Adrian Jacobsen Moen, Askil Amundøy Olsen,
Karoline Moe Arnesen

PLED (Pentesting Lab Environment Database)

Bachelor's project in IT-Operations and Information Security and Programming [Games|Applications]
Supervisor: Erik Hjelmås

May 2019

Bachelor's project

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

Sammendrag av Bacheloroppgaven

Tittel:	**PLED (Pentesting Lab Environment Database)**
Dato:	20.05.2019
Deltakere:	Adrian Jacobsen Moen(BITSEC) Askil Amundøy Olsen(BITSEC) Karoline Moe Arnesen(BPROG)
Veiledere:	Erik Hjelmås
Oppdragsgiver:	Norwegian Cyber Range - Danny Lopez Murillo
Kontaktperson:	Adrian J Moen, adrianjm@stud.ntnu.no, 45884608
Nøkkelord:	Cyber sikkerhet, API, Applikasjoner, Sårbare systemer, Sårbarehetsdatabase
Antall sider:	75
Antall vedlegg:	6
Tilgjengelighet:	Åpen

Sammendrag:

PLED er en database med en tilhørende REST API som tilbyr virtuelle plattformer filer som er nødvendig å starte cyber sikkerhets scenarioer igjennom PEMA (*Pentesting Exercise Management Application*). PLED kan lagre et mangold av forskjellige cyber sikkerhets relaterte filer, og databasen kan ta imot spørringer via REST API-en. I tillegg til fillagring tilbys et brukergrensesnitt mot databasen hvor en administrator eller instruktør kan laste opp, laste ned, slette og endre metadata på filer. Hovedformålet med PLED er å danne en støtte-plattform til PEMA der PLED tilbyr applikasjoner og andre filer som kan rulles ut i fullstendige lab øvelser via PEMA. Med et såpass stort mangfold av filer som kan hentes av PEMA, vil øvelsene kunne være med på å forberede studenter på virkelighetsnære cyber sikkerhet scenarioer. I tillegg blir det i stor grad utviklet sammen med studenter, i form av bachelor oppgaver. PEMA er en modulær skalerbar virtualiseringsplattform, der hensikten er å deploye virtuelle cyber scenarioer for bruk i etisk hacking, penetrasjonstesting, cyber-sikkerhetskonkurranser og liknende. PEMA er ment å gi instruktører et grensesnitt for å generere, deploye og overvåke cybersikkerhet-øvelser som studenter skal dra nytte av. Grunnet størrelsen av PEMA prosjektet, vil PLED bli utviklet til dels uavhengig av PEMA, det vil derfor være vanskelig å fullstendig integrere tjenestene med hverandre. Derfor vil PLED i tillegg lage en enkel måte å rulle ut PLED infrastrukturen ved hjelp av OpenStack Heat. I fremtiden er planen å integrere prosjektene til ett prosjekt.

Summary of Graduate Project

Title:	**PLED (Pentesting Lab Environment Database)**
Date:	20.05.2019
Authors:	Adrian Jacobsen Moen(BITSEC) Askil Amundøy Olsen(BITSEC) Karoline Moe Arnesen(BPROG)
Supervisor:	Erik Hjelmås
Employer:	Norwegian Cyber Range - Danny Lopez Murillo
Contact Person:	Adrian J Moen, adrianjm@stud.ntnu.no, 45884608
Keywords:	Cyber security, API, Applications, Vulnerable systems, Vulnerability-database
Pages:	75
Attachments:	6
Availability:	Open

Abstract:

PLED is a database and a REST API intended to provide educational platforms the files needed to launch educational cyber security scenarios through PEMA (*Pentesting Exercise Management Application*). PLED is capable of storing a wide range of cyber security related files, and the database can be queried using a REST API. Additionally, to storing files, an interface to the database is provided, where applications and files can be added, retrieved, modified, deleted and downloaded. PLEDs main purpose is to supply PEMA with applications and other cyber security related files for creation of fully functioning lab exercise environments. With such a wide range of files ready for retrieval via PEMA, these systems will help prepare students for real life cyber security scenarios. It's also being developed by involving students and their bachelor theses. PEMA is made out to be a modular scalable virtualization platform, with its purpose to deploy virtual cyber scenarios for use in ethical hacking, penetration testing and cyber security competitions. It will provide instructors an interface to create, deploy, log, and submit cyber security exercises, readily available for students to utilize. Due to the size of the PEMA project, PLED is independently developed side by side, and for that reason the PLED project will also provide a way to deploy the PLED infrastructure with OpenStack Heat. In the future the plan is to integrate these projects to one project.

Preface

This is a bachelor thesis written in 4 and a half months supervised by Norwegian University of Science and Technology and key individuals from the Norwegian Cyber Range. PLED was an idea spawned from PEMA, and the assignment was adapted into an assignment that accounted for both the IT-Operations and Information Security bachelor study and the Programming bachelor study. It became a very relevant assignment for all included parties, and hopefully it will see the light of day when the project is handed over to capable hands. It was a great learning experience and the group is thankful it was made possible.

The group would like to extend our thanks to everyone that supported PLED and contributed to is throughout the process. Especially our supervisor Erik Hjelmås, for being almost always available, doing thorough reviews of our report, and keeping a keen eye on the process made each week.

Additionally, a thanks to Danny Lopez who worked extremely hard in order to realize PEMA and PLED with the help of the NCR.

Finally, special thanks to Basel Katt and Lars Erik Pedersen for very valuable inputs and suggestions that was vitally important for the success of this project.

Contents

List of Figures

List of Tables

Listings

1 Introduction

In modern day computing everything is connected, easily accessible and it is relatively easy to orchestrate infrastructure. This accessibility makes everything easier to manage, and makes work and learning more efficient and often cheaper. With everything being so easily accessible gives the opportunity for misuse. Having poorly segregated networks and a multitude of services, increases the number of attack vectors to your systems and data. Attacks get more and more complex, and there is more focus on elaborate phishing attempts [1]. All of this increases the demand of cyber security and cyber security education, therefore it is crucial to have a controlled environment where students can explore, learn, and become experts in this ever demanding field.

1.1 Background

The Norwegian Cyber Range was created to educate in realistic and diverse cyber security scenarios. The best way to give truly realistic scenarios is to use applications and vulnerabilities meant for real systems, and deploy them in a controlled environment. This problem is what PEMA (*Pentesting Exercise Management Application*) and PLED (*Pentesting Lab Environment Database*) is intended to solve. PEMA is a platform meant to give instructors and students an interface to deploy, log and submit cyber security exercises for use in ethical hacking, penetration testing and cyber security competitions. PLED will provide a storage system where vulnerable applications, CTF-challenges, malware and possibly other files with its corresponding properties can be retrieved or uploaded for ease of use. PEMA can use this storage system to quickly create a wide range of different cyber security tasks.

1.2 Assignment Definition

The assignment is to create a service that... :

- Stores vulnerable applications and other cyber security related files, with their respective metadata.
- Makes it possible to retrieve, search, delete and add such files, along with their metadata, through an API.
- Offers a database web-interface for an overview and user-interaction with the database.

The vulnerable applications should be retrieved from differently available repositories. These applications aure usually linked with a CVE-ID, an ID that describes the vulnerability. To help achieve this, a database web-interface should be available for an administrator to upload files, modify metadata or otherwise interact with the database.

The database web-interface will be integrated in PEMA at a later stage, and PEMA will primarily use the API created by PLED to do operations on the database. In PLEDs

case, the database web-interface will be created for ease of use, testing purposes and for a better overview on what is stored in the database.

1.3 Scope

1.3.1 Assignment delineation

PLED is a sub project of the larger PEMA project and will not have direct a connection with the development phase of PEMA, besides coordination and possible functional requests for the API.

1.3.2 Constraints

The final result of this assignment is to be used for educational purposes only and is meant to be a tool to be used in the course Ethical Hacking. It is meant to educate students in fields of ethical hacking/white hat hacking, pen-testing and academic writing. For use by the Norwegian Cyber Range primarily. PLEDs service will be provided by an API interface and a database web-interface, and it will serve as a first fully functional prototype release of a series of future improvements when the project is handed over to the NCR.

1.4 Business Context

The NCR(Norwegian Cyber Range) is a newly established arena for testing, training and practising cyber security. In this arena both users and systems are exposed to realistic events in safe scenarios. The arena ensures efficient competence building based on real life observations and is created as cyber security is a crucial prerequisite for the digitization of the society.

1.5 Project organizing

PLED consists of three bachelor students, two from IT-Operations and Information Security and one from Programming - Applications. Several of the subjects are common between the two courses, meaning we have both share knowledge in the same areas and can supplement each other where knowledge lacks. Danny Lopez Murillo from NTNU Norwegian Cyber Range is the project owner, while Erik Hjelmås is our coordinator.

- Adrian Jacobsen Moen (BITSEC) - Developer, communication responsible, secretary and Security Champion.
- Askil Amundøy Olsen (BITSEC) - Developer and Security Champion.
- Karoline Moe Arnesen (BPROG) - Main API/backend developer, SQA engineer and database engineer

1.6 Project goals

The project goals of PLED are the following:

1.6.1 Project Effects

The effects of the finished product:

- Instructor workload decreased on lab setup and usage
- Students gets a better experience when working with the lab setup

- Realism and diversity in PEMA lab environments
- Better user experience for course instructors
- Ease of implementing vulnerable applications into other systems

1.6.2 Project Results

The results of PLED are the following:

- Provide a service capable of storing vulnerable applications and various other cyber security relevant files.
- Provide a graphical interface for database management and API administration.
- Provide an API interface towards the database.
- Provide the vulnerable applications and other files using the API interface.

1.7 High Level Use Cases

Figure 1: High level use case overview of PLED and PEMA system.

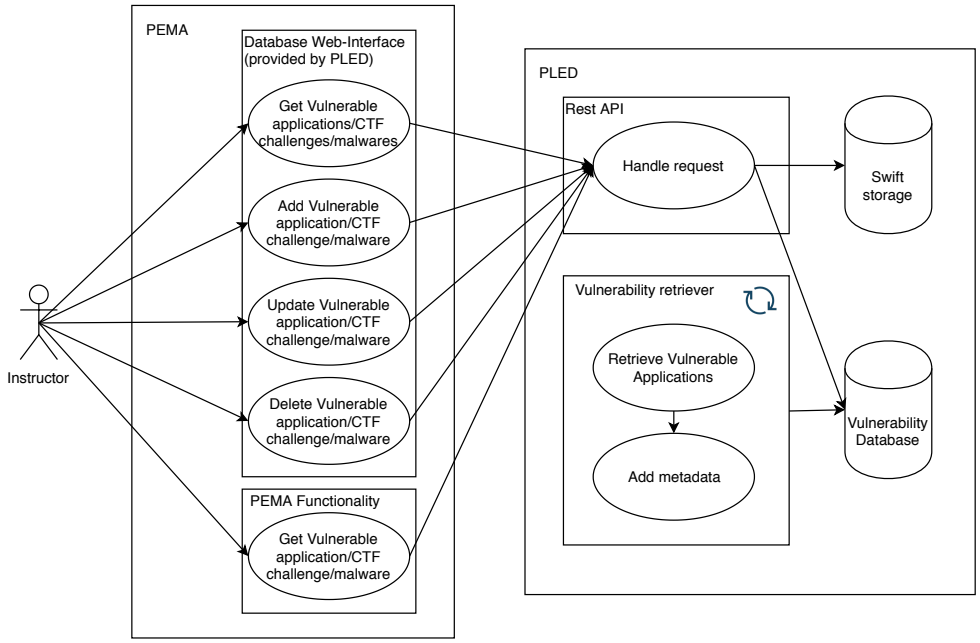


Figure [1] shows a high level use case of PEMA and PLED, this illustrates the different interactions PEMA can do with PLED and the internal mechanisms of PLED.

1.8 Development framework/process

1.8.1 Project modules

The project is divided into the following modules to ease the development phase of PLED:

- Design database schema
- Deploy database(s)
- Defining the REST API

- Implement REST API
- Create graphical interface
- Download and insert vulnerable applications into storage
- Deployment of PLED service
- Testing and Review

1.8.2 Software development framework/process

PLED have chosen the iterative and incremental method in agile development, as PLED needs to be able to adapt to requirement changes down the line, and some requirements are not yet established. Another reason for choosing incremental was that the assignment was not defined at the starting date and was subject to change in the upcoming weeks. Iterative delivery was chosen in order for our coordinator and project owner to efficiently assess the work that has been done since previous meetings. This allows for early feedback on components, and evaluation of how well the design choices worked with what the project owner wished for.

Along with the chosen process, PLED will also use a Kanban board to distribute workload among the group and use it to keep track of pending tasks and prioritize accordingly. Kanban also provides a visualization of the progress from start to finish. Kanban is often used with Scrum, but in PLEDs case, scrum seems inefficient as a development process. The level of organizing, planning and collaboration involved in Scrum is not needed when all group members work from the same location.

The members of PLED have different technological backgrounds so when developing the approach needs to be agile to allow several development processes to work in parallel, this makes development more efficient because one member can work on something only he/she has knowledge on, while another member works in parallel on something else. In a waterfall approach this would be inefficient because one component would have to be done before starting another, and when only one group member works on that component a lot of time is wasted.

1.8.3 Methods and approach

Development and documentation will be carried out continuously, where everything that is developed will be roughly documented and later improved on completion. PLED will be implemented incrementally, using this, a component can be developed along side another component, but when implementing into PLED one component needs to be complete for the other to fully work. An example of this can be that for the database web interface to be fully functional, the REST API and the database must be up and working.

1.8.4 Tools and technologies

Programming Languages

PLED has decided to use familiar tools and technologies in order to have a greater focus on the task at hand. Programming will be done in PHP primarily. PHP is widely used in NTNU's courses, thus known by the students in PLED in addition to fulfill the criteria and needs for this project, without having any considerable downsides.
Additionally other languages will be used for scripting to make the work more efficient for the project and the completed system, these languages are Bash and Python. OpenStack Heat will be used for deployment of the PLED service.

Version Control

Version control will be handled by Git [2], in NTNU NRC's adaptation of Phabricator [3] on `https://project.ncr.ntnu.no/`. Using NCR's Phabricator allows NCR to keep tabs on progress and have a better overview on the status of PEMA as a whole. Phabricator has other features like a wiki which comes in handy for issue tracking, logging and documentation.

Server provisioning

PLED will use NTNUs SkyHigh service for deploying servers for testing and development. SkyHigh gives a management interface for provisioning servers and networks and a CLI interface for quick deployment and deletion of stacks with infrastructure.

Docker

Most of the systems running on the servers are started using Docker, this makes environment and dependency handling easier as the service is shipped with all necessary requirements included.

1.9 Organization of this paper

The report is based off NTNUs own "Bachelor thesis template (NTNU)" found on GitHub.

Summary of the chapters in the report:

1. Introduction - An introduction to the project, describing the assignment and its scope.
2. Requirements - An overview on what requirements are set by the employer and the group, to meet the expectations.
3. Technical Design - Describes the system architecture, the chosen design on the services PLED will provide, and PLEDs features.
4. Implementation - Explains in greater detail how the group built core components by showing code snippets and explanation on how it was implemented.
5. Deployment - Showcasing and describing how the PLED service is deployed services to the infrastructure with OpenStack Heat.
6. Security - Explains security measures implemented in PLED.
7. Operations - Describes operational elements added to the infrastructure.
8. Testing - Lists what kind of tests and methods of testing was done to ensure a working system.
9. Discussion - Discussions on what the group achieved, how the end product was achieved.
10. Conclusion - Concluding the work that has been done and discussing future work.
11. Bibliography - Lists sources of information and citations used in the report.
12. Glossary - Explanation of perhaps unfamiliar words and technologies.

