Design Proposal
Dutch Forensic Reporting System

University of Essex - Secure Software Development Module

Nkosilathi Tauro, Abdulahi Alihu Ngamjeh, Francis Muwalo, Ales Tekavcic

Total word count - 879

12.06.2023

Background

The National Cyber Security Centre (NCSC) of the Netherlands plays a vital role in safeguarding critical ICT systems and infrastructure across various organisations (Government of the Netherlands, N.D.), contributing to the resilience of the country's digital ecosystem. However, ICT systems are susceptible to vulnerabilities and flaws (Šilić, Krolo and Delač, 2010), which can be exploited by malicious actors, posing security risks. With this in mind, the NCSC aims to develop a secure repository of the Dutch Police and our proposed system enables efficient reporting and management of ICT system flaws, promoting collaboration between Cyberdetectives and the general public, resulting in enhanced cybersecurity and improved ICT system resilience.

Proposed System

The Dutch Forensic Reporting System aims to provide a comprehensive reporting service for identifying flaws in ICT systems across various organisations. It caters to two primary user roles: Cyberdetectives (Dutch Police employees) and the general public. Cyberdetectives will handle and forward reports to relevant stakeholders, while certain privileged detectives will have additional access rights (See user roles, Page 3). The general public can submit reports anonymously or by providing their contact details. The system's primary goals are to prevent cyberattacks, enhance website functionalities, and ensure public safety. (Figure 3: Use Case diagram)

Assumptions

- The system will maintain high availability, operating 24/7 with minimal downtime for maintenance and upgrades.
- Users will have a reliable and secure internet connection to access the system.
- The data entered into the system will be accurate and reliable.

Functional Requirements

- Enable event logging and monitoring of IP addresses and login attempts.
- Support CRUD operations for reports.
- Implement data encryption throughout the entire data lifecycle.

Non-Functional Requirements

- Provide data anonymisation, consent management, and data retention policies to ensure compliance with GDPR. Users can request data deletion when relevant (Wolford, n.d.).
- Implement a robust user authentication mechanism to secure system access.
- Ensure a user-friendly interface and intuitive navigation for Cyberdetectives and the general public.

System Requirements

- **CPU**: Intel Xeon E5-2670 v2 Or AMD Equivalent (1 Core)
- **RAM**: 3GB
- **Minimum Disk Space**: 512MB (PythonAnywhere, N.D.)

User Roles

Cyberdetective Admin:
- View system/event logs
- Create CyberDetective accounts
- View all created reports.
- System account required

Cyberdetective:
- Create and submit reports
- Delete/Update their created reports
- System account required

General Public:
- Create and submit reports
- No system account required

Design Decisions

We use the MVC design pattern for our application. It divides the system into three microservices to handle scalability. The Backend and database microservices represent the Model, the Frontend microservice handles the View, and the Controller manages data flow and interactions. This pattern improves code organisation, reusability, and collaboration during agile development (Yu and Yang, 2019).

Each microservice has subcomponents, like the Backend System with the user service (Figure 1). They run in separate Docker containers, allowing flexible scaling based on demand (Singh and Peddoju, 2017). This approach optimises deployment costs and ensures availability during high website traffic.

System Architecture

Frontend

- Takes input from the initiator; this could be a General public or Cybersecurity employee.
- Provides appropriate user interface - according to the user roles.
- Responsible for visual representation of the system.
- Interacts with a browser.
- Provides feedback to the user.

Backend

- Establishes a secure connection with the Database and Frontend.
- Implement the CRUD functionality required via the REST API.
- Implements data sanitation.
- Implement the data validation received as input from forms and input fields.
- Actions taken (logins, reports) will be logged.

Security Risks and Mitigations

Following the OWASP secure system design recommendations, we have identified and addressed six key security risks to ensure the integrity and confidentiality of our system.

A03:2021 – Injection:

- Sanitising data through proper validation and parameterised API calls mitigates the risk of injection attacks (OWASP, 2021). Django handles these validation operations (Django, N.D), employing techniques such as escaping and encoding to prevent unwanted user operations.

