIS2 uses a very standard dev cycle and are considered safe enough for an opensource system. It's not perfect and there are still improvements that can be easily added like release signatures.	I consider it safe enough for DHIS2
V1	Architecture	1.1.2	Verify the use of threat modeling for every design change or sprint planning to identify threats, plan for countermeasures, facilitate appropriate risk responses, and guide security testing.		X	X	1053		☑	☐	☑	We are in the process of making available checklists etc.. We have a lot of pentests done on behalf of organizations that use DHIS2, lately there has been more interest from partners to ensure the system is secure. There has been a pretty continuous effort in terms of pentesting the last year. Automatically pentesting in the CI pipeline is also coming soon.	I consider it better than in the first assment. Should be passing.
V1	Architecture	1.1.5	Verify definition and security analysis of the application's high-level architecture and all connected remote services. [C1]		X	X	1059		☑	☐	☐		
V1	Architecture	1.1.7	Verify availability of a secure coding checklist, security requirements, guideline, or policy to all developers and testers.		X	X	637		☑	☐	☐	Work in progress.	
V1	Architecture	1.2.3	Verify that the application uses a single vetted authentication mechanism that is known to be secure, can be extended to include strong authentication, and has sufficient logging and monitoring to detect account abuse or breaches.		X	X	306		☑	☐	☑	In the last 6 months, there have been lots of improvements in this area. DHIS2 uses industry-standard authentication system in most aspect, like password algorithms and 2-factor auth. Lots of improments in auditing and montoring have been implemented lately too.	I consider this a pass now.
V1	Architecture	1.4.5	Verify that attribute or feature-based access control is used whereby the code checks the user's authorization for a feature/data item rather than just their role. Permissions should still be allocated using roles. [C7]		X	X	275		☑	☐	☐		
V1	Architecture	1.6.2	Verify that consumers of cryptographic services protect key material and other secrets by using key vaults or API based alternatives.		X	X	320		☑	☐	☐	DHIS2 has support for symmetric encryption of some values, for example, attributes that are considered sensitive. The encryption key management and configuration are pretty simple and can be improved upon. This is planned to be improved in upcoming versions.	
V1	Architecture	1.6.3	Verify that all keys and passwords are replaceable and are part of a well-defined process to re-encrypt sensitive data.		X	X	320		☑	☐	☐	Plans to implement a password re-hashing feature, making it possible having a different kind of password hashes at the same time.	
V1	Architecture	1.6.4	Verify that symmetric keys, passwords, or API secrets generated by or shared with clients are used only in protecting low risk secrets, such as encrypting local storage, or temporary ephemeral uses such as parameter obfuscation. Sharing secrets with clients is clear-text equivalent and architecturally should be treated as such.		X	X	320		☑	☐	☑		Pass
V1	Architecture	1.7.2	Verify that logs are securely transmitted to a preferably remote system for analysis, detection, alerting, and escalation. [C9]		X	X			☑	☐	☐		
V1	Architecture	1.12.2	Verify that user-uploaded files - if required to be displayed or downloaded from the application - are served by either octet stream downloads, or from an unrelated domain, such as a cloud file storage bucket. Implement a suitable content security policy to reduce the risk from XSS vectors or other attacks from the uploaded file.		X	X	646		☑	☐	☐	Currently being fixed as part of pentest, should be available in next version.	Passing soon

			C1-C10 Top 10 Proactive Controls: https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering)						258/286	174/258	189/258		
V2	Authentication	2.1.7	Verify that passwords submitted during account registration, login, and password change are checked against a set of breached passwords either locally (such as the top 1,000 or 10,000 most common passwords which match the system's password policy) or using an external API. If using an API a zero knowledge proof or other mechanism should be used to ensure that the plain text password is not sent or used in verifying the breach status of the password. If the password is breached, the application must require the user to set a new non-breached password. [C6]	X	X	X	521	5.1.1.2	☑	☐	☐	DHIS2 does not have such a feature yet. This is a planned feature.	