2 Software requirements specification

2.1 Introduction

This section lists the requirements articulating the needed capabilities, functions, innovations and constraints for the PLED software development project. These requirements establish the basis agreement between the PLED-team and NCR on how the PLED projects finished product should function. It includes requirements for PLEDs REST API, database, database web-interface and other additional functionality included in the project. The group has ranked these specifications using the following critically levels;

- Mandatory - Cannot be sacrificed.
- Desirable - Important, but could be sacrificed if necessary to uphold the schedule.
- Optional - May not be developed or implemented, but "nice to have".

The requirements section is divided by the project modules where business, non-functional, functional and technical requirements are listed. The collection of requirements for each module defines the complete function, features and characteristics for that specific module to satisfy stated needs in the project assignment. This means the specification of behavior between outputs and inputs and their quality features. To be of help for quality assurance (non-functional requirements), the *Quality Assurance Model* described in the *ISO/IEC 25010:2011 standard*[4] is included.

ISO25010 was chosen to aid in the process of determining which quality characteristics to use when evaluating the properties of the requirements. These are used in turn to evaluate if the system that is developed satisfies the needs.

2.2 PLED Vulnerability Database

2.2.1 Introduction

One of the first issues for the group to figure out is how to store its vulnerable applications/files and associated metadata onto the system. Since the system is intended to consist of a larger number of applications and files, the group do not want to store all these files directly in the database. Instead the group needs to create a database for all metadata and link to the different formats for storing the vulnerable applications/files. To achieve this, the database needs to be able to retrieve, add, update and delete documents synchronous and relational with the file storage to achieve consistency in the data.

2.2.2 PLED Vulnerability Database - Business Requirements

Table 1: **BR-1** - Vulnerability metadata storage

Requirement	Storage for metadata associated to vulnerable applications.
Description	The NCR is in need of a storage for metadata associated to vulnerable applications to be of use in the learning platform, PEMA.
Criticality	Mandatory.

2.2.3 PLED Vulnerability Database - Non-functional Requirements

Table 2: **NFR-1** - Database traffic encryption

Requirement	Database traffic encryption.
Description	Encrypt communication between database and host to ensure confidentiality and integrity.
Criticality	Mandatory.
Functional Requirement(s)	FR-1
Quality Assurance	Integrity, confidentiality.

Table 3: **NFR-2**- Recoverable database

Requirement	Recoverable database.
Description	If the database is corrupt, reverting to the previous version is possible.
Criticality	Mandatory.
Functional Requirement(s)	FR-2
Quality Assurance	Reliability.

Table 4: **NFR-3** - External database configuration

Requirement	External database configuration.
Description	Configuration files are stored on a separate location from the running database.
Criticality	Desirable.
Functional Requirement(s)	FR-3
Quality Assurance	Portability, Maintainability.

Table 5: **NFR-4** - Database data is modifiable

Requirement	Database data is modifiable.
Description	Data stored should be modified if needed.
Criticality	Desirable.
Functional Requirement(s)	FR-5, FR-6, FR-7
Quality Assurance	Maintainability

Table 6: **NFR-5** - Database remote access

Requirement	Database remote access.
Description	Database is available beyond local network
Criticality	Mandatory.
Functional Requirement(s)	FR-4
Quality Assurance	Availability

Table 7: **NFR-6** - Database authentication

Requirement	Database authentication.
Description	Authenticate use of database
Criticality	Mandatory.
Functional Requirement(s)	FR-9
Quality Assurance	Authenticity

2.2.4 PLED Vulnerability Database - Functional Requirements

Table 8: **FR-1** - Encrypted database traffic with TLS/SSL

Requirement	Encrypted database traffic with TLS/SSL.
Description	All traffic is encrypted using database configuration for TLS/SSL.
Criticality	Mandatory.
Dependencies with other requirements	NFR-1

Table 9: **FR-2** - Replicated database service

Requirement	Replicated database service.
Description	Database uses replica sets for recoverability and redundancy.
Criticality	Mandatory.
Dependencies with other requirements	NFR-2

Table 10: **FR-3** - External database Configuration

Requirement	External database Configuration.
Description	Using volumes to store the configuration separately.
Criticality	Desirable.
Dependencies with other requirements	NFR-3

Table 11: **FR-4** - Remote database connection

Requirement	Remote database connection.
Description	Establish a remote connection to the database using an externally available IP.
Criticality	Mandatory.
Dependencies with other requirements	NFR-5

Table 12: **FR-5** - Add content to the database

Requirement	Add content to the database.
Description	Content can be added to the database using both database web-interface or REST API interface.
Criticality	Mandatory.
Dependencies with other requirements	NFR-4, NFR-5

Table 13: **FR-6** - Update content(s) of the database

Requirement	Update content(s) of the database.
Description	Any added content can be updated.
Criticality	Desirable.
Dependencies with other requirements	NFR-4, NFR-5

Table 14: **FR-7** - Delete content from the database

Requirement	Delete content from the database.
Description	Any added content can also be deleted.
Criticality	Mandatory.
Dependencies with other requirements	NFR-4, NFR-5

Table 15: **FR-8** - Database auditing

Requirement	Database auditing.
Description	All query actions are logged for auditing purposes.
Criticality	Optional.
Dependencies with other requirements	None.

Table 16: **FR-9** - Authenticated database sessions

Requirement	Authenticated database sessions.
Description	All sessions towards the database needs to be authenticated before allowed access.
Criticality	Mandatory.
Dependencies with other requirements	NFR-6

2.3 File Storage

2.3.1 Introduction

When metadata is stored in the database, PLED needs a location to store files. These files can be the actual applications, configuration files, CTF-challenges and more. The storage solution will have the actual file stream, while the database store a reference to that file stream. The stored files need to have unique identifiers and the storage solution must be scalable.

2.3.2 File storage - Business Requirements

Table 17: **BR-2** - Storing files corresponding to database metadata

Requirement	Storing files corresponding to database metadata
Description	PLED needs a storage solution for the uploaded or retrieved files
Criticality	Mandatory

2.3.3 PLED File storage - Non-functional Requirements

Table 18: **NFR-7** - Interactability with file storage service

Requirement	Interactability with file storage service.
Description	The API should be able to interact with the file storage.
Criticality	Mandatory
Dependencies with other requirements	FR-10, FR-11, FR-12
Quality Assurance	Usability

Table 19: **NFR-8** - Adaptable storage

Requirement	Adaptable storage
Description	The storage solution needs to adapt to what kind of data is inserted.
Criticality	Desirable
Dependencies with other requirements	FR-13
Quality Assurance	Adaptability

Table 20: **NFR-9** - Scalable storage

Requirement	Scalable storage
Description	The storage solution needs to be able to scale according to the current demand
Criticality	Desirable
Dependencies with other requirements	FR-13
Quality Assurance	Capacity

2.3.4 PLED File Storage - Functional Requirements

Table 21: **FR-10** Insert file

Requirement	Insert file.
Description	Insert a file into storage with a unique identifier.
Criticality	Mandatory.
Dependencies with other requirements	NFR-7

Table 22: **FR-11** Retrieve file

Requirement	Retrieve file.
Description	Retrieve a file stored.
Criticality	Mandatory.
Dependencies with other requirements	NFR-7

Table 23: **FR-12** Delete file

Requirement	Delete file
Description	Delete a file stored when the data is deleted from the database
Criticality	Mandatory
Dependencies with other requirements	NFR-7

Table 24: **FR-13** Object Storage

Requirement	Object Storage
Description	PLED should use object storage to provide adaptable and scalable storage based on key:value pairs
Criticality	Desirable
Dependencies with other requirements	NFR-8, NFR-9

2.4 PLED REST API

2.4.1 Introduction

The PLED REST API is created to handle all communication with the vulnerability database. This way any project or individual can be granted access to the storage by simple be given a user with customized access to the REST API. For the scope of this thesis, PLEDs main goal is to satisfy user/instructor needs from the PEMA project on the storage. This includes insertion, modification and deletion, in addition to retrieval with customized filtering. The REST API need to retrieve all metadata in addition to the vulnerable applications/files or other media attached to the vulnerability, in a readable JSON format as agreed and coordinated with the PEMA-team.

2.4.2 PLED REST API - Business Requirements

Table 25: **BR-3** - Storage communication

Requirement	Storage communication
Description	In addition to having a way to store vulnerable applications/files, the NCR is in need of a way to communicate with this data storage to be able to make use of it in PEMA.
Criticality	Mandatory

2.4.3 PLED REST API - Non-functional Requirements

Table 26: **NFR-10** - Query request authentication

Requirement	Query request authentication
Description	Since the API is deployed in an semi-open environment, it needs a way to limit the application access to permitted users.
Criticality	Desirable
Dependencies with other requirements	FR-14
Quality Assurance	Security

Table 27: **NFR-11** - API traffic encryption

Requirement	API traffic encryption
Description	Inbound and outbound API traffic should be encrypted.
Criticality	Desirable
Dependencies with other requirements	FR-15
Quality Assurance	Security

Table 28: **NFR-12** - API documentation

Requirement	API documentation
Description	Thorough documentation allowing easy maintainability and usage.
Criticality	Mandatory
Dependencies with other requirements	FR-16
Quality Assurance	Usability

Table 29: **NFR-13** - Ensuring correct query results

Requirement	Ensuring correct query results
Description	The results must correspond with the query input, and satisfy the instruction set in the documentation.
Criticality	Desirable
Dependencies with other requirements	FR-17
Quality Assurance	Correctness

Table 30: **NFR-14** - Remotely available API

Requirement	Remotely available API.
Description	The API should be available from other servers and services.
Criticality	Mandatory.
Dependencies with other requirements	FR-18
Quality Assurance	Availability.

2.4.4 PLED REST API - Functional Requirements

Table 31: **FR-14** - Query request authentication

Requirement	Query request authentication.
Description	DreamFactory applies authentication and role based permissions to the API, with API-keys.
Criticality	Desirable.
Dependencies with other requirements	NFR-10

Table 32: **FR-15** - API traffic encryption

Requirement	API traffic encryption.
Description	The API endpoint is configured to run over HTTPS using TLS/SSL for encryption.
Criticality	Desirable.
Dependencies with other requirements	NFR-11

Table 33: **FR-16** - API documentation

Requirement	API documentation.
Description	The API is thoroughly documented live using DreamFactorys implementation of Swagger.
Criticality	Mandatory.
Dependencies with other requirements	NFR-13

Table 34: **FR-17** - Ensuring correct query results

Requirement	Ensuring correct query results.
Description	Using functional- and unit-testing to ensure correct query results.
Criticality	Desirable.
Dependencies with other requirements	NFR-13

Table 35: **FR-18** API remote access

Requirement	API remote access
Description	Establish a remote connection to the API using an externally available IP.
Criticality	Optional
Dependencies with other requirements	NFR-14

Table 36: **FR-19** Query vulnerability metadata documents

Requirement	Query vulnerability metadata documents
Description	Return all available vulnerability metadata documents currently stored in the PLED database
Criticality	Optional
Inputs	_table/vuln_applications
Outputs	JSON document with current available vulnerability metadata stored in the database.
Dependencies with other requirements	NFR-14

Table 37: **FR-20** Query vulnerability metadata document

Requirement	Query vulnerability metadata document
Description	Returns vulnerability metadata that matches the input data.
Criticality	Mandatory
Inputs	Any key(s) existing in the collections. Ex. Vulnerable application ExploitDB ID: _table/vuln_applications?filter= applications.exploitdb_id='317'
Outputs	JSON document where vulnerability metadata matched.
Dependencies with other requirements	NFR-14

Table 38: **FR-21** POST vulnerability metadata

Requirement	POST vulnerability metadata
Description	Add vulnerability metadata into the Vulnerability Database.
Criticality	Mandatory
Inputs	JSON Body of document to be inserted
Outputs	None.
Dependencies with other requirements	NFR-14

Table 39: **FR-22** PUT/PATCH vulnerability metadata

Requirement	PUT/PATCH vulnerability metadata
Description	Updating vulnerability metadata in the Vulnerability Database.
Criticality	Mandatory
Inputs	JSON Body of document to be updated
Outputs	None,
Dependencies with other requirements	NFR-14

Table 40: **FR-23** DELETE vulnerability metadata

Requirement	DELETE vulnerability metadata
Description	Deleting vulnerability metadata in the vulnerability database
Criticality	Mandatory
Inputs	Filter of what document to be deleted.
Outputs	None.
Dependencies with other requirements	NFR-14

2.5 PLED Database Web-Interface

The database web-interface is created to simplify how an instructor can modify data in the database. In production the interface will be implemented and managed by the PEMA project.
For the project and testing purposes, a version of the interface will be implemented by the group, as it is closely related to modification of the database and knowledge of the database design is mandatory. The interface will still be using the REST APIs authenticated PEMA user and query requests to communicate with the database, but represents a standardized method for the different modification methods that should be available from PEMA.

2.5.1 Database web-interface - Business Requirements

Table 41: **BR-4** - Ease of Vulnerability Database modification

Requirement	Ease of Vulnerability database modification
Description	NCR need a simple way to manually modify the database in PEMA.
Criticality	Mandatory

2.5.2 Database web-interface - Non-functional Requirements

Table 42: **NFR-15** - Standardized views for database modification

Requirement	Standardized views for database modification
Description	The Database web-interface should provide different standardized methods for modification on data in the Vulnerability Database. This includes views for insertion, modification and deletion.
Criticality	Mandatory
Dependencies with other requirements	FR-24, FR-25, FR-26, FR-27
Quality Assurance	Usability

Table 43: **NFR-16** - Access Control

Requirement	Access Control
Description	When a user tires to access the database interface, a username and password must be entered to authenticate the user before accessing the interface
Criticality	Desireable
Dependencies with other requirements	FR-28
Quality Assurance	Security

2.5.3 Database web-interface - Functional Requirements

Table 44: **FR-24** - Add content to Vulnerability Database

Requirement	Add content to Vulnerability Database
Description	Instructor can add content to the database using forms in the database web-interface
Criticality	Mandatory
Dependencies with other requirements	NFR-15

Table 45: **FR-25** - Update content in the Vulnerability Database

Requirement	Update content in the Vulnerability Database
Description	Instructor can update content from the database using the database web-interface.
Criticality	Desirable
Dependencies with other requirements	NFR-15

Table 46: **FR-26** - Delete content in the Vulnerability Database

Requirement	Delete content in the Vulnerability Database
Description	Instructor can delete content from the database using the database web-interface
Criticality	Mandatory
Dependencies with other requirements	NFR-15

Table 47: **FR-27** - View content of the Vulnerability Database

Requirement	View content of the Vulnerability Database
Description	Instructor can view content from the database using the database web-interface
Criticality	Desirable
Dependencies with other requirements	NFR-15

Table 48: **FR-28** - Authenticate User

Requirement	Authenticate User
Description : **Criticality**	Use a API query to authenticate a user, before allowing access to the interface Desirable
Dependencies with other requirements	FR-14, NFR-16

2.6 PLED Administrative Interface

2.6.1 Introduction

The ensure authentication when using the API there must be a way of creating roles and assigning roles to different functions of the API. PLED and NCR will need a management interface where these roles can be created and administered.

2.6.2 PLED Administrative Interface - Business Requirements

Table 49: **BR-5** - Ease of administration

Requirement	Ease of administration
Description	The NCR is in need of a simple way to be able to administrate the API, its users and user permissions.
Criticality	Mandatory

2.6.3 PLED Administrative Interface - Non-functional Requirements

Table 50: **NFR-17** - Encrypted communication with the API administrative interface

Requirement	Encrypted administrative interface traffic.
Description	Communication with the administrative interface should be encrypted with TLS/SSL.
Criticality	Desirable
Dependencies with other requirements	None
Quality Assurance	Security.