A04:2021 – Insecure Design:

- A secure design considers project goals and business risks, encompassing the entire software development lifecycle. Incorporating secure design patterns

throughout the process helps avoid insecure design and ensures the overall security of the system (OWASP, 2021).

A06:2021 – Vulnerable and Outdated Components:

- Regularly updating software dependencies and being aware of library versions helps protect against vulnerabilities associated with outdated components. Utilising dependency tools like retire.js or dependency-check (Python) aids in identifying and addressing potential risks (OWASP, 2021).

A01:2021 – Broken Access Control:

- Implementing effective access validation mechanisms prevents unauthorised access and modification of data. Each user operation should undergo validation processes to ensure proper access controls (Clancy, 2022).

A09:2021 – Security Logging and Monitoring Failures:

- Monitoring API actions and implementing alert systems for multiple requests helps mitigate DDoS attacks (Embroker, 2023). Cloud-based deployment provides resilience against server crashes, ensuring uninterrupted service availability. Additionally, utilising Django's built-in Admin feature for privileged access enhances security logging and monitoring capabilities (Django, N.D).

A02:2021 – Cryptographic Failures:

- Employing trusted authentication libraries like Django's Default Auth provider (Django, N.D) ensures the secure handling of sensitive data. Enforcing TLS across the application guarantees the encryption of data transmitted between the Frontend and Backend, safeguarding against cryptographic failures (OWASP, 2021).

By addressing these security risks and implementing appropriate measures, we enhance the reliability, confidentiality, and integrity of our system, mitigating potential threats and ensuring a secure software environment.

Tools and Libraries

Created with Python v3.11.3.

Database	SQLite v3.42.0
Framework	Django v4.2.1
Frontend Markup/Styling	HTML5, CSS, Django-tailwind v3.5.0/Bootstrap v5.3.x
Events	Python Logging module v0.4.9.6, SQLite3
Authentication	Django OAuth Toolkit v3.1+
Linter	Pylint v2.17.4
Testing	Unittest2 v1.1.0
Deployment	PythonAnywhere
Containers	Docker
Version Control	Git v2.40, Github
Additional Libraries	System libraries, DateTime v5.1, cryptography v40.0.2, regex v2023.5.5
UML Tools	Visual Paradigm, Mermaid v10.1.0