V2	Authentication	2.1.8	Verify that a password strength meter is provided to help users set a stronger password.	X	X	X	521	5.1.1.2	☑	☐	☐		
V2	Authentication	2.1.9	Verify that there are no password composition rules limiting the type of characters permitted. There should be no requirement for upper or lower case or numbers or special characters. [C6]	X	X	X	521	5.1.1.2	☑	☐	☐		
V2	Authentication	2.1.12	Verify that the user can choose to either temporarily view the entire masked password, or temporarily view the last typed character of the password on platforms that do not have this as native functionality.	X	X	X	521	5.1.1.2	☑	☐	☐		
V2	Authentication	2.2.3	Verify that secure notifications are sent to users after updates to authentication details, such as credential resets, email or address changes, logging in from unknown or risky locations. The use of push notifications - rather than SMS or email - is preferred, but in the absence of push notifications, SMS or email is acceptable as long as no sensitive information is disclosed in the notification.	X	X	X	620		☑	☐	☐	This is not implemented. This is a planned feature.	
V2	Authentication	2.4.3	Verify that if PBKDF2 is used, the iteration count SHOULD be as large as verification server performance will allow, typically at least 100,000 iterations. [C6]		X	X	916	5.1.1.2	☑	☐	☑		Pass
V2	Authentication	2.4.4	Verify that if bcrypt is used, the work factor SHOULD be as large as verification server performance will allow, typically at least 13. [C6]		X	X	916	5.1.1.2	☑	☐	☑		Pass
V2	Authentication	2.5.1	Verify that a system generated initial activation or recovery secret is not sent in clear text to the user. [C6]	X	X	X	640	5.1.1.2	☑	☐	☑	This is fixed now.	Should be a pass
V2	Authentication	2.5.5	Verify that if an authentication factor is changed or replaced, that the user is notified of this event.	X	X	X	304	6.1.2.3	☑	☐	☐		
V3	Session	3.3.4	Verify that users are able to view and log out of any or all currently active sessions and devices.		X	X	613	7.1	☑	☐	☐	This is not implemented. This is a planned feature.	
V3	Session	3.4.4	Verify that cookie-based session tokens use "__Host-" prefix (see references) to provide session cookie confidentiality.	X	X	X	16	7.1.1	☑	☐	☑	This is fixed in latest version.	Should be a pass
V3	Session	3.6.2	Verify that CSPs inform relying parties of the last authentication event, to allow RPs to determine if they need to re-authenticate the user.			X	613	7.2.1	☑	☐	☐		
V3	Session	3.7.1	Verify the application ensures a valid login session or requires re-authentication or secondary verification before allowing any sensitive transactions or account modifications.	X	X	X	778		☑	☐	☐	This is not implemented. This is a planned feature. We protect password change with brute force check. To change password you need to enter old password.	Partial pass.
V4	Access	4.2.2	Verify that the application or framework enforces a strong anti-CSRF mechanism to protect authenticated functionality, and effective anti-automation or anti-CSRF protects unauthenticated functionality.	X	X	X	352		☑	☐	☐	This is not implemented. This is a planned feature.	
V4	Access	4.3.1	Verify administrative interfaces use appropriate multi-factor authentication to prevent unauthorized use.	X	X	X	419		☑	☐	☐	DHIS2 supports 2-factor. This is configurable and up to the respective users/admins to decide if they want to use.	
V5	Validation	5.1.5	Verify that URL redirects and forwards only allow whitelisted destinations, or show a warning when redirecting to potentially untrusted content.	X	X	X	601		☑	☐	☐	This is not implemented. This is a planned feature.	