Table 51: **NFR-18** - Authentication for the administrative interface

Requirement	Authentication for the administrative interface.
Description	Users must be authenticated before getting access to the administrative interface.
Criticality	Mandatory
Dependencies with other requirements	None
Quality Assurance	Security.

2.6.4 PLED Administrative Interface - Functional Requirements

Table 52: **FR-29** - Generate API keys

Requirement	Generate API keys
Description	Before an API can communicate with a service, the instructor must generate an API key for said service.
Criticality	Mandatory
Dependencies with other requirements	NFR-18
Quality Assurance	Security.

Table 53: **FR-30** - Create user

Requirement	Create user
Description	The interface should provide creation of users to enable authentication and to assign roles
Criticality	Mandatory
Dependencies with other requirements	NFR-18
Quality Assurance	Security, usability

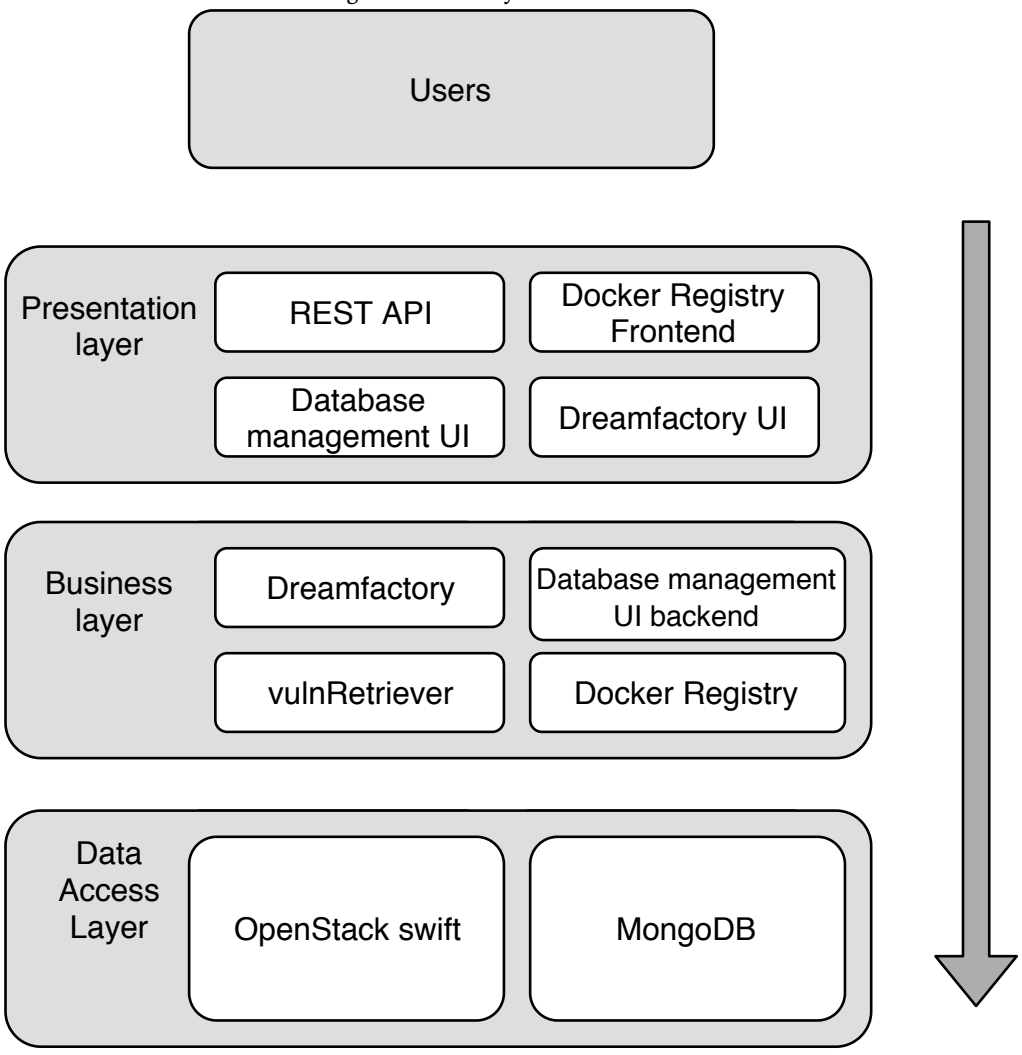
Table 54: **FR-31** - User management

Requirement	User management
Description	Users can be assigned to roles with specific permissions on what they are allowed and not allowed to do.
Criticality	Mandatory
Dependencies with other requirements	NFR-18
Quality Assurance	Security.

3 Technical Design

3.1 System Architecture

Figure 2: PLED System architecture



3.1.1 Introduction

Figure [2] shows PLED services architectural design, this design is based on the requirement to be a modular and loosely coupled system. where each component can be changed and updated with minimal effect on the other components. Having a flexible solution will help when joining PLED with PEMA or even other NCR projects. This is also done for future work on the project, for example if it is decided that object storage

did not fit the requirements as expected, that component can be changed to something different only the file location in the database would have to be updated.

Each section in this chapter will present the architecture of the segmented components and how it is designed to satisfy the functional and non-functional requirements.

3.1.2 Architectural design

Because the group wants PLED to be modular and decoupled, a layered architecture will be used. This architecture is based on horizontal layers that each have a separate role in the system. Each layer is designed to be independent from the other layers, this way one layer can be changed out with minimal effect on the other layers. The architecture used consists of three layers:

Presentation Layer

This layer is the one users interact with. It is used to present data gathered from the business layer. In PLED this is primarily the REST API and database web-interface, but DreamFactory and the Docker Registry frontend is also included.

Business Layer

In this layer, all the business logic is carried out. Business logic is everything that is concerning the processing of data such as calculating or formatting. In PLED, this layer consists of the vulnerability retriever*(retrieve vulnerable apps from ExploitDB)*, database web-interface back end*(PHP logic)*, Docker Registry *(Docker image processing)* and also DreamFactory*(processing data from database to the API endpoint)*.

Data Access Layer

This layer is all about the data storage itself. The business layer will insert or retrieve data directly from this layer, to be processed. In PLED, this layer consists of MongoDB and OpenStack Swift

3.2 Database Design

3.2.1 Preparation

To better understand the reasoning and background for decisions in the database design phase, several aspects has been considered and assumptions has been made.

BASE

Starting with BASE, meaning Basic Availability, Soft-state and Eventual consistency.

Basic availability: Automatic population will happen once per week, if enabled. Manual population will happen at random times. Meaning the database should be mostly available, but it is not paramount that availability is ensured at all times[3.2.1].

Soft-state: State is not necessarily consistent, and the current state is not critical for retrieving/using the provided service.

Eventual consistency: The database will not be constantly synchronized with ExploitDB, it will be updated once per week, again, only if enabled. What is more important is that the database is filled with a multitude of files and applications ready for retrieval through the REST API. In the replica set, the replicated databases will eventually become consistent with the master node. During this, the time gap whereas the database is inconsistent is extremely small, as updating the database consists of few and small operations.

Usage pattern

Based on what is known at the time of writing the report, assumptions are made from meetings and conversations with NCR, the supervisor and the employer. Meeting logs can be found in appendix [A]. Roughly defined usage patterns have been made up to create a scope for the database. During population of the database via the vulnerability retriever, writes will happen consistently over a long period of time, approximately 12 hours, as the scraper needs a sleep interval in order not to be blocked by the site holding the content [3.5.1].

The group assumes few, 1 - 10, users will interact with the database, at random times, moderately few times per week (0-1000 requests). The reasoning behind this is that after PEMA has deployed its lab environment, PLED will generally not see more usages until the next lab environment needs to be deployed by PEMA.

Interactions from PEMA will more often than not consist of GET-requests, which means searching documents for key-words, for example supported platforms and CVE-ID. PEMA will to a lesser degree generate PUT requests to fill in custom content. Once per week, a check will run to update the database in line with what's available on ExploitDB.

Database content

The content of the database is divided into collections, where each collection holds documents containing *metadata* for the individual files in storage. The collection provided are

- *vuln_applications*: Vulnerable applications, mostly automatically retrieved from scraping ExploitDB, other applications can be added manually
- *ctf_challenges*: All CTF-based challenges, these are all manually added by PEMA or through the database web-interface.
- *malicious_files*: Malware and malicious files used for malware analysis and malware lab projects

E.g. MongoDB offers a flexible data model, meaning that the document oriented database is schema-less. Different documents can have very inconsistent schemas and data content, unlike a relational schema where each tuple in a relation will have the same attributes.

Other considerations

- Instructors should be able to add their own files that can be applied to a CTF-challenge or lab environment, which not necessarily has the same metadata as other documents present in the database. *Files* in this regard means it's a lot of varying file-types with different attributes, size and usages.
- Not knowing what kind of files and metadata will populate the database in future development, is a major driver for going for a non-relational database.

- The data is not complex, large tree structures is not applicable in this scenario.

To ensure functionality with future changes in requirements, the group needs to take this into consideration when choosing what type of database is best for the assignment. Thus the group has decided to use a Document Oriented Database, a subset of a NoSQL database. More specifically, the group went for MongoDB as their database-software, utilizing the unstructured data support, high throughput, storage of potentially large binary files and similar syntax compared to MySQL.

3.2.2 Design schema

Using the previously discussed topics will help the group choose between three different database schemas.

Denomalization

Every file or application document is individual with no relation to any other document, for example all applications have their own CVE-field.

Table 55: Denormalization

Pros	**Cons**
When an application document is deleted, so is the CVE	Duplicated data
Simple schema	If a CVE changes, already existing CVE-IDs in the database wont be consistent
Faster reads, as there are no joins	Larger database
Easy to implement	
Fits PLEDs usage pattern	

Reference document

All file or application documents reference a document with the respective CVE describing the vulnerability.

Table 56: Reference document

Pros	**Cons**
Documents remain small	Need an application level to join documents
	Difficult to view the referenced CVE document in DreamFactory
	If a CVE-ID is deleted, application documents references a non existent document

Embedded document

Each file or application documents are embedded within a CVE document.

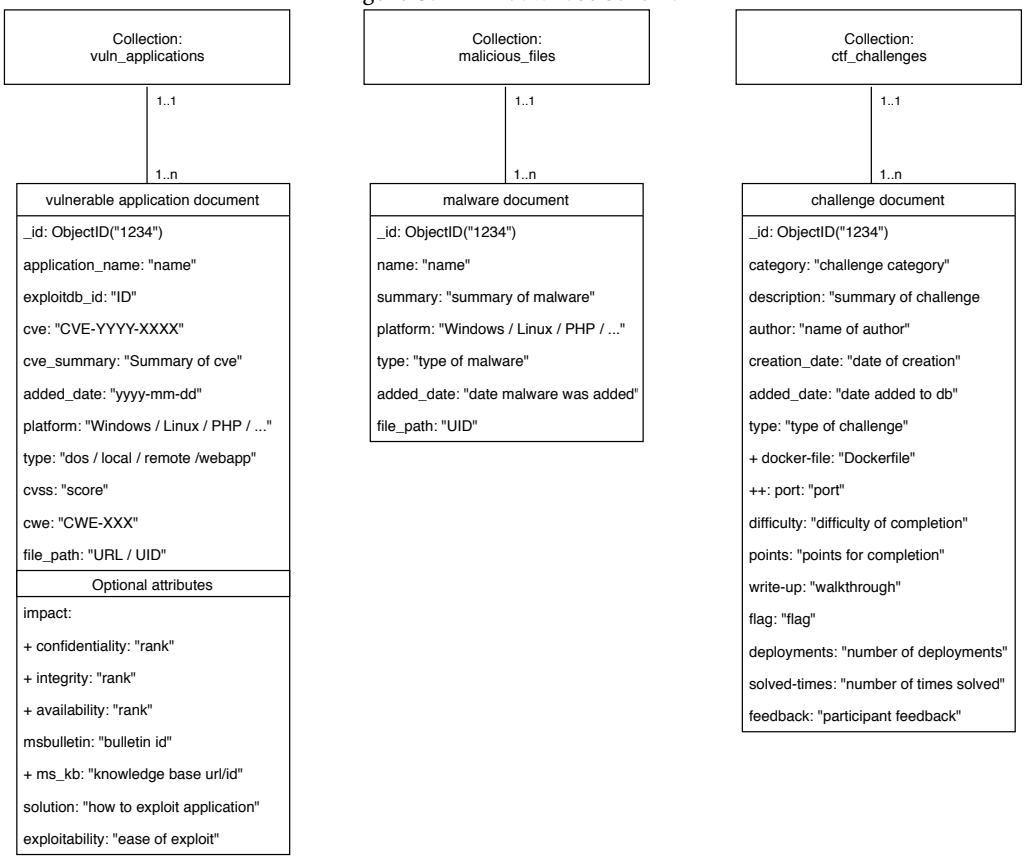
Table 57: Embedded document

Pros	**Cons**
Reference fields directly with dotted notation	Unbound growth of embedded documents
Retrieve subset of fields	

Because of the estimated usage pattern and the way the database will be queried, using embedded documents will create extra overhead when PEMA is searching for an application. This is because the parent document, which in this case is the CVE document, will always be returned with all its data when searching for an application. So all the applications no matter the filter selected will be returned.

All things considered, the best way for us to structure the database is to denormalize it where each document will have all the necessary data. This way will make the insertion of documents and filtering of documents more efficient. The database will still remain organized in the sense that the documents are divided into respective collections. Figure [3] shows a detailed overview of the database structure.

Figure 3: PLED database schema



3.2.3 Secure Communication

All internal traffic to and from the database is to be encrypted using TLS/SSL, this is to ensure authentication, confidentiality and integrity. TLS/SSL will be implemented in MongoDB using a self-signed certificate, this is for testing and development purposes only. A self-signed certificate encrypts communication, but has no way of validating the server identity. Meaning, its susceptible to man-in-the-middle attacks, but traffic cannot be eavesdropped.

For a production environment, it is therefore recommended to use an internally generated and signed certificate authority for proper implementation of TLS/SSL.

3.2.4 Backup and redundancy

For redundancy the database will run in a replication cluster so that the server it is running on does not become a single point of failure. All the databases will also have backups, both full database dumps and incremental backups for changes in the database.

3.3 File storage

There were several methods of storing files that appealed to PLEDs use cases, too many to evaluate. But the main contenders are usage of GridFS, a MongoDB solution of storing binary files above the max document size [5] in the database through sharding; a file server with corresponding documents in the database in which a file path field is used to retrieve the actual file; or the same of the previous suggestion, but storing the files in OpenStack Object Storage through the Swift API [6].

File Server

Table 58: File server Pros/Cons

Pros	Cons
• Storage of files is simple, as files are not sharded into smaller files. • Much cheaper in terms of processing and storage [7]. • Simple setup.	• A change in file structure would be problematic, as all metadata would potentially need to be updated.

Object Storage (Swift)

Table 59: Object Storage (Swift) Pros/Cons

Pros	Cons
• Infrastructure for Swift and Object Storage is already built and ready for use. • API support. • Authentication is implemented. • Supports up to 5GB files, and larger files are segmented before upload.	• Can be complex to setup • Dependent on 3rd party software (boto3 for Python, S3/Openstack client for PHP)

GridFS

Table 60: GridFS Pros/Cons

Pros	Cons
• Easier to protect access to files so only authorized users can see them. • File integrity ensured in a greater degree. • Files stored in the database do not require a different backup strategy. Files stored on a file server do. In other words, easier backup/restore. • Easier portability and scalability of the database.	• More processing is needed to "unshard" the applications in the database, aka heavy load on db server. • Database storage is usually more expensive than file system storage. • Could potentially become a massive database. • GridFS is great for accessing information from portions of large files, like streaming. But this is not in favour of the groups case. • Doesn't really meet PLEDs intended usage.