UML Diagrams

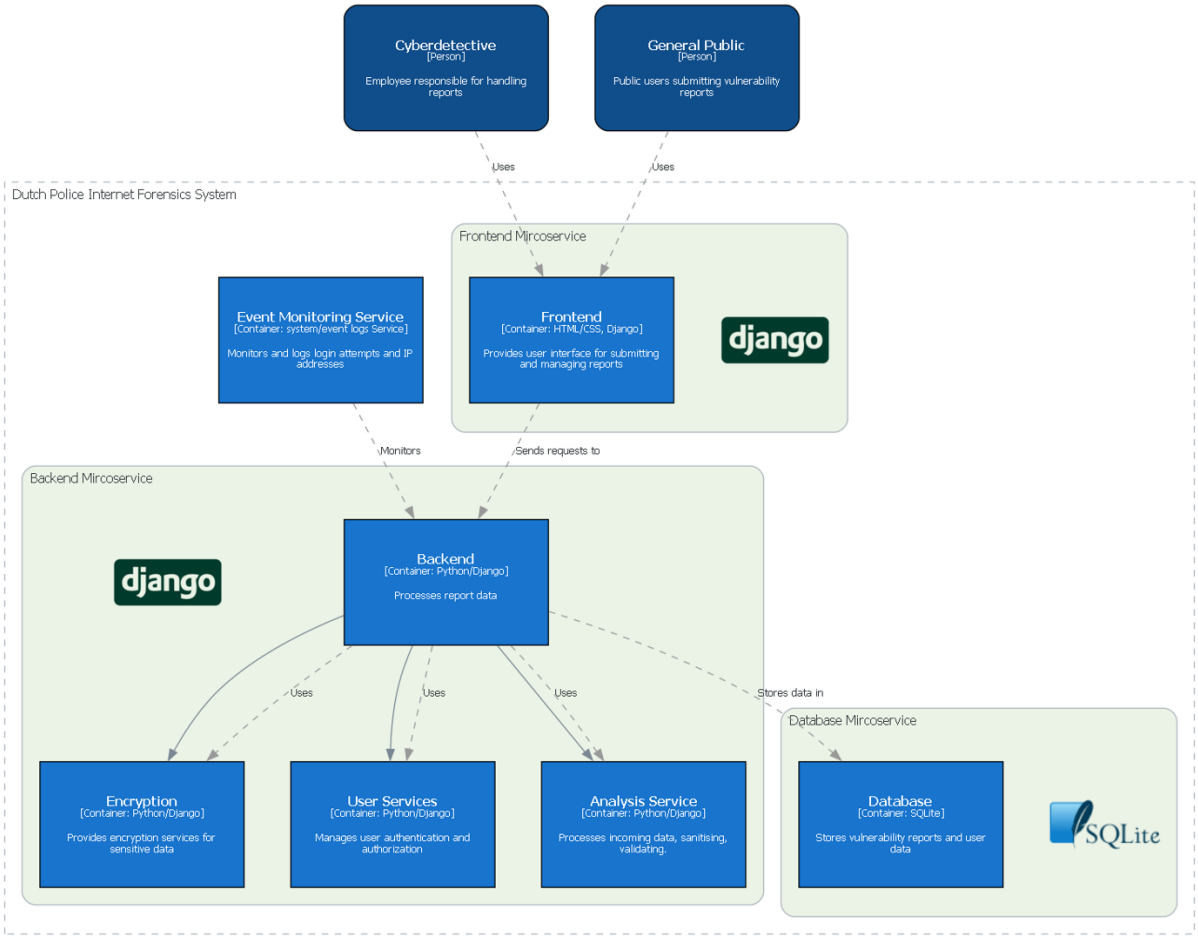

Figure 1: Architecture diagram for The Dutch Forensic Reporting System

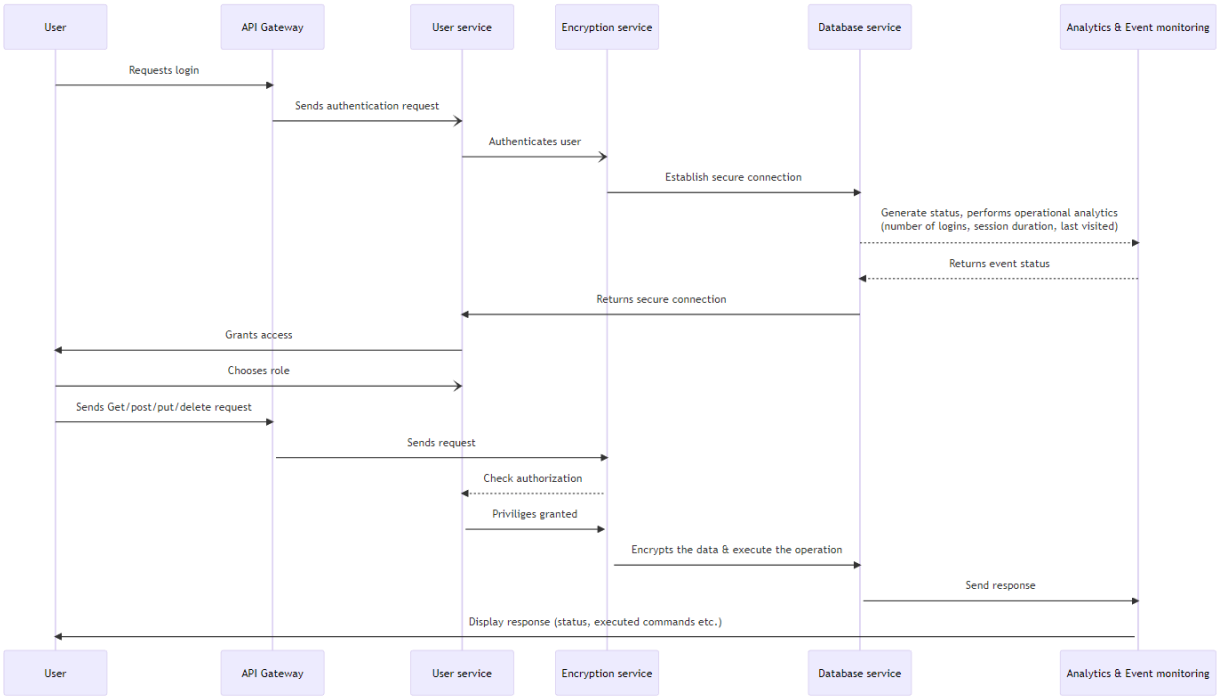

Figure 2: Sequence diagram showing the login flow for an Admin user

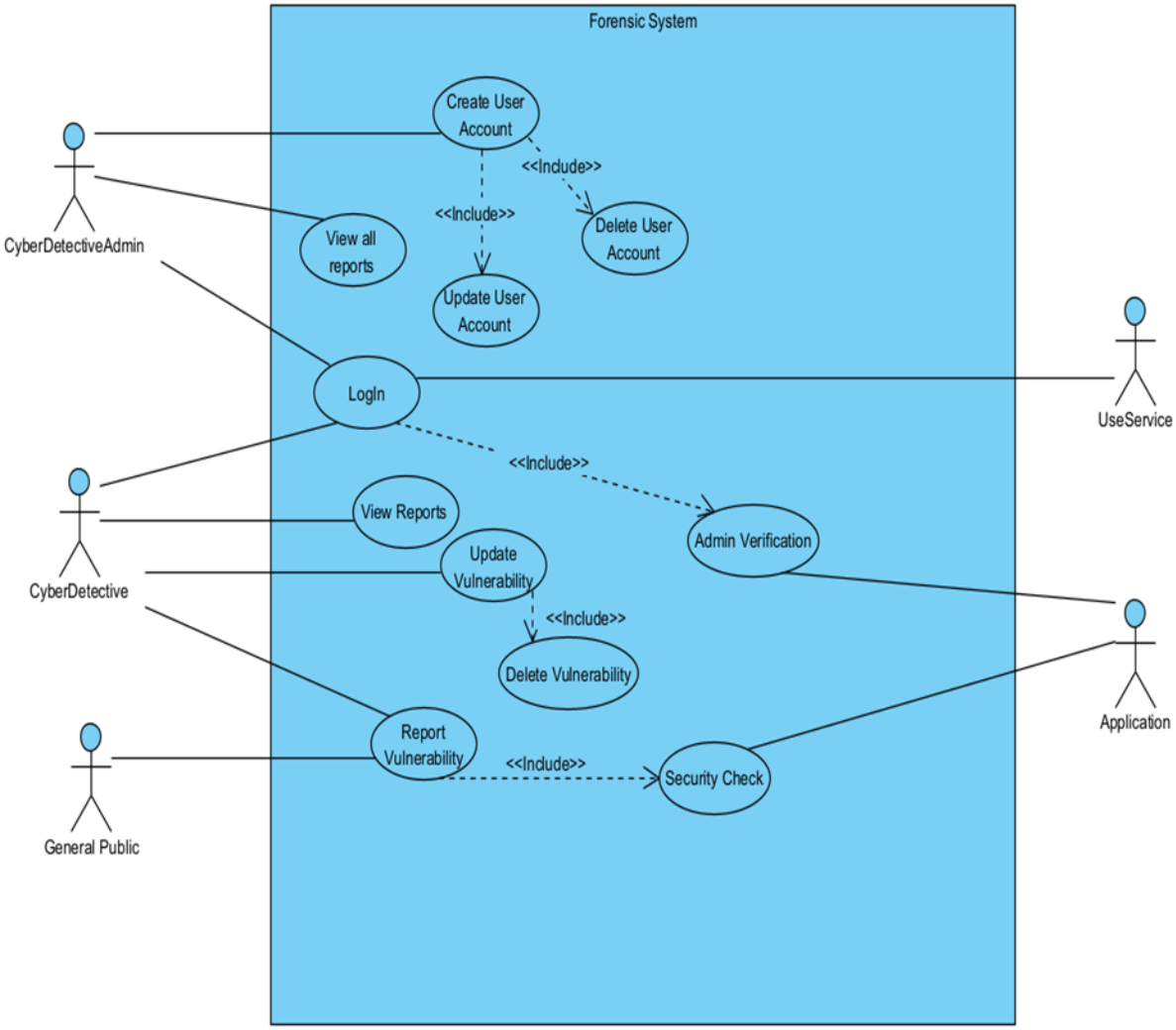

Figure 3: Use Case diagram for the Dutch Forensic Reporting System

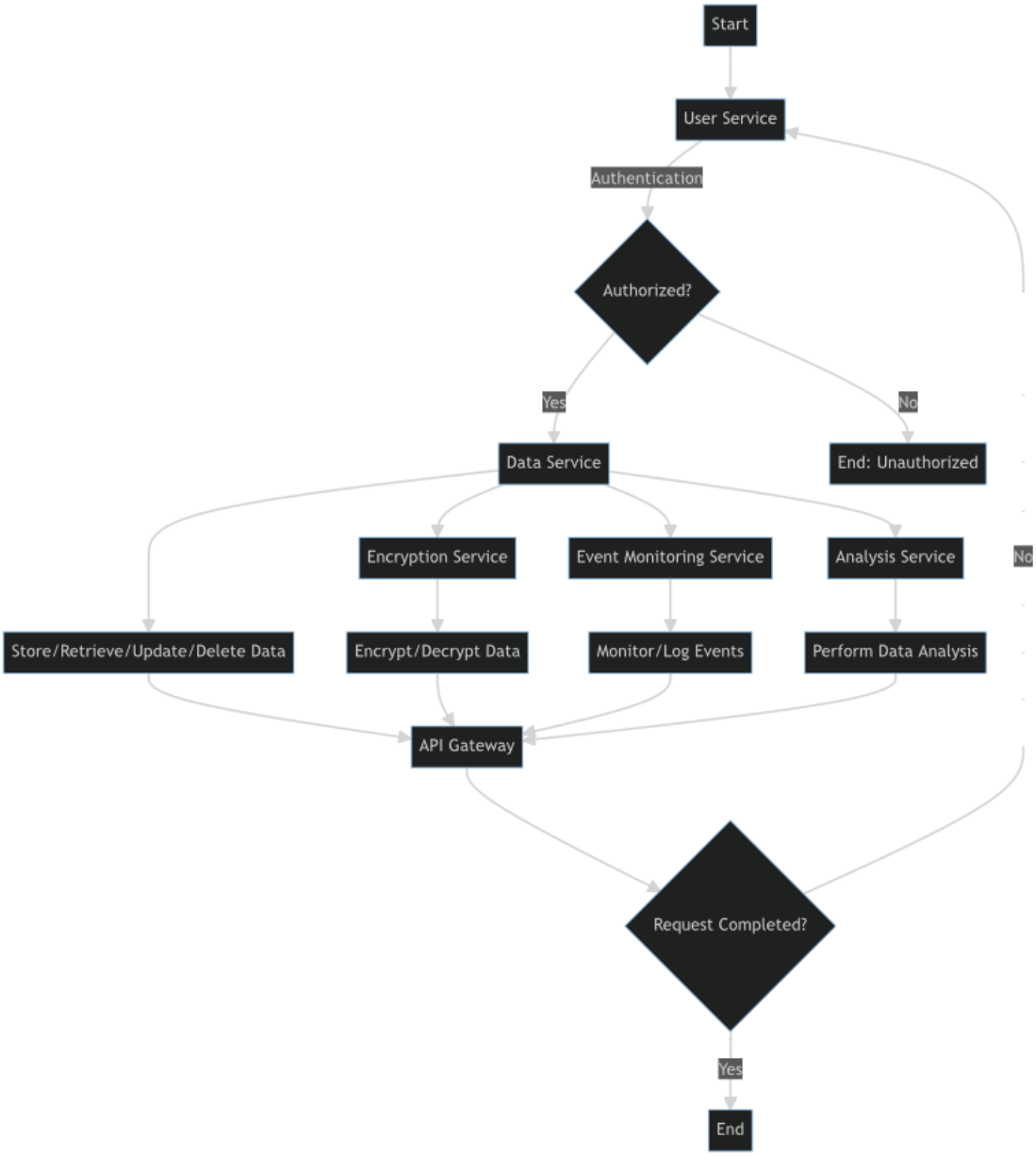

Figure 4: Activity diagram showing a Cyberdetective logging in and creating a report.

References:

Clancy, R., (2022). What Is Broken Access Control Vulnerability, and How Can I Prevent It? EC Council Cybersecurity Exchange. Available from: https://www.eccouncil.org/cybersecurity-exchange/web-application-hacking/broken-access-control-vulnerability/ [Accessed 20 May 2022]