V5	Validation	5.2.1	Verify that all untrusted HTML input from WYSIWYG editors or similar is properly sanitized with an HTML sanitizer library or framework feature. [C5]	X	X	X	116		☑	☐	☑	This is fixed in latet versions.	Should be a pass now
V5	Validation	5.2.2	Verify that unstructured data is sanitized to enforce safety measures such as allowed characters and length.	X	X	X	138		☑	☐	☑	This is fixed in latet versions.	Should be a pass now
V5	Validation	5.2.5	Verify that the application protects against template injection attacks by ensuring that any user input being included is sanitized or sandboxed.	X	X	X	94		☑	☐	☑		Pass
V6	Cryptography	6.1.3	Verify that regulated financial data is stored encrypted while at rest, such as financial accounts, defaults or credit history, tax records, pay history, beneficiaries, or de-anonymized market or research records.		X	X	311		☑	☐	☐		N/A

			C1-C10 Top 10 Proactive Controls: https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering)						258/286	174/258	189/258		
V6	Cryptography	6.3.1	Verify that all random numbers, random file names, random GUIDs, and random strings are generated using the cryptographic module's approved cryptographically secure random number generator when these random values are intended to be not guessable by an attacker.		X	X	338		☑	☐	☑	This is fixed in latet versions.	Should be a pass now
V8	Data	8.1.1	Verify the application protects sensitive data from being cached in server components such as load balancers and application caches.		X	X	524		☑	☐	☐		
V8	Data	8.1.4	Verify the application can detect and alert on abnormal numbers of requests, such as by IP, user, total per hour or day, or whatever makes sense for the application.		X	X	770		☑	☐	☑	Partial support by protecting the most sensitive API endpoints with brute-force checks. We also encourage system admins to implement a proxy layer in front of their installations to prevent brute force and DOS attacks.	Pass
V8	Data	8.3.5	Verify accessing sensitive data is audited (without logging the sensitive data itself), if the data is collected under relevant data protection directives or where logging of access is required.		X	X	532		☑	☐	☑	DHIS2 have better audit functionality in newest versions.	Should be a pass now
V8	Data	8.3.6	Verify that sensitive information contained in memory is overwritten as soon as it is no longer required to mitigate memory dumping attacks, using zeroes or random data.		X	X	226		☑	☐	☐	Partially implemented in libs used. Planned task to do a deeper analysis	
V10	Malicious	10.2.4	Verify that the application source code and third party libraries does not contain time bombs by searching for date and time related functions.			X	511		☑	☐	☐		
V10	Malicious	10.2.5	Verify that the application source code and third party libraries does not contain malicious code, such as salami attacks, logic bypasses, or logic bombs.			X	511		☑	☐	☐	Work in progress to implement into CI pipeline.	
V10	Malicious	10.2.6	Verify that the application source code and third party libraries do not contain Easter eggs or any other potentially unwanted functionality.			X	507		☑	☐	☐	Work in progress to implement into CI pipeline.	
V12	Files	12.1.1	Verify that the application will not accept large files that could fill up storage or cause a denial of service attack.	X	X	X	400		☑	☐	☐	Currently being fixed as part of pentest, should be available in next version.	Passing soon
V12	Files	12.1.2	Verify that compressed files are checked for "zip bombs" - small input files that will decompress into huge files thus exhausting file storage limits.		X	X	409		☑	☐	☐	Currently being fixed as part of pentest, should be available in next version.	Passing soon
V12	Files	12.1.3	Verify that a file size quota and maximum number of files per user is enforced to ensure that a single user cannot fill up the storage with too many files, or excessively large files.		X	X	770		☑	☐	☐	Currently being fixed as part of pentest, should be available in next version.	Passing soon
V12	Files	12.2.1	Verify that files obtained from untrusted sources are validated to be of expected type based on the file's content.		X	X	434		☑	☐	☐	Currently being fixed as part of pentest, should be available in next version.	Passing soon
V12	Files	12.4.2	Verify that files obtained from untrusted sources are scanned by antivirus scanners to prevent upload of known malicious content.	X	X	X	509		☑	☐	☐	Currently being fixed as part of pentest, should be available in next version.	Passing soon