Based on these arguments the group chose Object Storage/Swift, storing applications and files in OpenStack without metadata (other than an ID). The metadata is stored separately as documents in a MongoDB instance, referencing the application stored in Swift in a field in the document. This way, searches are more optimized as the database is better at handling advanced searches in documents, compared to the API searching through a multitude of large files. Swift utilizes storage containers, containing both folders and the actual files. The folders are named relative to the corresponding collection in the database.

Figure 4: PLED storage diagram

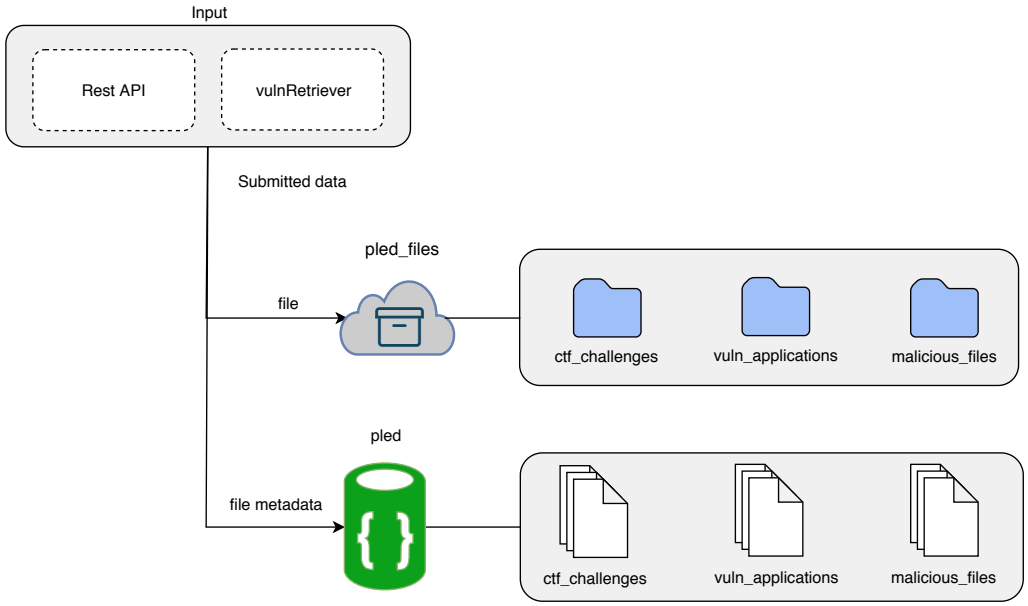


Figure [4] show how the different storage mechanisms are laid out. A file will be stored in the *pled_files* container in Swift, and its metadata are added as a document in one of the three collections.

3.3.1 Storing of Docker images

In CTF-based challenges, those challenges requiring remote access will use Docker Containers. These containers will be created using a Dockerfile that is uploaded to PLED. When uploading a Dockerfile the user specifies if it requires a port and any additional files or metadata.

Docker Registry

Additional to the Dockerfile upload function, it will be deployed a Docker Registry that users are able to upload built images to, this service is not intended to be the main Docker storage, but will work as an additional tool that the NCR can use.
The Docker Registry will have an interface that users can browse images in, this will also require basic authentication that is implemented when setting up the registry.

3.4 REST API Design

Initially the group intended to create their own REST API using GoLang since a member of the team had knowledge on this from the course Cloud Computing. Creation of a REST API with secure authentication and different users with different permission layers would be a tedious and time consuming activity that would extend over larger parts of the projects time schedule. It would be critical to consider several aspects such as usability, security and efficiency. The group was recommended to look into platforms that automates the REST API creation to both save time and to secure correct implementation, in addition to supply other needed features.

For creation of the REST API, the group decided to use the recommended *DreamFactory Service Platform* [8]. This platform developed by DreamFactory Software company delivers complete automation of a REST API able to communicate with several services, in this case on MongoDB. In addition the DreamFactory Service Platform supplies other necessary features needed on the REST API such as user management and access control. The platform also includes live documentation generated through *Swagger*[9], and plenty other functionality which could be useful for further development in the NCR. By using this platform the group saves a lot of time and effort on design and implementation that can instead be used to focus on the other goals and ideas for the project to further expand the project scope.

Since the DreamFactory Service Platform automatically creates its own design and structure for the REST API, its still necessary to understand how to use the platform and create a use guide for further use also.

3.4.1 Documentation

The aforementioned documentation is live, meaning its automatically up to date depending on the database content. It can be interacted with for testing purposes in DreamFactory's own UI, all of which PEMA has access to. Figure [5] shows an example on how the documentation looks like. Each of the rows can be expanded for further explanation on how to use the commands available.

Figure 5: DreamFactory API Documentation

Additional documentation is provided on PLEDs own wiki hosted on Phabricator, also included in appendix [F]. This is meant to provide users with examples on how to use the available filters and options in the DreamFactory provided REST API.

3.4.2 User management

Users are required to create a layer of access control when using the DreamFactory UI or when querying the API. Users are manually created in DreamFactory and can be assigned roles. These roles determine what API queries a user can use through a token generated, that is added to the request URL. A 'super admin' needs to be created at first so that that user later can create additional users. Each user created can also log into the DreamFactory interface and view the documentation for the API.

Figure 6: API sequence diagram

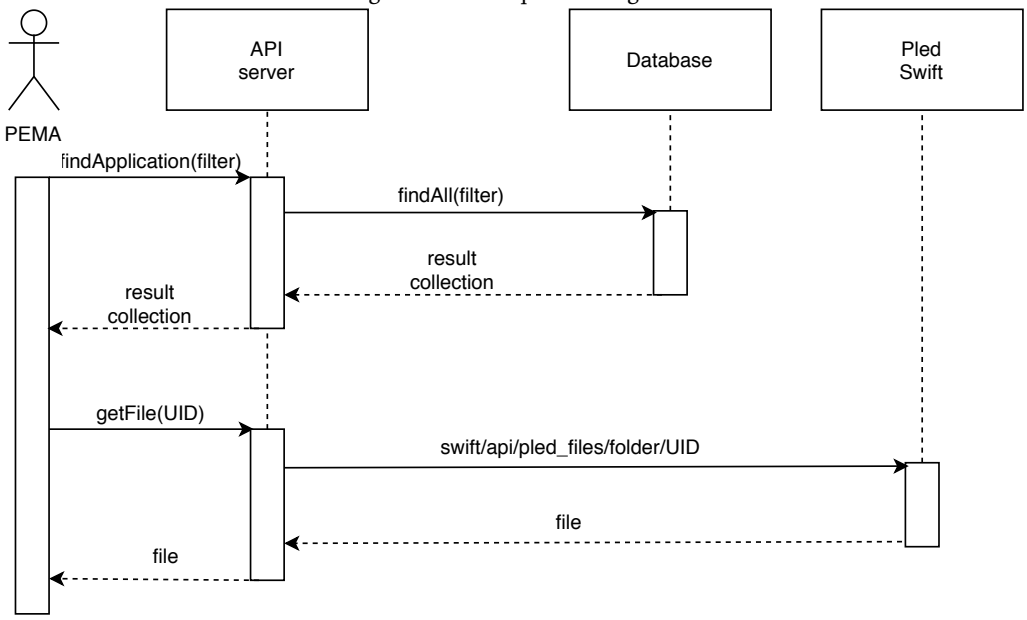


Figure [6] show the how the user authentication is involved when querying the api.

3.4.3 Domain semantics

The instructor or PEMA can request the different applications, challanges and malware stored in the database through the API using single or multiple filters of fields in a query. The vulnerable applications contain metadata such as a CVE ID, a CVSS score, vulnerability description, supported platform, version number of the application, type of vulnerability.

Table 61: API domain semantics

Operation	Method	Description
Retrieve	GET	Retrieve one or more records
Create	POST	Create one or more records
Update	PUT	Update (replace) one or more records
Update	PATCH	Update (patch) one or more records
Delete	DELETE	Delete one or more records

The DreamFactory Service Platform REST API provides the following ways for applications, challenges and malware to retrieve, create, update and delete data from the collections. The collection name is always sent as a part of the URL, whilst the other operations are sent as query parameters or as part of the posted data [10]. One query parameter available on all request is the 'Field' which dictates what fields to be returned for the affected records of the operation. These can either be sent as comma-delimiter string of field-names when passed as a query parameter, or an array of field names when passed in posted data. In GET requests this parameter defaults to returning all fields, while all other request types return only the identifying fields by default. This way the client does not need to do a second query to get thing like updated or auto-filled field

values. The output will always reflect the input, meaning an array posted will result in an array received.

Retrieving records

There are several ways that an application can retrieve data through the DreamFactory REST API. In PLED, records are stored by unique identifiers created by MongoDB, these identifiers can be used to retrieve single or multiple matching records. By a single record identifier, the identifying field is passed as a part of the URL after the collection name. When retrieving records with several identifiers we use the 'ids' query parameter sent as a comma-delimiter string of id values or an array of id values in posted data.

3.5 Database web-interface

For viewing, adding, updating and deleting vulnerable applications, challenges or malware in the database, a database web-interface will be provided as proof of concept. This interface will allow an instructor or administrator to upload metadata documents to the database and corresponding files to Swift, as well as updating or deleting documents. Figure [7] displays the different views that is intended to be displayed in the database web-interface. Since this part of the project will be integrated into PEMA at a later stage, it is a bit uncertain what technologies and methods will be used. PEMA is looking into having other students work on the front-end part of their service, but this would have to be done at a later stage. For further use, it would be an idea that the group could deliver the database web-interface as several components or plugins for different frameworks such as angular, react or WordPress, but for the scope of the thesis it is chosen to create a simple template that could easily be implemented into different frameworks.

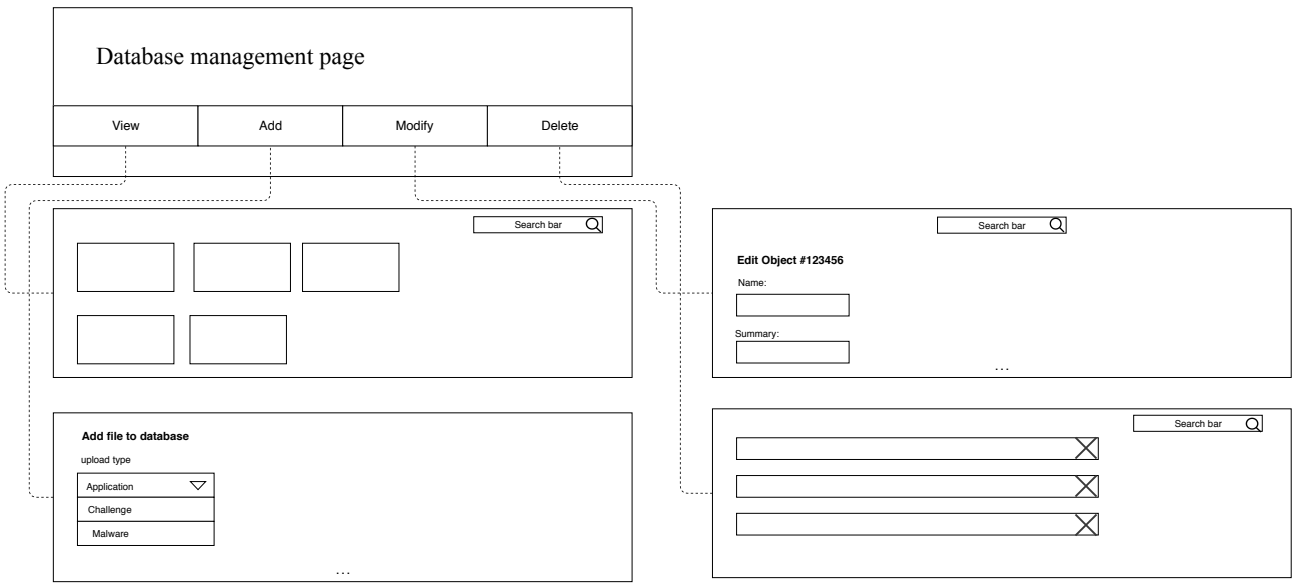


Figure 7: Sketch of database web-interface

3.5.1 Features

View and Search contents of database

The first feature the instructor will be presented with, is the view of the last inserted database contents. This view can than be used by the instructor to view all currently available vulnerable applications, challenges and malware to be used in PEMA. In addition, the instructor is given a field for searching for contents. The contents and search results needs to be presented in a organized and structured matter so that content can be easily found and read.

Forms for adding content

The database web-interface will allow the instructor to submit vulnerable applications, CTF-challenges or malware to the database. This will be done by having different forms available according to the types of file the instructor wants to upload. The data will then be processed by the back end according to type the instructor chose. The different types of file are:

- CTF-challenge
- Vulnerable application
- Malware

The metadata of the file is inserted into the corresponding MongoDB collection (using an API POST query), along with a ID corresponding to the file in OpenStack Swift.

For vulnerable applications, the instructor can include a CVE for the application to be uploaded. When inserted and the instructor submits the form, the back end need to check if the CVE is valid. If found, corresponding metadata for that CVE is appended to the application metadata. PLED will use *CIRCL API* [11] to retrieve this information add it to the MongoDB documents.

Forms for updating metadata

For modification and updating of metadata related to the different file types, the database web-interface will provide a form to update the current data stored. This feature is added to provide the instructor with a way to fix potential mistakes or update outdated summaries etc.

Deletion of content

The database web-interface will also provide a way to delete contents of the database and associated stored file.

3.6 Vulnerable Application Retrieval

As a part of the database, a way of automatically retrieving vulnerable applications and inserting them into the database is preferred. This feature is not required and is not viewed as a crucial component of PLED. Following the requirements of what is needed from a vulnerable application, the source of the application needs to provide certain properties for the application. This is to ease the user when searching for applications and to make deployment of the application more efficient.