Django, (N.D). Security In Django. Available from: https://docs.djangoproject.com/en/4.2/topics/security/ [Accessed 29 May 2023]

Django, (N.D). The Django admin site. Available from: https://docs.djangoproject.com/en/4.2/ref/contrib/admin/ [Accessed 29 May 2023]

Embroker, (2023). How to Prevent DDoS Attacks. Blog Risk Management. Available from: https://www.embroker.com/blog/how-to-prevent-ddos-attacks/ [Accessed 20 May 2023]

Government of the Netherlands, (N.D.). National Cyber Security Centre. Available from: https://english.ncsc.nl/about-the-ncsc [Accessed 27 May 2023]

OWASP, (2021). A06:2021 – Vulnerable and Outdated Components.OWASP Top 10:2021. Available from: https://owasp.org/Top10/A06_2021-Vulnerable_and_Outdated_Components/ [Accessed 20 May 2023]

OWASP, (2021). A02:2021 – Cryptographic Failures.OWASP Top 10:2021. Available from: https://owasp.org/Top10/A02_2021-Cryptographic_Failures/ [Accessed 29 May 2023]

OWASP, (N.D.). Encode and Escape Data. OWASP Proactive Controls. Available from: https://owasp-top-10-proactive-controls-2018.readthedocs.io/en/latest/c4-encode-escape-data.html [Accessed 20 May 2023]

PythonAnywhere official website, (N.D.). Plans and Pricing. Available from: https://www.pythonanywhere.com/pricing/ [Accessed 22 May 2023]

PythonAnywhere Q&A, (N.D.). RAM Limits. Available from https://help.pythonanywhere.com/pages/RAMLimit/ [Accessed 22 May 2023]

Šilić, M., Krolo, J. and Delač, G. (2010). Security vulnerabilities in modern web browser architecture. [online] IEEE Xplore. Available at: https://ieeexplore.ieee.org/document/5533657 [Accessed 10 Jun. 2023].

Singh, V. and Peddoju, S.K. (2017). Container-based microservice architecture for cloud applications. 2017 International Conference on Computing, Communication and Automation (ICCCA). doi: https://doi.org/10.1109/ccaa.2017.8229914.

University of Essex Online, (2023). Lecturecasts. [Accessed 20 May 2023]

Wolford, B. (N.D.). What is GDPR, the EU's new data protection law? GDPR.EU. Available from: https://gdpr.eu/what-is-gdpr/ [Accessed 27 May 2023]

Yu, Q. and Yang, W.-N. (2019). The Analysis and Design of System of Experimental Consumables Based on Django and QR code. IEEE Xplore. Available at doi: https://doi.org/10.1109/iicspi48186.2019.9095914.