V13	API	13.2.3	Verify that RESTful web services that utilize cookies are protected from Cross-Site Request Forgery via the use of at least one or more of the following: triple or double submit cookie pattern (see https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet), CSRF nonces, or ORIGIN request header checks.	X	X	X	352		☑	☐	☐		
V13	API	13.2.4	Verify that REST services have anti-automation controls to protect against excessive calls, especially if the API is unauthenticated.		X	X	779		☑	☐	☑		Pass
V14	Config	14.2.5	Verify that an inventory catalog is maintained of all third party libraries in use. [C2]		X	X			☑	☐	☐		
V14	Config	14.2.6	Verify that the attack surface is reduced by sandboxing or encapsulating third party libraries to expose only the required behaviour into the application. ([C2](https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering))		X	X	265		☑	☐	☐	Currently being fixed as part of pentest, should be available in next version.	Passing soon
V14	Config	14.4.2	Verify that all API responses contain Content-Disposition: attachment; filename="api.json" (or other appropriate filename for the content type).	X	X	X	116		☑	☐	☐	Currently being fixed as part of pentest, should be available in next version.	Passing soon
V14	Config	14.5.4	Verify that HTTP headers added by a trusted proxy or SSO devices, such as a bearer token, are authenticated by the application.		X	X	306		☑	☐	☐		

Phase 3: Ranking spreadsheet

This spreadsheet shows the results of the ranking phase. The spreadsheet shows all 50 failed controls, and the comments and ranking given by the two participants.

			C1-C10 Top 10 Proactive Controls: https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering)						258/286	50/258			
			ASVS 4.0									Security engineer	Frontend
Section	**Name**	**Item**	**Description**	**L1**	**L2**	**L3**	**CWE**	**NIST**	**Applicable**	**Failed**		**Comments**	**Comments**
V1	Architecture	1.1.1	Verify the use of a secure software development lifecycle that addresses security in all stages of development. [C1]		X	X			☑	☐		Probably the most scary thing, if perhaps some attacker can commit some code into the repository, and the code gets redistributed through different channels. That's more on the development lifecycle, making sure that every commit is signed, use checksums for distributions. Rank: 5	I consider security a process more than an end state, so this is something we should do. Rank: 5.
V1	Architecture	1.1.2	Verify the use of threat modeling for every design change or sprint planning to identify threats, plan for countermeasures, facilitate appropriate risk responses, and guide security testing.		X	X	1053		☑	☐		Don't have any good formalizie process or checklists for doing this, but it's on a to-do list, to have a checklist and some basic thinking about what they are actually doing and implementing. Rank: 4	I'm not familiar with threat modeling processes. We have recently hired a security engineer that should provide guidance on how to integrate this into our processes. Rank: 3
V1	Architecture	1.1.5	Verify definition and security analysis of the application's high-level architecture and all connected remote services. [C1]		X	X	1059		☑	☐		Rank: 4	This is on the implementation side, but we should provide clear guidance on what the sysadmins MUST and MUST NOT do in terms of maintaining a secure system. Rank: 4
V1	Architecture	1.1.7	Verify availability of a secure coding checklist, security requirements, guideline, or policy to all developers and testers.		X	X	637		☑	☐		Rank: 4	This is something that we would benefit from. Rank: 3
V1	Architecture	1.2.3	Verify that the application uses a single vetted authentication mechanism that is known to be secure, can be extended to include strong authentication, and has sufficient logging and monitoring to detect account abuse or breaches.		X	X	306		☑	☐		Kind of have this. Have this on protection for password recovery. Was possible to do a limited number of passworwd recoveries. Falls into brutec force and rate limiting. Falls into account locking, notifying the user. This part could be improved upon. The basic one-factor and two-factor is pretty standard stuff, using Spring security and using known practices. Currently, not possible to do an ulimited number of recoveries. Not a total fail, but the basics are in place. Rank: 5 (if this was a hard fail). It is a broad security requirement. The description could be broken into more parts. The first part of the sentence is the most important thing.	Must implement. It is just a matter of time until the first breach happens as a result of this. Rank: 5