Figure 8: vulnRetriever sequence diagram

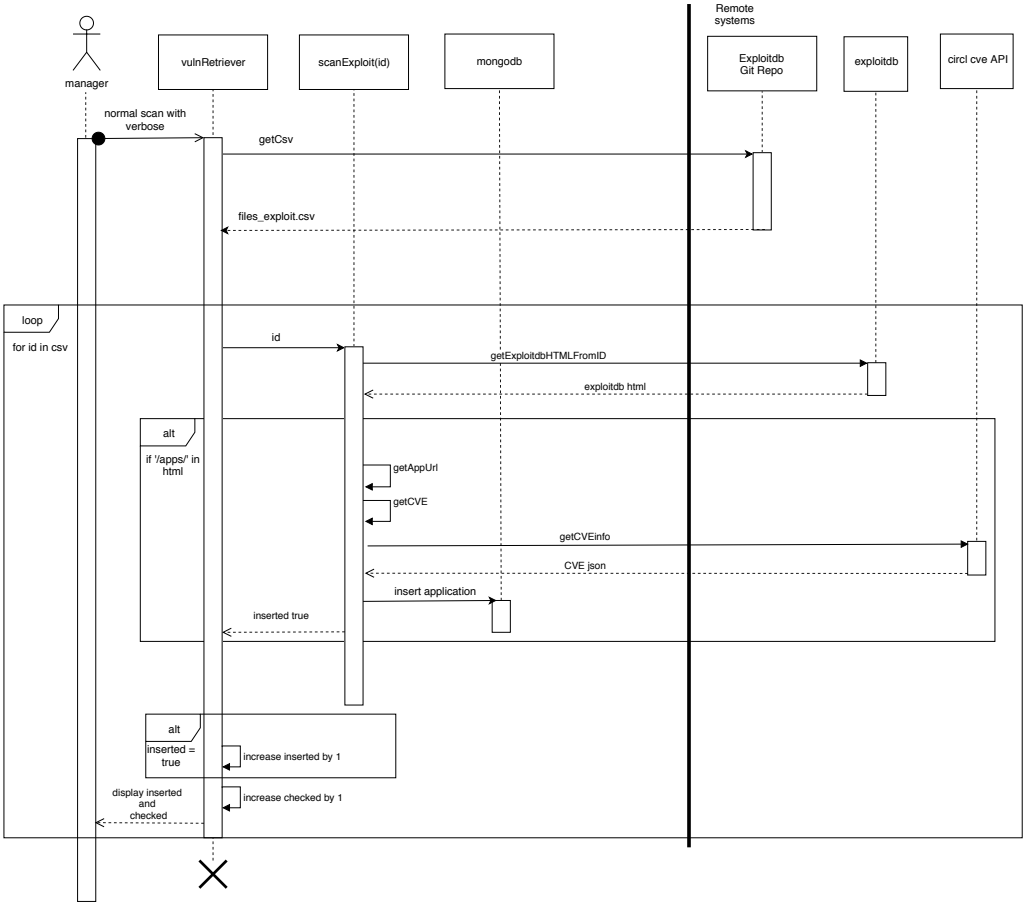


Figure [8] show the sequence of the vulnerability retriever, this illustrates the different components involved and how they cooperate to provide the required functionality.
The script will use the CSV file from the ExploitDB git repository to get the ID of each exploit. It will use this ID to scan the corresponding exploits web page for a vulnerable application. If an application is found, it will look for the CVE-id of the vulnerability and store this along with other required metadata. When the data is collected, the script will insert the data into the database and scan the next exploit.

4 Implementation

4.1 Introduction

PLEDs services runs on MongoDB, DreamFactory and Docker with several other dependencies, as Docker containers. Configuration files for each of the services in use are stored on a separate volume defined in the docker-compose file, meaning that the configuration is persistent even if a container is restarted/removed.

4.2 File storage

The file storage is integrated in SkyHigh as Swift Object Storage. By using Swift, PLED will have a container for all the files added to storage and this container is called pled_files. When a file is added, it is pushed to the container and added to the folder respective to the file type, whether it is a vulnerable application, CTF-challenge or malware.

Listing 4.1: Upload vulnerable file to swift container

```
$content = file_get_contents($_FILES['app_fileToUpload'
   ]['tmp_name']);
$filename = hash('md5', $_FILES['app_fileToUpload']['
   name'].date_timestamp_get(date_create()));
array_push($files, $filename);

try{
    $r = $s3->putObject([
        'Bucket' => 'pled_files/vulnerable_applications'
           ,
        'Key' => $filename,
        'Body' => $content
    ]);
```

The Swift container needs authentication to limit the access to the storage, this is implemented using EC2 credentials for OpenStack. Using the *openstack ec2 credentials create* command, an access key and a secret will be generated. The access key and secret are used with the Amazon S3Client PHP SDK[12] to gain access and insert/retrieve files in the storage container, also referred to as buckets.

Listing 4.2: Configuration of S3Client

```
$s3 = new S3Client([
    'region' => 'SkyHiGh',
    'endpoint' => 'https://swift.skyhigh.iik.ntnu.no/swift/
       v1/a056038438f14cd9b3b69744dc334774',
    'version' => 'latest',
    'credentials' => [
        'key' => '******************',
        'secret' => '******************',
    ],
    'use_path_style_endpoint' => true,
]);
```

To retrieve files, Swift has an API endpoint that can be accessed and the file can be retrieved. The S3Client supports downloading files in PHP so for the database web-interface this has been implemented by creating a pre-signed URL for the object. This URL will download the file when accessed.

Listing 4.3: Create download link with s3Client

```
$cmd = $s3->getCommand('GetObject', [
    'Bucket' => 'pled_files',
    'Key'    => 'ctf_challenges/'.$v['file_path']
  ]);
  $signed_url = $s3->createPresignedRequest($cmd, '+1 hour')
     ;
  $data['searchres'][$key]['file'] = $signed_url->getUri();
```

4.3 Docker Registry

The Docker container will be submitted through the database web-interface as a Dockerfile. Additionally to storing the Dockerfile, PLED will offer the Docker registry for easy Docker image management. The registry for storing Docker containers requires some persistent storage. A common way of implementing this is using persistent volumes in Docker, when doing this there is little persistence when taking the host server down, without using external volumes. Since PLED is already using the OpenStack Swift object storage service, it was decided to use this for the registry as well, since it will keep the registry storage persistent across server outages. The registry uses docker-compose for setup, this creates a stack consisting of a HAProxy loadbalancer, the Docker Registry, and a frontend where it's possible to browse the available images. Using external configuration, the registry is configured to store in an OpenStack Swift container. Listing 4.4 shows the configuration of the registry, related to how it communicates with Swift storage.

Listing 4.4: Storage setting for Docker Registry

```
storage:
  cache:
    blobdescriptor: inmemory
  swift:
    username: <OPENSTACK_USER>
    password: <OPENSTACK_PASSWORD>
    authurl: <OPENSTACK_AUTHURL>
    domain: <OPENSTACK_DOMAIN>
    tenantid: <OPENSTACK_TENTANT/PROJECT_ID>
    insecureskipverify: true
    region: <OPENSTACK_REGION>
    container: <OPENSTACK__REGISTRY_CONTAINER>
```

4.4 MongoDB

The database runs in a MongoDB replica set consisting of a primary node, a secondary node and an arbiter node. Each of these nodes run in separate Docker containers and are launched via a docker-compose file. In the design phase it was mentioned that SSL/TLS encryption would be enabled for database communication [3.2.3], but this was not integrated due to limited functionality in Bitnami's Docker image.

This means that communication with the database wil not be encrypted, but the authentication credentials exchanged within the replica set will be encrypted by default [13]. As the database is only exposed to the API and if enabled, MongoExpress. Both of these services are again protected with authentication, additionally will the front end be encrypted with SSL/TLS via HAproxy and a self-signed certificate. This is explain in further detail later in the report 4.5.2.

The dockef-compose file for starting the MongoDB replica set is depicted in the listing below. The configurations and database data is stored in an external volume hosted on OpenStack.

Listing 4.5: Docker-compose file for MongoDB Replica Set

```
version: '3.1'

services:
  mongodb-primary:
    image: 'bitnami/mongodb:latest'
    environment:
      - MONGODB_REPLICA_SET_MODE=primary
      - MONGODB_ROOT_PASSWORD=****
      - MONGODB_REPLICA_SET_KEY=****
    volumes:
      - mongodb-master-data:/bitnami
    ports:
      - "27017:27017"

  mongodb-secondary:
    image: 'bitnami/mongodb:latest'
    depends_on:
      - mongodb-primary
    environment:
```

```
      - MONGODB_REPLICA_SET_MODE=secondary
      - MONGODB_PRIMARY_HOST=mongodb-primary
      - MONGODB_PRIMARY_ROOT_PASSWORD=****
      - MONGODB_REPLICA_SET_KEY=****
      - MONGODB_PRIMARY_PORT_NUMBER=27017

  mongodb-arbiter:
    image: 'bitnami/mongodb:latest'
    depends_on:
      - mongodb-primary
    environment:
      - MONGODB_REPLICA_SET_MODE=arbiter
      - MONGODB_PRIMARY_HOST=mongodb-primary
      - MONGODB_PRIMARY_ROOT_PASSWORD=****
      - MONGODB_REPLICA_SET_KEY=****
      - MONGODB_PRIMARY_PORT_NUMBER=27017

volumes:
  mongodb-master-data:
    external:
      name: mongodb-master-data
```

PLED provides an additional tool for quick and easy database management, MongoExpress which is a MongoDB interface for interaction with the database. This can be enabled in the environment file for OpenStack Heat, by changing the *mgoexpress* variable from *false* to *true*.

4.5 DreamFactory

4.5.1 Administrative interface

PLED uses the docker-compose file developed and maintained by Bitnami, as suggested by DreamFactory on their Docker-Hub account [14]. The docker-compose file in listing [4.5.1], implements all the dependencies (MongoDB, MariaDB and redis) for DreamFactory, and enables volume storage for the configuration files for all services, including itself. For DreamFactory to be able to read the contents and modify files in the volumes, each folder for each service needs to be created inside the volume, and chown-ed with 1001. This switches the user from the default root to 1001, making the folders not owned by root as user 1001 is not a special user with special permissions. This is done with the following command:

```
chown -R 1001:1001 /bitnami
```

DreamFactory and its services is started with *docker-compose up*. It exposes port 443 for external access.

Listing 4.6: Docker Compose file for DreamFactory

```
  dreamfactory:
    image: 'bitnami/dreamfactory:latest'
    labels:
      kompose.service.type: nodeport
    ports:
      - '443:443'
```

```
    depends_on:
      - mariadb
      - redis
      - mongodb
    volumes:
      - dreamfactory-persistence:/bitnami
    networks:
        - pled_net
volumes:
  mariadb-persistence:
    external:
      name: mariadb-persistence
  redis-persistence:
    external:
      name: redis-persistence
  mongodb-persistence:
    external:
      name: mongodb-persistence
  dreamfactory-persistence:
    external:
      name: dreamfactory-persistence
```

4.5.2 Load balancing

HAproxy is implemented in front of DreamFactory, and if configured properly, will load balance traffic accross added nodes. At this point, its main purpose is to encrypt incoming and outgoing traffic. TLS/SSL is implemented at the load balancing level, redirecting every request to our DreamFactory service to a HTTPS request. A snippet of the configuration of HAproxy is shown below in listing [4.7].

Listing 4.7: HAProxy configuration snippets

```
frontend dfapi-https
        bind *:80
        bind *:443 ssl crt /etc/ssl/private/dfapi.pem
        bind *:4444 ssl crt /etc/ssl/private/dbweb-intf.pem
        # Applies self signed certificate
        redirect scheme https if !{ ssl_fc }
        # Redir HTTP req to HTTPS, makes DF HTTPS only.
        mode http
        acl a1 dst_port 4444
        use_backend dbweb if a1
    # If request is on port 4444, use backend dbweb.
        default_backend dfapi

backend dfapi
        mode http
        balance roundrobin
        option forwardfor
        option httpchk HEAD / HTTP/1.1\r\nHost:localhost
        server web-api DFAPIIP:80 check
        http-request set-header X-Forwarded-Port %[dst_port]
        http-request add-header X-Forwarded-Proto https if {
            ssl_fc }

backend dbweb
    server db-web-interface WEBINTERFACEIP:4444 check

listen stats
        bind *:1936
        stats enable
        stats uri /
        stats hide-version
        stats auth UNAME:PWD
```

In this project, a self-signed certificate is used to verify our service. It is not recommended to use a self-signed certificate in production, but its viable in this case for showcasing the functionality of our service. The certificate and key was generate by openssl with this command:

```
openssl req -x509 -nodes -days 500 -newkey rsa:2048 -keyout
   /etc/apache2/ssl/dfapiself.key -out /etc/apache2/ssl/
   dfapiself.crt
```

Followed by concatenating the certificate file and key file into a *.pem* file, a format for storing and sending cryptographic keys, certificates, and other data [15]. This file is used by HAProxy on incoming connections on port 443, as a certificate to authenticate and encrypt communication with DreamFactory.

4.5.3 Server side scripting

For some of the requirements from PEMA, a custom service in DreamFactory was needed. This service gives an endpoint to PEMA for listing all available application platforms

and types currently in the database. This was implemented using a scripting service in DreamFactory.

Listing 4.8: Custom search script in dreamfactory

```
verb = event.request.method
if verb != 'GET':
    raise Exception('Only HTTP GET is allowed on this
       endpoint.')
resource = event.resource

if resource =="":
    result = {'resource':['platform', 'type']}
elif resource =="platform" or resource =="type":
    url = 'mongodb/_table/vuln_applications'
    result = platform.api.get(url)
    data = result.read()
    jsonData = json.loads(data)
    resultlist = []
    for line in jsonData['resource']:
        if line[resource] not in resultlist:
            resultlist.append(line[resource].strip())
    result = {resource: resultlist}
else:
    raise Exception('Invalid or missing resource name.')

return result
```

This gives and endpoint for searching for platforms and types at
<dreamfactoryip>/api/v2/customsearch/<parameter>
The script requires the Python module *bunch*, the only alteration done here was adding a Dockerfile that installed this module when starting the DreamFactory service.

```
#DREAMFACTORY DOCKERFILE
from bitnami/dreamfactory:latest

RUN apt update -y && apt install python-pip -y && pip
   install bunch
```

4.5.4 Retrieving stored files

Stored files can be downloaded in the database web-interface.
To access the files from another back end, a s3Client needs to be set up in order to access the Object Storage. For temporary use, a cURL request can be sent to the Swift API with the users auth-token. This is not recommended since the auth-token has short expiry time and will have to be renewed often.

4.6 Database web-interface

The database web-interface is implemented using PHP, Javascript and HTML. To render the HTML from PHP, Twig is used as it is known by two of the group members from the course Web Technology and will be of great use when retrieving database content to be displayed. Twig is a template engine, meaning you can create static template files in your project. At runtime, Twig replaces variables in the template files with actual values and transforms the template into an HTML file sent to the client. The simple structure and setup of Twig, in addition to being familiar, makes it preferable over other solutions/frameworks as many of them have more advanced structures that would result in more time spent learning and structuring instead of developing.

The web server for the database web-interface runs in a Docker container, the image is built with the web documents needed and is started with docker-compose. When the server is deployed, a script will pull down the git repository containing the web page content, and run it with Docker.

Before running the container, a few dependencies has to be installed with Composer, which is a tool for dependency management in PHP, shown below.

```
#Add environment variable for Composer and installs
   dependencies with Composer
cd /home/ubuntu/pled/www && \
export COMPOSER_HOME="/home/ubuntu/pled/www" && \
composer install --ignore-platform-reqs
```

The command *composer install* will install all required dependencies listed in a *.json* file.

4.6.1 Authentication

A requirement for the database web-interface is that it should have user authentication when accessing it. This is implemented using HTTP basic authentication with the DreamFactory configured users. Basic is a very simple way of adding access control and since the database web-interface will eventually be integrated into PEMA, the group considered developing a whole authentication system with NTNUs LDAP servers as out of scope. When the user first tries to access the interface a prompt for username and password will be presented, when the user enters the values and submits a request to the DreamFactory API with the username and password, if the request is approved the user is authenticated and can access the interface.
The code below illustrates how the HTTP basic authentication checks for a set username and if not it will prompt for username and password

Listing 4.9: Authenticate user

```
if (!isset($_SERVER['PHP_AUTH_USER'])){
        //User pressed cancel
        header('WWW-Authenticate: Basic realm="PLED"');
        header('HTTP/1.0 401 Unauthorized');
        die('Unauthorized');
//Authenticate in Dreamfactory with username and password,
    die if false
} elseif (!aunthenticate($_SERVER['PHP_AUTH_USER'], $_SERVER
    ['PHP_AUTH_PW'], $ini_array['ip'])) {
        //User not authorized
        unset($_SERVER['PHP_AUTH_USER']);
        unset($_SERVER['PHP_AUTH_PW']);
        die('Unauthorized');
}
```

4.6.2 First Time Setup

When the database web-interface is first deployed, it will present a page where certain values needs to be entered, these values are the DreamFactory API key and host, as well as the AWS S3Client information such as key, secret, region and endpoint.
This configuration is required to use the interface and when entered will generate a configuration file that the back end will use. Listing 4.10 shows how the back end checks if the configuration file is generated.