V1	Architecture	1.4.5	Verify that attribute or feature-based access control is used whereby the code checks the user's authorization for a feature/data item rather than just their role. Permissions should still be allocated using roles. [C7]		X	X	275		☑	☐			
V1	Architecture	1.6.2	Verify that consumers of cryptographic services protect key material and other secrets by using key vaults or API based alternatives.		X	X	320		☑	☐		Since the encryption they re doing is symmetric encyption is very limited, and there are few settings to configure. Looked at implementation and its not very good. Having the option to hook into KeyVault, should be an option. Rank: 4	
V1	Architecture	1.6.3	Verify that all keys and passwords are replaceable and are part of a well-defined process to re-encrypt sensitive data.		X	X	320		☑	☐		Task to do a reharshing of the password, having a different kind of password hashes. You pre-pend the hash with the kind of hash you are using, so you can look ito the database and do a reharashing if its necessary. Rank: 4	
V1	Architecture	1.6.4	Verify that symmetric keys, passwords, or API secrets generated by or shared with clients are used only in protecting low risk secrets, such as encrypting local storage, or temporary ephemeral uses such as parameter obfuscation. Sharing secrets with clients is clear-text equivalent and architecturally should be treated as such.		X	X	320		☑	☐			
V1	Architecture	1.7.2	Verify that logs are securely transmitted to a preferably remote system for analysis, detection, alerting, and escalation. [C9]		X	X			☑	☐			
V1	Architecture	1.12.2	Verify that user-uploaded files - if required to be displayed or downloaded from the application - are served by either octet stream downloads, or from an unrelated domain, such as a cloud file storage bucket. Implement a suitable content security policy to reduce the risk from XSS vectors or other attacks from the uploaded file.		X	X	646		☑	☐			

			C1-C10 Top 10 Proactive Controls: https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering)						258/286	50/258			
V2	Authentication	2.1.7	Verify that passwords submitted during account registration, login, and password change are checked against a set of breached passwords either locally (such as the top 1,000 or 10,000 most common passwords which match the system's password policy) or using an external API. If using an API a zero knowledge proof or other mechanism should be used to ensure that the plain text password is not sent or used in verifying the breach status of the password. If the password is breached, the application must require the user to set a new non-breached password. [C6]	X	X	X	521	5.1.1.2	☑	☐	Don't have password breach device, but probably should. Api's they could subsribe to, a lot of companies making money of hosting this bridge password databases. Becoming quite popular, this service for doing this. How this is applied to an open source system, not quite sure, beacuse the service sells the service as a paid subscription, so not sure how this can be done in a free service. Maybe you can add some configuration, so it is possible for the guys that host the system that add it if they want, make it some pluggible system ouf of it. Could be an extra security measure for people that want it. Doesn't see it as a major big security thing, but people could of course use bad passwords. Rank: 3	We should match against known breached passwords anytime a password is set or updated. Rank: 4
V2	Authentication	2.1.8	Verify that a password strength meter is provided to help users set a stronger password.	X	X	X	521	5.1.1.2	☑	☐	Helpful for a user interface, its a helpful feedback. Rank: 1/2	Nice to have, though I have not researched if this actually affects anything if not enforced. Rank: 1
V2	Authentication	2.1.9	Verify that there are no password composition rules limiting the type of characters permitted. There should be no requirement for upper or lower case or numbers or special characters. [C6]	X	X	X	521	5.1.1.2	☑	☐	Would have liked to see it go away. Rank: 2	I have a pet peeve not being allowed full unicode charset when setting passwords, so personally I find this important. Rank: 3
V2	Authentication	2.1.12	Verify that the user can choose to either temporarily view the entire masked password, or temporarily view the last typed character of the password on platforms that do not have this as native functionality.	X	X	X	521	5.1.1.2	☑	☐	User interface concern. Rank: 1/2	Would be handy, not the highest priority we have. Rank: 1
V2	Authentication	2.2.3	Verify that secure notifications are sent to users after updates to authentication details, such as credential resets, email or address changes, logging in from unknown or risky locations. The use of push notifications - rather than SMS or email - is preferred, but in the absence of push notifications, SMS or email is acceptable as long as no sensitive information is disclosed in the notification.	X	X	X	620		☑	☐	More serious. Have been working on bruce force protection. When suspicious activity hapens on your accounts, when someone trying to login from different places, it changes very fast, you can't physically change that fast. Detecting strange behaviors, and doing something with it, that information - sending notifications Rank: 4	Must have. Currently if a user's password is changed without her knowledge she will not be aware until the next use, which may be too late. Rank: 4
V2	Authentication	2.4.3	Verify that if PBKDF2 is used, the iteration count SHOULD be as large as verification server performance will allow, typically at least 100,000 iterations. [C6]		X	X	916	5.1.1.2	☑	☐		
V2	Authentication	2.4.4	Verify that if bcrypt is used, the work factor SHOULD be as large as verification server performance will allow, typically at least 13. [C6]		X	X	916	5.1.1.2	☑	☐		
V2	Authentication	2.5.1	Verify that a system generated initial activation or recovery secret is not sent in clear text to the user. [C6]	X	X	X	640	5.1.1.2	☑	☐	Skip	Embarrassing that we fail on this. Rank: 5
V2	Authentication	2.5.5	Verify that if an authentication factor is changed or replaced, that the user is notified of this event.	X	X	X	304	6.1.2.3	☑	☐	Related to the other notification issue Rank: 4	Ties into 2.5.1, we should notify the user when sensitive information on her account changes. Rank: 5
V3	Session	3.3.4	Verify that users are able to view and log out of any or all currently active sessions and devices.		X	X	613	7.1	☑	☐	It's lacking. Fairly easy to do on the backend, but a lot of work on the front-end. Comes with the part on getting notification and seeing a log of when you logged in and out. Rank 4	This would be a very useful tool to provide for users. We should be giving the user enough tools to manage their own security as it is the first line of defence. Rank: 3
V3	Session	3.4.4	Verify that cookie-based session tokens use "__Host-" prefix (see references) to provide session cookie confidentiality.	X	X	X	16	7.1.1	☑	☐	Doing it as part of a pentest they have. This is an ongoing thing, important. Rank: 4	Not fully aware of the consequences, but it is a trivial thing to fix and we should follow best practice on this. 3
V3	Session	3.6.2	Verify that CSPs inform relying parties of the last authentication event, to allow RPs to determine if they need to re-authenticate the user.			X	613	7.2.1	☑	☐		Unsure of the consequences of this omission. Rank: 2
V3	Session	3.7.1	Verify the application ensures a valid login session or requires re-authentication or secondary verification before allowing any sensitive transactions or account modifications.	X	X	X	778		☑	☐	Don't do it. Steal your session, either physically into your machine or, some other kind of trickery. Should prevent the attacker from doing that. Rank: 4	There is no need for a user to be at full privileges at all times, and could easily escalate his active privileges through a secondary password entry when dealing with sensitive operations. Rank: 4
V4	Access	4.2.2	Verify that the application or framework enforces a strong anti-CSRF mechanism to protect authenticated functionality, and effective anti-automation or anti-CSRF protects unauthenticated functionality.	X	X	X	352		☑	☐	Currently are not doing this, it's on the latest pentest list. It's possible to do, but other priorities. Rank: 3	Important. We are vulnerable to this throughout. Rankl: 5