Listing 4.10: Check if config file is set

```
if (file_exists($_SERVER['DOCUMENT_ROOT']."/conf/phpconfig.
    ini")) {
  $ini_array = parse_ini_file("./conf/phpconfig.ini", true);
} else {
  echo $twig->render('generateConfig.html', array());
  die();
}
```

Listing 4.11 shows how the *phpconfig.ini* file is generated on first time setup

Listing 4.11: Generate the config file on first time setup

```
  $data = 'api_key = '.$_POST['apikey'].PHP_EOL.
      'ip = '.$_POST['apiurl'].PHP_EOL.
      's3_key = '.$_POST['s3key'].PHP_EOL.
      's3_secret = '.$_POST['s3secret'].PHP_EOL.
      's3_region = '.$_POST['s3region'].PHP_EOL.
      's3_endpoint = '.$_POST['s3endpoint'].PHP_EOL;
  fwrite($file, $data);
  fclose($file);
```

4.6.3 Front end

As shown and discussed in the design chapter [3.5], the database web-interface consists of several different pages for viewing and modifying contents of the database. The logical index file for the database web-interface is the outermost file called *index.php*. This file is responsible for creating and retrieving the variables to be displayed in the index HTML page, *databaseManagementPage.html*, more on this in the back end section. This index page displays a container with three tabs for each of the pages. One page for viewing, adding and modifying content. The purpose of this page is simply to be of aid for an instructor using the interface, and a third option to MongoExpress and the API Docs in DreamFactory for finding and displaying content.

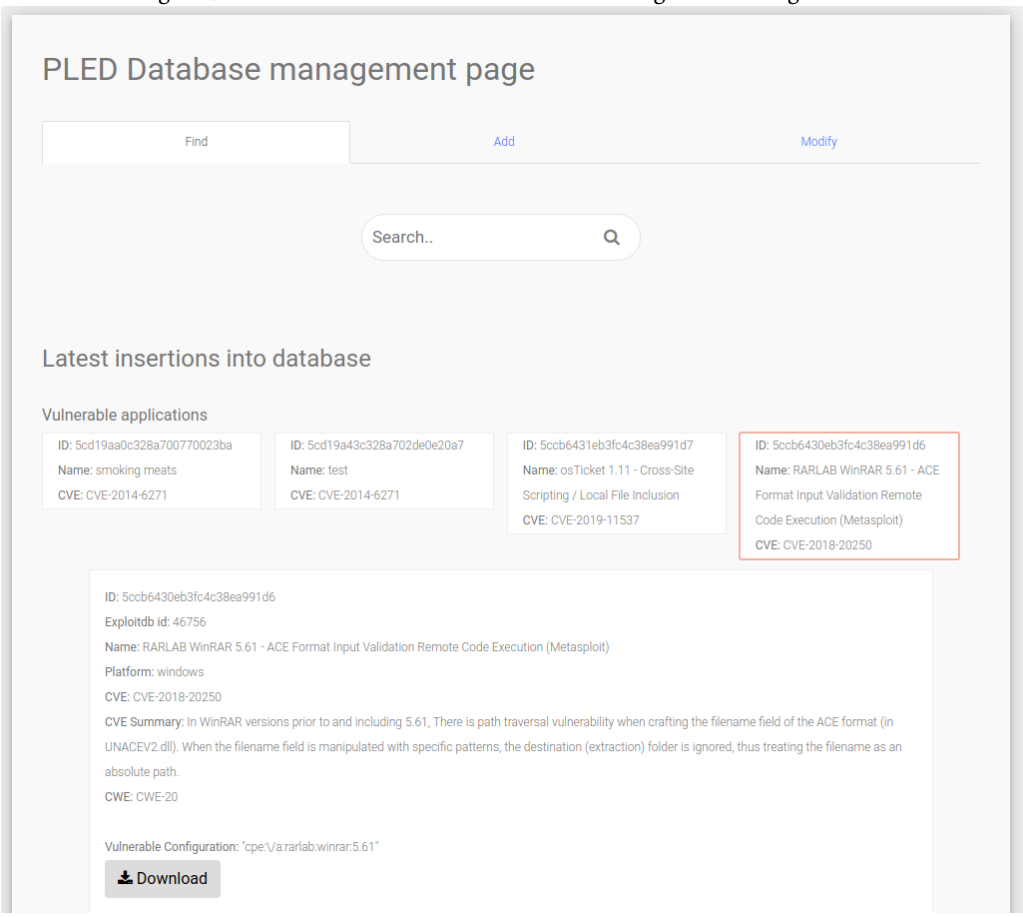
View/Find content

The first tab presents a search bar and the latest insertions into the database respectful to its type (vulnerable application, CTF-challenge or malware) as shown in figure [9]. To use the search bar, you simply enter any value from any fields that could be stored in the database and hits are displayed underneath the input field. Each hit acts a button which displays a collapsible, more detailed view on its data stored in the database, when clicked. Click again and the collapsible closes. This button and collapsible is also used for displaying the contents of the latest insertions into the database. This is a form of the Progressive Disclosure Design Pattern [16] where the goal is to reduce unnecessary data displayed to the instructor upon opening the interface. In addition, a download button is included in the collapsible that will download the file connected to the chosen object displayed. Listing [9] shows the method for creating collapsibles for each element from search and for the latest insertions. It also displays the method for looping through data retrieved by Twig using for loops.

Listing 4.12: Twig for loop and collapsibles

```
{% if vuln_applications %}
    {% for application in vuln_applications %}
        <button class="collapsible">
            <b>ID:</b> {{ application._id }} <br>
            <b>Name:</b> {{ application.application_name }}<
                br>
            <b>CVE:</b> {{ application.cve }}<br>
        </button>
        <div class="content" style="display: none; overflow:
            hidden;">

            <b>ID: </b>{{ application._id }}<br>

            {% if application.exploitdb_id %}
            <b>Exploitdb id: </b>
            {{ application.exploitdb_id }}<br>
            {% endif %}

            <b>Name: </b>{{ application.application_name }}<
                br>

            {% if application.summary %}
            <b>Summary: </b>
            {{ application.summary }}<br>
            {% endif %}

            ...

        </div>
    {% endfor %}
{% else %}
    No vulnerable applications currently in database.
{% endif %}
<script>
    var coll = document.getElementsByClassName("collapsible"
        );
    var i;

    for (i = 0; i < coll.length; i++) {
        coll[i].addEventListener("click", function() {
            this.classList.toggle("active");
            var content = this.nextElementSibling;
            if (content.style.display === "block") {
                content.style.display = "none";
            } else {
                content.style.display = "block";
            }
        });
    }
</script>
```

Figure 9: Database web-interface forms for finding and viewing content



Add content

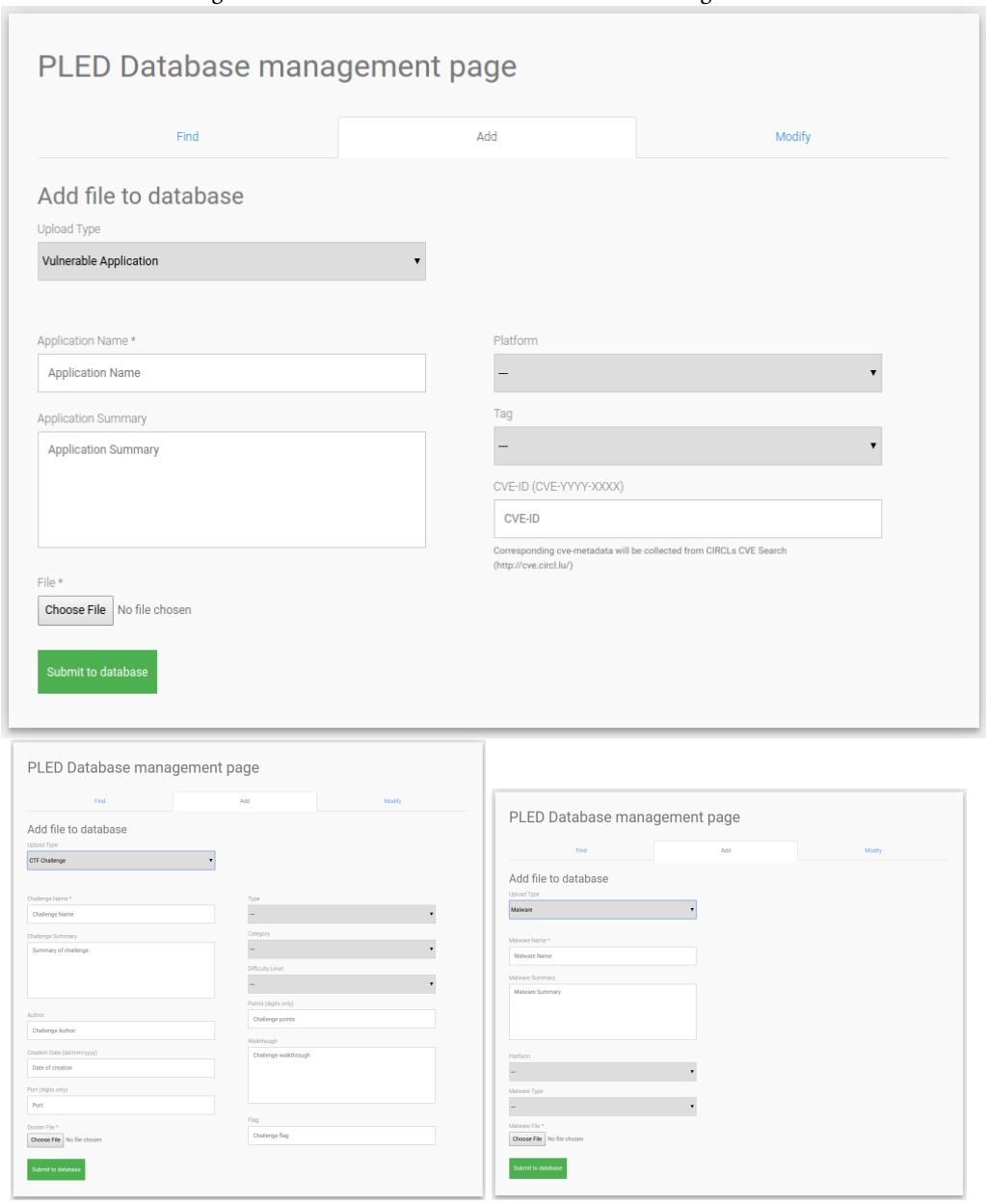
In the second tab the instructor is presented with forms for adding content to the database. A dropdown is presented for choosing what type is to be uploaded between vulnerable applications, CTF-challenges and malware. When the instructor chooses a type, the respective form is displayed using the JavaScript shown in listing [4.13]. Each form contains the input fields for the values described in the meta model for each type, see figure [3].

Listing 4.13: Display form given in dropdown

```
<script>
    if (localStorage.getItem('form_frame')) {
        $("#uploadtypeselector option").eq(localStorage.
            getItem('form_frame')).prop('selected', true);
            v = localStorage.getItem('form_frame');
            changeview(v);
        }

    $("#uploadtypeselector").change(function() {
        val = $(this).val();
        deleteview(v);
        changeview(val);
        v = val;
    });

    function changeview(val) {
        console.log("changed to: " + val);
        document.getElementById(val).style.display = "block"
            ;
        document.getElementById('uploadtypeselector').value
            = val;
        localStorage.setItem('form_frame', val);
    }

    function deleteview(val) {
        console.log("Deleted view: " + val);
        document.getElementById(val).style.display = "none";
    }
</script>
```

When adding a vulnerable application, it is possible to add CVE as metadata. If included, it is checked that it is valid by CIRCLs CVE Search terms. If the response is empty, we conclude it is an invalid/non-existing CVE and the instructor needs to either not include CVE or enter a valid one. In addition to a valid CVE, some input fields are mandatory such as "name" and "file". These fields are mandatory for all the types to upload and will return a message to the instructor if empty on submit.

Figure 10: Database web-interface forms for adding content

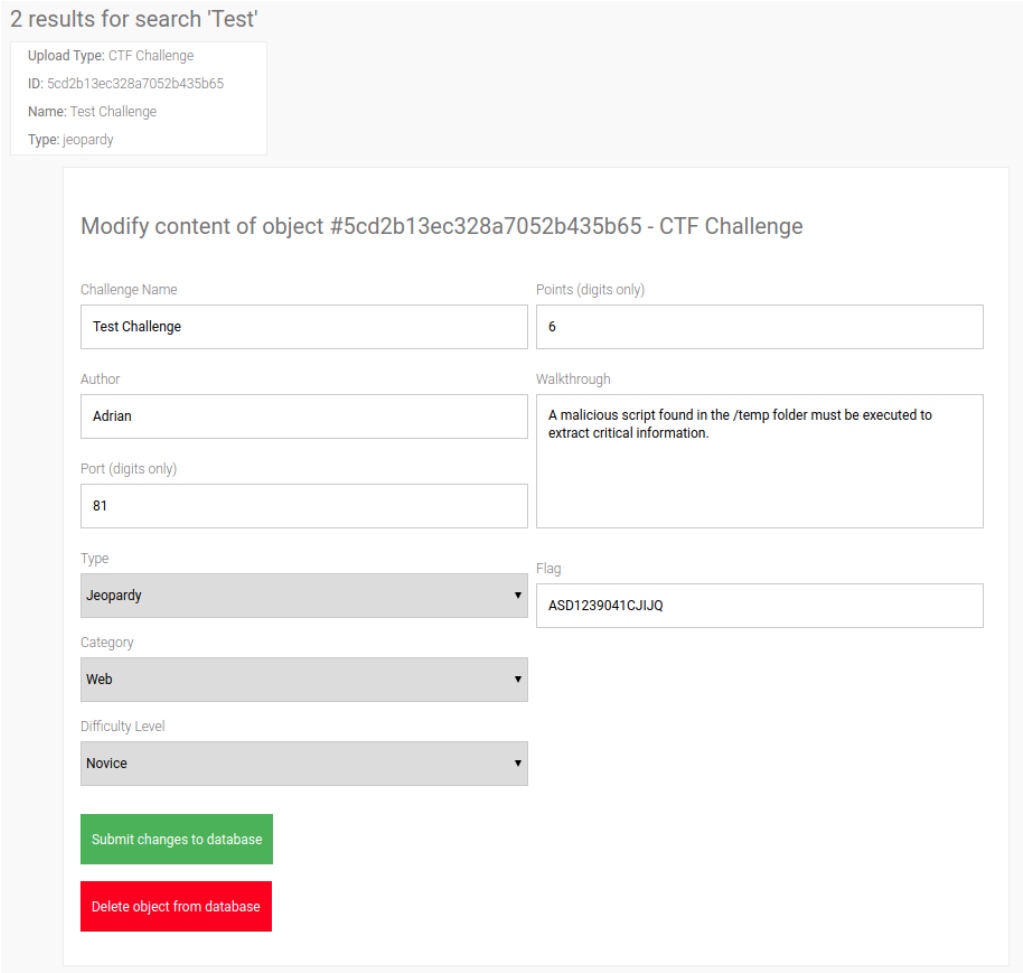


Modify/Delete content

The modify section presents the user with a search bar, previously explained in View/Find content. Again, the user can search for any element present in the JSON document stored in the database. The result is displayed as collapsible views which can be expanded by clicking on them. When the element is expanded, the user is free to modify its content, or add/remove values that has or hasn't been set. Whether it's a vulnerable application, a malware or a CTF-challenge, different values can be modified or added/removed specific to the type of element. When an element is modified, the change can be submitted to the database. On the other hand, if an element is up for deletion, it is the same procedure

as modify for finding the right element. But instead of modifying the element, one must click the *Delete object from database* button. An example of the modify form can be shown in figure 11.

Figure 11: Database web-interface modify content



4.6.4 Back end

The back end consists of a configuration file, an index page, an API class and several PHP files for database actions. Before using the services offered through the web-interface, a number of credentials and other values has to be inserted to the environment file prior to accessing the services in order to authenticate yourself. Below is an example on how the configuration file would look like. The S3 credentials are generated through OpenStack.

Listing 4.14: Example of php config file

```
api_key = YOUR-DREAMFACTORY-API-KEY-HERE
ip = DREAMFACTORY-IP-HERE
s3_key = OPENSTACK-S3-CREDENTIALS-KEY-HERE
s3_secret = OPENSTACK-S3-CREDENTIALS-SECRET-KEY-HERE
s3_region = OPENSTACK-REGION
s3_endpoint = ENDPOINTURL
```

The variables defined in the file above are used as an array wherever its needed, as for example:

```
#Load the phpconfig.ini file containing credentials, IP's
    and more.
$ini_array = parse_ini_file("../conf/phpconfig.ini", true);

// Vuln_applications
$json = file_get_contents('http://'.$ini_array["ip"].'/api/
    v2/mongodb/_table/vuln_applications?limit=4&order=_id%20
    DESC&api_key='.$ini_array["api_key"]);
```

Index.php retrieves data from the database using the API class, that requests DreamFactory for data to be used in the rendered HTML file. It is responsible for collecting the latest insertions which is the only data needed when the page is first loaded. The rest of the back end will only be executed by an action of the instructor.

API Class

For accessing the API, the back end uses an API class, the class does all operations concerning the API such as adding, searching, deleting and modifying content. The class is created to make the code less repetitive and simplify the use of the API.
Listing 4.15 shows an example use of the API class in use:

Listing 4.15: API class usage example

```
//Create Api object, parameter is the php config created
    during first time setup
$api = new Api($ini_array);
//For each collection, get data and render with Twig
foreach($collections as $collection){
  $data[$collection] = $api->getContents($collection);
}
```

This will retrieve the content of each collection and display on the web page.

Search

When an instructor searches for content in the database, *search.php* is responsible for handling the action and returning the results to the frontend using Twig. It first needs to find the value from the search input and then execute the search, for searching the API method *search()* is used to filter the request for the specified search, the result of the search uses the methods *getVuln_applications*, *getCtf_challenges* and *getMalware* to format the data to be rendered with Twig. Listing 4.16 shows the creation of an object with search-results when search input equals $_POST['search'].

Listing 4.16: Sourcecode for search

```
$search = $_POST['search'];
$search = urlencode($search);
$data['searchres'] = [];
foreach($collections as $collection){
  $data['searchres'][$collection] = $api->search($search,
      $collection);
}
```

```
//API search() method
public function search($search, $collection) {
    $data = [];
    if($this->collectionExists($collection)){
        switch ($collection) {
            case 'vuln_applications':
                $json = file_get_contents(<
                    DreamFactory_URL_WITH_FILTER_FOR_$SEARCH)
                    ;
                $data = $this->getVuln_applications($json);
                break;
            case 'ctf_challenges':
                $json = file_get_contents(<
                    DreamFactory_URL_WITH_FILTER_FOR_$SEARCH)
                    ;
                $data = $this->getCtf_challenges($json);
                break;
            case 'malware':
                $json = file_get_contents(<
                    DreamFactory_URL_WITH_FILTER_FOR_$SEARCH)
                    ;
                $data = $this->getMalware($json);
                break;
            default:

        }
    }
    return $data;
}
```

Download

When the data is retrieved in the back end, a pre-signed-URL [17] is generated for each object, this URL is added as a download button in the collapsible of each element. When clicked the file will download with file name corresponding to the key in Swift Storage. A way of implementing this is shown in listing [4.2].

Upload

Each form has its corresponding PHP file to handle the upload action from the instructor. It first check if mandatory fields are set using *isset()*. If they are empty, both a message and the fields that are empty are returned to the frontend to let the user know what is missing. In addition it also returns the values for any of the other inputs which was set on the request so that the instructor do not need to re-fill the entire form a second time. If everything that need to be entered is set, the metadata can be uploaded to MongoDB through the API method *insert()* and the file is uploaded to Swift.

Listing 4.17: Upload content to MongoDB and Swift

```
$ini_array = parse_ini_file($_SERVER['DOCUMENT_ROOT']."/conf
    /phpconfig.ini", true);

$s3 = new S3Client([
    'region' => $ini_array['s3_region'],
    'endpoint' => $ini_array['s3_endpoint'],
```

```
    'version' => 'latest',
    'credentials' => [
    'key' => $ini_array['s3_key'],
    'secret' => $ini_array['s3_secret'],
    ],
    'use_path_style_endpoint' => true,
]);

try{
    $r = $s3->putObject([
        'Bucket' => 'pled_files/vuln_applications',
        'Key' => $filename,
        'Body' => $content
    ]);

    $resource = json_decode('{}');
    $resource->resource = [];
    $metadata = json_decode('{}');

    // Add values from $_POST, repeat for all values
    if (!empty($_POST['someValue']) {
        $metadata->value = $_POST['someValue'];
    }

    // Create the request json body
    $resource->resource[0] = $metadata;
    $body = json_encode($resource, true);
    //API class call
    $api = new Api($ini_array);
    $res = $api->insert($body, 'vuln_applications');
    $obj = json_decode($res['response'], true);
    // Check for errors
    if($res['response'] === FALSE){
        die($res['error']);
        $data['error'] = 'dferror';
        echo $twig->render('databaseManagementPage.html',
            $data); // Render html
    }

    // Check for errors
    if($response === FALSE){
        die(curl_error($ch));
        $data['error'] = 'dferror';
        echo $twig->render('databaseManagementPage.html',
            $data); // Render html
    }

    if ($obj['error']['code'] != 200) {
        $data['uploaded'] = 'false';
    } else {
        $data['uploaded'] = 'true';
    }
```

```
    header('Location: ../index.php?uploaded=' . $data['
        uploaded']);

} catch (S3Exception $e) {
    echo $e->getMessage() . PHP_EOL; // Not to be used in
        production
    $data['error'] = 's3Exeption';
    echo $twig->render('databaseManagementPage.html', $data)
        ; // Render html
}
```

Listing 4.18 shows the API method for inserting into a collection

Listing 4.18: API class insert method

```
public function insert($data, $collection) {
    $ch = curl_init();
    $options = array(CURLOPT_URL => 'http://'.$this->
        ini_array["ip"].'/api/v2/mongodb/_table/'.$collection
        .'/',
          CURLOPT_HTTPHEADER => array('X-DreamFactory-API-Key
              :'.$this->ini_array["api_key"],
                    'Content-Type: application/json'),
          CURLOPT_POST => 1,
          CURLOPT_POSTFIELDS => $data,
          CURLOPT_RETURNTRANSFER => 1
         );

    curl_setopt_array($ch, $options);

    // Send the request
    $response = curl_exec($ch);
    $res['response'] = $response;
    if($response === FALSE) {
```

Modify/Delete

When the back end is called to modify metadata for a vulnerable application, CTF-challenge or malware, it first creates the request body with each value from the form and then calls the function to send the PATCH-request with corresponding collection and the created body as arguments. The function creates the cURL and sends the request to DreamFactory via the API class. If everything goes as planned it changes the header location to *index.php*.

Listing 4.19: Send patch request and check response

```
function sendPatchRequest($collection, $body, $apikey) {
  $dfapikey = 'X-DreamFactory-API-Key:'.$ini_array;
    //API class call
    $api = new Api($ini_array);
  $res = $api->patch($body, 'vuln_applications');
  //Decode response
  $obj = json_decode($res['response'], true);
    // Check for errors
```

```
    if($res['response'] === FALSE){
        die($res['error']);
        $data['response'] = 'Something is wrong, check
            DreamFactory configurations';
        echo $twig->render('databaseManagementPage.html',
            $data); // Render html
    }

    // Error handling for code != 200
    if (!empty($obj['error']['code']) && ($obj['error']['
        code'] != 200)) {
        $data['updated'] = 'false';
    } else {
        $data['updated'] = 'true';
    }
    header('Location: ../index.php?updated=' . $data['
        updated']);
}
```

Listing 4.20 shows the API method for patching a collection.

Listing 4.20: API class patch method

```
public function patch($data, $collection) {
  $ch = curl_init();
    $options = array(CURLOPT_URL => 'http://'.$this->
        ini_array["ip"].'/api/v2/mongodb/_table/'.$collection
        .'?filter=_id='.$_POST['_id'],
          CURLOPT_HTTPHEADER => array('X-DreamFactory-API-Key
              :'.$this->ini_array["api_key"],
                'Content-Type: application/json'),
          CURLOPT_CUSTOMREQUEST => 'PATCH',
          CURLOPT_POSTFIELDS => $data,
          CURLOPT_RETURNTRANSFER => 1
         );

    curl_setopt_array($ch, $options);
    // Send the request
    $response = curl_exec($ch);
    $res['response'] = $response;
    if($response === FALSE) {
        $res['error'] = curl_error($ch);
    }
    return $res;
}
```

4.7 Vulnerable Application Retrieval

4.7.1 Using ExploitDB search API

ExploitDB provides a program for searching their database, called SearchSploit. This program allows for search with parameters such as platform, type and id. SearchSploit does not provide a method of searching for CVE or to check if an exploit has a vulnerable application. So we needed something that could give us more data.

4.7.2 Scraping exploitdb: vulnRetriever.py

Since the SearchSploit program is inadequate for PLEDs purposes, another way of getting the data is required. Scraping the HTML of the ExploitDB website works by getting the id from a CSV file located on ExploitDB Github repository. This id is used to create an URL that is scanned for a vulnerable app and to see if it is verified

Listing 4.21: Check for vulnerable application and verification

```
url = EXPLOITURL + id
#Request the page with the selected headers
page = requests.get(url, headers=HEADER)
tree = html.fromstring(page.content)
hasapp = tree.xpath("//a[re:match(@href,␣'/apps/')]",
        namespaces={"re": "http://exslt.org/regular-
           expressions"})
#Checks if the exploit is verified, using the checkmark
   class
isverified = tree.xpath("//i[contains(@class,␣'mdi-check')
   ]")
```

If the app has an application URL and is verified, the CVE id is extracted from the page and used with CVESearch to gather the required metadata of a vulnerability

Listing 4.22: Search for CVE data

```
      #search for cve id
    if 'CVE' in link.attrib['href']:
      vulnData['cve'] = 'CVE-' + link.text_content().strip
         ()
      cve_search = CVESearch()
      #lookup cve
      data = cve_search.id(vulnData['cve'])
      #get data from returned json
      if data:
        if 'summary' in data:
            vulnData['cve_summary'] = data.get('summary')
        if 'cvss' in data:
            vulnData['cvss'] = data.get('cvss')
        if 'cwe' in data:
            vulnData['cwe'] = data.get('cwe')
        if 'impact' in data:
          vulnData['impact'] = data.get('impact')
        if 'msbulletin' in data:
          vulnData['msbulletin'] = data.get('msbulletin')
        if 'vulnerable_configuration_cpe_2_2' in data:
          vulnData['vulnerable_configuration'] = data.get(
             'vulnerable_configuration_cpe_2_2')
```

When all the data is collected the application is inserted into the MongoDB collection

The scraper was originally written in PHP, but was instead exchanged for Python as it requires less dependencies, it's simpler to use and it's more flexible.

Configuration

vulnRetriever is configurable from the *vulnRetriever.ini* file, .INI format is preferred since it is a well known format for configuration files and can divide the configuration into sections that is parse-able by the Config-parser Python module[18], here the user can specify the database configuration, the date from when to start searching (what date the app was published) or other configurations related to the search that would not need to be altered unless something in the exploit-db.com website is changed.

Listing 4.23: Example of configuration

```
[SETTINGS]
StartDate = <PUBLISHED_DATE_TO_START_FROM> #eks 2000-01-01
Checkfile = <NAME_OF_CHECKED_FILE> #eks checked.txt

[SCANNER]
Exploiturl = <EXPLOITDB_BASE_URL> #https://www.exploit-db.
    com/exploits/

[DATABASE]
Mongo_ip = <MONGODB_HOST> #eks 192.168.10.10 OR hostname
Mongo_user = <MONGODB_USER>
Mongo_pw = <MONGODB_PASSWORD>
Mongo_database  = <DATABASE_TO_STORE_DATA>
Mongo_collection = <DATABASE_COLLECTION>

[CSV]
Csvurl = <EXPLOITDB_CSV_LOCATION> #https://raw.
    githubusercontent.com/offensive-security/exploitdb/master
    /files_exploits.csv
```

vulnRetriever.py options

The scraper allows for some arguments to be added when running it:

- *-q, –quick* Scan and simultaneously store scanned ids in a file, so that if interrupted the program can start from where is was saved
- *-u, –update* Run in update mode, checks PLED for the latest added exploit, by its date. Only adds exploits from ExploitDB that was added after said date.
- *-v, –verbose* Run with verbose outputted to the console
- *-l, –log* Output in logging format, for when running automatically on a schedule
- *-i, –id* Scan a specific ExploitDB id for a vulnerable application

5 Deployment

5.1 Heat Template

OpenStack takes care of networking, routing, servers and more, based on the applied Heat template. Instead of building the whole infrastructure manually from the bottom up, a Heat template from Erik Hjelmås' GitHub was used as a basis for PLEDs infrastructure.

The top level Heat template, *top_pled.yaml*, defines two resources, one resource named infrastructure and one named workers. The infrastructure resource invokes *infrastructure.yaml*, which builds the network and network components. The worker resource invokes its own *workers.yaml* file, which is dependent on the previously mentioned infrastructure resource. This is because the network needs to be up and running before assigning servers with IP addresses.

Before starting the stack, one must fill out the environment variables located in *pled_top_env.yaml*. This defines parameters for the Swift username and password among other things. The parameters are further piped into the needed resources, which sends them as parameters into custom *bash* scripts. This is done with the intention of improving security by having passwords and user names passed as variables at creation, instead of storing them as clear text in the scripts executed on each server.

5.1.1 Source code

All the source code used for the workers are located in a public Git repo, this is because it makes setup and configuration on the workers easy by cloning down the repository on the respective instances. And the source code will always be up to date, making change management more efficient.

5.1.2 infrastructure

This resource creates the network infrastructure consisting of a network, address pool, a router with ports, a default gateway and a security group. The security group enlists all necessary ports to enable services outside the LAN to communicate with PLED. Only when the resource *infrastructure* is finished, the next resource *workers* is launched. Below is an example on how the security group is built, specifically how the port for SSH, port 22, and the port for MongoDB, port 27017, is created:

```
security_group_linux:
    type: OS::Neutron::SecurityGroup
    properties:
      description: Create a PLED security group
      name: linux_PLED
      rules:
        - remote_ip_prefix: 0.0.0.0/0
          protocol: tcp
          port_range_min: 22
          port_range_max: 22
```

```
        - remote_ip_prefix: 0.0.0.0/0
          protocol: tcp
          port_range_min: 27017
          port_range_max: 27017
```

5.1.3 workers

The *workers* resource invokes the *workers.yaml* file, which creates defined servers, and each server executes their own bash script in order to configure server specific settings not defined in the docker-compose files. The set of workers makes up the PLED service. The Heat template can provide defined parameters to the bash script, e.g. an IP-address of a specific instance. An example is provided below:

Manager

```
manager-server:
    type: OS::Nova::Server
    properties:
      name: manager-server
      image: { get_param: image_linux }
      flavor: { get_param: flavor_manager }
      key_name: { get_param: key_name }
      networks:
        - port: { get_resource: manager_port }
      user_data:
        str_replace:
          template: { get_file: scripts/manager_boot.bash }
          params:
            df_ip: { get_attr: [web-api, networks,
                pled_admin_net, 0] }
            db_ip: { get_attr: [dbreplica-t, networks,
                pled_admin_net, 0] }
            pwd: { get_attr: [bkup_pw, value] }
            mongodbrootpw: { get_attr: [mongodb_root_pw,
                value] }

bkup_pw:
  type: OS::Heat::RandomString
  properties:
    length: 16
```

The resource named *manager-server* is launched with the appropriate settings. In the property user_data, a custom bash script *manager_boot.bash* is invoked, and with it, some important parameters. *df_ip* is the parameter containing the IP of the instance running the DreamFactory service, and *db_ip* contains the IP address of the database. The parameter *pwd* is a randomly generated string. Similar to pwd, *mongodbrootpw* is the MongoDB root password needed to run *mongodump* commands for backup.