			C1-C10 Top 10 Proactive Controls: https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering)						258/286	50/258			
V4	Access	4.3.1	Verify administrative interfaces use appropriate multi-factor authentication to prevent unauthorized use.	X	X	X	419		☑	☐		Do have two-factor, but it is not enforced on admin accounts. Its hard to enforce. It's more like a end-user UI decision, more than a technical decision. It can make some users annoyed. It's not super important. Rank: 2	Important, especially if we do not fix the CSRF-problems ASAP. Rank: 5
V5	Validation	5.1.5	Verify that URL redirects and forwards only allow whitelisted destinations, or show a warning when redirecting to potentially untrusted content.	X	X	X	601		☑	☐		You have to implement it into the client, if you want to really protect against it. Don't do server-side redirection in that sence. If you are not logged in, you are redirected else. Rank: 4 (but not sure if applicable).	Nice to have, I do not consider it high priority. Rank: 3
V5	Validation	5.2.1	Verify that all untrusted HTML input from WYSIWYG editors or similar is properly sanitized with an HTML sanitizer library or framework feature. [C5]	X	X	X	116		☑	☐		On the pentest, possible to add some javascript into the form and then an admin user would read it, and it would get executed, and could do bad things. Hard fail. Rank: 5	Must implement, we have open issues on this that are being worked on. Rank: 4
V5	Validation	5.2.2	Verify that unstructured data is sanitized to enforce safety measures such as allowed characters and length.	X	X	X	138		☑	☐		Similar to 5.2.1. Related. Rank: 5	Should be provided on a service level layer (backend). Ranl: 3
V5	Validation	5.2.5	Verify that the application protects against template injection attacks by ensuring that any user input being included is sanitized or sandboxed.	X	X	X	94		☑	☐			
V6	Cryptography	6.1.3	Verify that regulated financial data is stored encrypted while at rest, such as financial accounts, defaults or credit history, tax records, pay history, beneficiaries, or de-anonymized market or research records.		X	X	311		☑	☐			
V6	Cryptography	6.3.1	Verify that all random numbers, random file names, random GUIDs, and random strings are generated using the cryptographic module's approved cryptographically secure random number generator when these random values are intended to be not guessable by an attacker.		X	X	338		☑	☐		Been trying to check if secure random is being used, haven't gotten an overview over it yet. This is super important. The rank depends on the context. Rank; 4	Should probably use something like UUIDv4 over our naively generated identificators, but is a considerable change. Rank: 3
V8	Data	8.1.1	Verify the application protects sensitive data from being cached in server components such as load balancers and application caches.		X	X	524		☑	☐			
V8	Data	8.1.4	Verify the application can detect and alert on abnormal numbers of requests, such as by IP, user, total per hour or day, or whatever makes sense for the application.		X	X	770		☑	☐		Been working on this. If there is an ongoing attack, it's a lot of system don't have protection for this at all. If you don't have any systems to detect this, you are basically, it's hard to look through logs and a lot of extra work to figure out what is going on. Rank 4	This could (and should) be handled on the implementers side on a server level, as that is where the reverse proxy for DHIS2 lives, and regardless, sysadmins should be aware of what requests are hitting there server and not DHIS2 specifically. We could provide an interface that only monitors requests that get through to DHIS2, but I find it of dubious value
V8	Data	8.3.5	Verify accessing sensitive data is audited (without logging the sensitive data itself), if the data is collected under relevant data protection directives or where logging of access is required.		X	X	532		☑	☐		This goes into audit stuff, haven't been looking much at it yet. If this is lacking, then Rank: 3/4	Useful for security forensic activities. Rank: 3.
V8	Data	8.3.6	Verify that sensitive information contained in memory is overwritten as soon as it is no longer required to mitigate memory dumping attacks, using zeroes or random data.		X	X	226		☑	☐		Trick to test, tricky to verify that your protection actually works. Rank: 3/4	Unsure how this could be done, but it should at least be investigated. Ranl: 2.
V10	Malicious	10.2.4	Verify that the application source code and third party libraries does not contain time bombs by searching for date and time related functions.			X	511		☑	☐		Could do more third-party checking verifications, right now its not much to do about it. Rank: 3	
V10	Malicious	10.2.5	Verify that the application source code and third party libraries does not contain malicious code, such as salami attacks, logic bypasses, or logic bombs.			X	511		☑	☐		More on the total development lifecycle thing, and therefore important. Rank 4.	We rely on CVE reports on libraries to verify if there are any known problems. We use those reports to automatically update the libraries. We need to do this more systematically and preferably automated. Rank: 4
V10	Malicious	10.2.6	Verify that the application source code and third party libraries do not contain Easter eggs or any other potentially unwanted functionality.			X	507		☑	☐		Depends on the kind of Easter eggs you use. Rank 4	We rely on CVE reports on libraries to verify if there are any known problems. We use those reports to automatically update the libraries. We need to do this more systematically and preferably automated. Rank: 4
V12	Files	12.1.1	Verify that the application will not accept large files that could fill up storage or cause a denial of service attack.	X	X	X	400		☑	☐		Imporant, but hard to do in the code. If you allow streaming uploads. Rank: 4	This is something we should have. Rank: 3

			C1-C10 Top 10 Proactive Controls: https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering)						258/286	50/258			
V12	Files	12.1.2	Verify that compressed files are checked for "zip bombs" - small input files that will decompress into huge files thus exhausting file storage limits.		X	X	409		☑	☐		No check for that. Rank: 3	This is something we should have. Rank: 3
V12	Files	12.1.3	Verify that a file size quota and maximum number of files per user is enforced to ensure that a single user cannot fill up the storage with too many files, or excessively large files.		X	X	770		☑	☐		Really important. Rank: 4	This is something we should have. Rank: 3
V12	Files	12.2.1	Verify that files obtained from untrusted sources are validated to be of expected type based on the file's content.		X	X	434		☑	☐			
V12	Files	12.4.2	Verify that files obtained from untrusted sources are scanned by antivirus scanners to prevent upload of known malicious content.	X	X	X	509		☑	☐		Related to 12.1.1. Don't know how this mechanism for adding apps is done. If it's possible to upload an application that can run any time of code, then that is a problem. Not sure if it is possible to 100% fix that kind of problem. That is pasically what google play is trying to do. Rank: 5	This should be done on the implementers side, we can provide guidance that they _should_ be doing it. Rank: 1
V13	API	13.2.3	Verify that RESTful web services that utilize cookies are protected from Cross-Site Request Forgery via the use of at least one or more of the following: triple or double submit cookie pattern (see https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet), CSRF nonces, or ORIGIN request header checks.	X	X	X	352		☑	☐		On the pentest list. Rank: 4	Must do, have an open issue on this. Rank: 5
V13	API	13.2.4	Verify that REST services have anti-automation controls to protect against excessive calls, especially if the API is unauthenticated.		X	X	779		☑	☐		Related to red-limiting that is being discussed. Rank: 4	Should do, though we do not have an unauthenticated API. Rank: 3
V14	Config	14.2.5	Verify that an inventory catalog is maintained of all third party libraries in use. [C2]		X	X			☑	☐			
V14	Config	14.2.6	Verify that the attack surface is reduced by sandboxing or encapsulating third party libraries to expose only the required behaviour into the application. ([C2](https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering))		X	X	265		☑	☐		This applies if you are have third-party apps in the app store. Sandboxing is commonly preventing third-party apps, especially apps that are totally not checked. Rank 4	We have started doing some of this on the frontend side, we could be doing more of it. Rank: 3
V14	Config	14.4.2	Verify that all API responses contain Content-Disposition: attachment; filename="api.json" (or other appropriate filename for the content type).	X	X	X	116		☑	☐		Not sure what you can abuse with this. Can trick someone to download something else. Not sure.	Would a handy addition to make the API more predictable. Rank: 3
V14	Config	14.5.4	Verify that HTTP headers added by a trusted proxy or SSO devices, such as a bearer token, are authenticated by the application.		X	X	306		☑	☐		Important. Rank: 3/4	I don't know the impact of this, but verifying the HTTP headers on the application boundaries seems wise. Rank: 3