The bash script *manager_boot.bash* uses the parameters db_ip and df_ip to automatically insert the correct IP-addresses to the backup jobs in crontab. And the pwd parameter is used as a password to create the user *backupbot*. This allows the manager-server to relay the public key through ssh-copy-id as a first-time operation with username and password to establish a credential free login with SSH. The process is done on the servers

meant to have a backup job. An example of this can be seen in the snippet below:

```
#Copy SSH-key to relevant machines, as the Ubuntu user
sudo -Hu ubuntu /usr/bin/sshpass -v -p pwd ssh-copy-id -i /
    home/ubuntu/.ssh/bkup_key.pub -o StrictHostKeyChecking=no
     backupbot@df_ip
sudo -Hu ubuntu /usr/bin/sshpass -v -p pwd ssh-copy-id -i /
    home/ubuntu/.ssh/bkup_key.pub -o StrictHostKeyChecking=no
     backupbot@db_ip

#Adding cronjob for backup of DreamFactory
crontab -l | { cat; echo "0 0 * * * /usr/bin/ssh -i /home/
    ubuntu/.ssh/bkup_key backupbot@df_ip '/bin/bash /home/
    ubuntu/pled/misc/backup_dfvol.sh'"; } | crontab -

#Adding cronjob for backup of MongoDB
crontab -l | { cat; echo "0 0 * * * /usr/bin/mongodump --
    username root --password mongodbrootpw --
    authenticationDatabase admin --host db_ip --port 27017 --
    db pled --forceTableScan -o /backup/mongodb_\$(date +\%Y_
    \%m_\%d) --gzip
"; } | crontab -
```

Other servers

Furthermore, the following resources spawns these servers:

- balancer - Balancer server in front of the API and the database web-interface.
- web-api - The API, running on the DreamFactory framework.
- dbreplica - A replication enabled MongoDB server for storage of metadata.
- docker-registry - Private Docker Registry.
- database web-interface - The database web-interface for manual interaction with the database, and consequently the file storage.

The file proceeds with creating three persistent volumes and attaches them to the appropriate servers. One backup volume, one volume for the database and one volume for the DreamFactory service.

Floating IP's are assigned to:

- balancer - Expose the API and the database web-interface to services outside the LAN.
- dbreplica - A floating IP is assigned to dbreplica, in case an administrator want to use the MongoExpress tool along with database.
- docker-registry - Exposes the front-end of the Docker Registry service.
- manager-server - Externally reach the manager-server, for general management on the infrastructure.
- web-api - Expose the DreamFactory admin panel and the API documentation.

Database server

In regards to the database server, the vulnRetriever has already been running and pre-filled a database that is restored at the creation of the database server. By doing this, the administrator(s) of PLED will not have to run the vulnRetriever, which can take up to several hours. This is done by executing *mongorestore*, a tool for restoration of MongoDB

dumps like in the snippet below:

```
#Dump exploitdb applications into database
mongorestore --username root --password mongodbrootpw --
    authenticationDatabase admin --host localhost --port
    27017 --collection vuln_applications --db pled /home/
    ubuntu/pled/misc/vuln_applications_edb.bson
```

As previously mentioned in [4.4], if the environment variable *mgoexpress* is set to *true*, this code will be executed. It uses the variable *mongodbrootpw* to set the root password for the MongoDB server, and basic authentication is enabled with credentials added to the Heat environment file.

```
#Start mongo express if specified
if [ mgoexpress ]; then
     docker run -d --network docker_default -e
        ME_CONFIG_MONGODB_SERVER=mongodb-primary -e
        ME_CONFIG_MONGODB_ADMINUSERNAME='root' -e
        ME_CONFIG_MONGODB_ADMINPASSWORD='mongodbrootpw' -e
        ME_CONFIG_BASICAUTH_USERNAME='basicuname' -e
        ME_CONFIG_BASICAUTH_PASSWORD='basicpw' -p 8088:8081
        mongo-express
```

6 Security

6.1 Introduction

When creating a vulnerability database and populating it with vulnerable systems and applications, there are some security aspects to take into consideration. These aspects need to be assessed and proper actions needs to be taken to ensure that all systems in the network remain healthy when deploying and using the PLED system.

6.2 Vulnerable application storage

When storing vulnerable apps there are some risks to consider. One risk is that a vulnerable application will be used in a production environment by some fault or misuse. This risk is not very likely and with the access control for the database and the API, it will be an accepted risk. When adding and retrieving files from the Swift container the user must authenticate with EC2 credentials, this is to prevent unauthorized use of the file storage.

6.3 CTF-challenge storage

Able students or participants of a CTF-challenge might be enticed to try to access the database in order to get the flags. To minimize this risk we have implemented user authentication on the database with a secure password generated by Heat.

6.4 Malware storage

The dormant malware does not pose a risk unless its executed, and how its executed in lab environments is outside PLEDs scope. However, introducing malware to platform is never completely safe. The human factor is always a risk, whether its intentional or not. Someone might use one of these malwares on a outdated machine which is exposed to the world wide web, and in a worst case scenario, infect parts of a network or similar.

To stop unintentional use and malicious acts, access control has been enabled for the DreamFactory administrative interface, DreamFactory API, Database web interface (through PEMA) and Swift with access via DreamFactory. By only allowing specific people and services to access the repository, the risk introduced by storing malware is significantly reduced. To further reduce the residual risk, traffic to and from DreamFactory is encrypted with SSL/TLS. The residual risk is at an acceptable level.

6.5 Vulnerability Retriever

The vulnRetriever is using HTML scraping for collecting the vulnerable applications, this tends to put stress on the website being scraped. Because of this, the script should be ran with caution and have a small delay between the individual searches in order to avoid flooding the system, and resulting in blocking NTNUs IP range from accessing the site. To make sure that the script won't cause any problems some warnings are implemented

when running the script to make sure it is not started by accident. When deploying the solution the script will run fully automated.

6.6 REST API

As the API has access to both file storage and the MongoDB server, it is vital to ensure that only the necessary services has access to it. DreamFactory takes care of API-key generation with role based privileges, which in turn is provided to the services which needs to communicate with the API.

6.7 DreamFactory admin interface

The DreamFactory service is open source, making its inner workings visible to anyone with interest in doing code review, making it extremely unlikely for it to exist backdoors or other maliciously intended code. None of the data flowing through DreamFactory is sent to their servers, as the service runs locally at NTNUs SkyHigh platform. By using this automated framework, complexity is reduced when multiple services interact with each other, as its all from one provider, opposed to several different services combined by students in the group.

These features combined, along with several others like the already mentioned user management and role based access, the group is comfortable with using this framework from a security perspective, and trusts the services provided by DreamFactory [19].

6.8 Database web-interface

The database interface deployed with PLED is created for testing purposes and proof of concept, in a production environment proper access control should be implemented as when the interface is accessed there is no as of now no limit to what a user is authorized to do, this can be done with roles in DreamFactory since the database web-interface is using the API endpoints to perform actions.

7 Operations

7.1 Backup

Data that needs to be backed up are located on two persistent storage volumes attached to the respective machines, one for DreamFactory, and one for MongoDB. For both jobs, a pull model is utilized for safety, in order to avoid multiple machines with write-access to the manager server. For both jobs, a default backup rotation of 14 days is set in place and anything older than 14 days is deleted. The *find* program is used to decide what files to delete by looking at the files metadata, specifically its creation date. A cronjob starts the process after previously executed backup-jobs are finished, and deletes files older than 14 days as seen below:

```
0 1 * * * /usr/bin/find /backup/dfbackup/ -mtime +14 -type f
    -delete
```

The file names are appended with a date/time-stamp: *YYYY_mm_dd* for better human readability.

The aforementioned backup plan in the design chapter [3.2.4], was adapted into a better suited plan for PLED and its data. Incremental backup was dropped due to the small file sizes, and the single point of failure still stands, but the backup data is stored on an external volume. The server might fail, but the backup data is otherwise safe and is fully recoverable.

7.1.1 DreamFactory

The volume attached to the machine which runs the DreamFactory service is backed up once per day by compressing all files on the volume with *tar*, which is a tool for compressing files or folders. These files are relatively small, and will remain small even if the application grows, as the contents mainly consists of configuration files and user management related files for a user database. A cronjob runs the *tar* script once per day:

```
0 0 * * * /usr/bin/ssh -i /home/ubuntu/.ssh/bkup_key
   backupbot@df_ip '/bin/bash /home/ubuntu/pled/misc/
   backup_dfvol.sh'
```

The manager runs rsync, which is more fault tolerant than SCP since it can for example retry the execution in case of a network outage. Rsync is executed shortly after prior backup jobs, in order to synchronize the backup folder on the server running DreamFactory. The *-e* option is used in order to specify which SSH key is to be used:

```
10 0 * * * /usr/bin/rsync -a -e "/usr/bin/ssh -i /home/
   ubuntu/.ssh/bkup_key" backupbot@df_ip:/backup/dfbackup/ /
   backup/dfbackup/
```

7.1.2 MongoDB

The manager server will dump the MongoDB database with all the application metadata. This is stored on an external volume, the data is dumped using *mongodump* and each

collection is stored as a zipped file to take less space. To restore a dump, MongoDB has a *mongorestore* function which takes in the path to the dump and will automatically restore the database. The dump is run using the following command:

```
1 0 * * * /usr/bin/mongodump --username <MONGO_USER> --
    password <MONGO_PW> --authenticationDatabase admin --host
     <MONGO_HOST>--port 27017 --db <DB_NAME> --forceTableScan
     -o /backup/mongodb_backup/mongodb_\$(date +\%Y_\%m_\%d)
    --gzip
```

The backup is carried out once per day.

Restoring the database content

To restore the content from the dump into the database, the following command must be run

```
mongorestore --username <MONGO_USER> --password <MONGO_PW>
    --authenticationDatabase admin --host <MONGO_HOST> --port
     27017 --db <DB_NAME> <BACKUPDUMP>
```

7.2 Logging

All services, except the vulnerability retriever, run in Docker containers. Docker offers rich logging output functionality which supports a multitude of options, like to and from date, only show the latest X entries, tailing with procedural output and more. More on this can be read about on the wiki.

An example on how to output logs to a file, for the DreamFactory service would be:

```
docker-compose -f docker-compose-df.yml logs dreamfactory >>
     output.log
```

To enable debugging for DreamFactory, uncomment the environment variable APP_LOG_LEVEL in the docker-compose file for DreamFactory and set it to *debug*.

Logging in MongoDB is based on verbosity levels, these can be specified when starting the Docker service, the levels goes from 0 to 5. The level is defined in an environment variable in the docker-compose file called *MONGODB_SYSTEM_LOG_VERBOSITY*.

Vulnerability Retriever

For logging the Python script that extracts the vulnerable applications, PLED uses the stdout from the script and concatenates it to a file, the script has a logging option that can be set to print data that is useful for logging and error handling.
Below is an example of log file when the vulnerability retriever is running once per day on *update* mode.

```
ubuntu@manager:~$ cat .vuln.log
    Time: 2019-05-08 13:37:26.285440
    Scan complete, 0 applications added
    Time elapsed: 00:00:36
```

7.3 Monitoring

Monitoring is important in any system to see the status of all the services running at any given time, it can be crucial when something unexpected happens that the monitoring service alerts and shows what has caused an error. Monitoring can also be useful when implementing new features, the service can be configured to check for the new feature and see how the other systems react to it.

PLEDs monitoring has mainly been through Docker status checks like *docker ps* and *docker logs* to check the container status, in a production environment a more automated solution is recommended. This can be achieved through a wide range of already developed services. PLED has done some testing with the ELK stack service. The ELK stack is a logging service consisting of a search engine Elasticsearch, log storage Logstash, and a dashboard called Kibana. This has been run as a Docker stack, and by using Docker's log driver PLED got logs from the Apache server running the database web-interface and displayed this in Kibana. Other services such as TICK stack or Docker Vizualiser can be used, but requires more setup. These types of monitoring services are considered out of scope for the PLED project. Below is an example of how logging was enabled to stream log data to ELK in the database web-interface using the syslog driver.

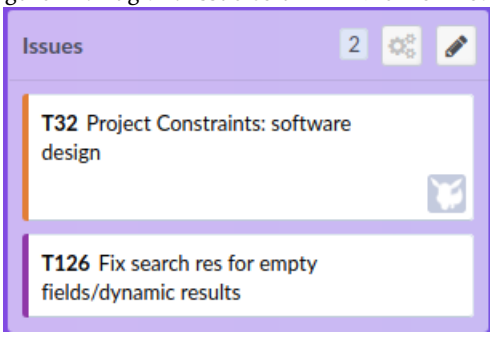
```
docker run --log-driver=syslog --log-opt syslog-address=tcp
    ://$logstaship:5000 --log-opt syslog-facility=daemon --
    name $containername -d -v $wwwfolder:/var/www/html/ -p
    $port:80 $registry/db-web-ui:latest
```

DreamFactory also supports integration with ELK stack, when running an enterprise tier [20].

7.4 Bug Tracking

Any system will have bugs in one way or another, the group needs to ensure that the bugs are properly organized and tracked. Therefore a bug tracker can be used to be able to manage and prioritize these bugs, the bug tracker is present in NCRs Phabricator, in the workboard. As we already use Phabricator for a number of other things, it was decided to utilize the workboard for tracking problems and bugs. The bugs and issues has their own section in the workboard, as to not be confused with the backlog or other columns in the workboard.

Figure 12: Bug and issue column in the workboard



7.5 Upgrading software

7.5.1 DreamFactory

It is recommended to use tagging on Docker images to prevent automatically applied updates from breaking the system. Bitnami (the provider of the currently used Docker images) provides a simple guide for how to upgrade the services provided by them. It boils down to stopping the service, snapshot it if the upgrade should fail, remove the stopped container and at last rebuild it with *docker-compose up*. It's the same procedure with all other services from Bitnami, this includes the MongoDB service running on PLEDs database server. For a more detailed explanation, see the document reference from Bitnami. Before doing an upgrade, it is wise to check if the change log indicates breaking changes. If this is the case, it is very important to ensure the snapshots/backups are in working condition, before proceeding with the upgrade.

7.5.2 Vulnerability Retriever

When upgrading the vulnerability retrieving script you are working directly on the source code, since this is an in house developed script with a limited scope. It is fairly straight forward to upgrade if the developer is familiar with the script and the Python language. the groups intention is that the script is well documented and commented so that a future developer can easily upgrade or change the functions of the script.

8 Testing

8.1 vulnRetriever

The vulnerability retriever will be tested using Python *linter* and some unit testing to check that different inputs to the scraper is treated as expected.

8.2 MongoDB search performance test

By creating dummy data with extensive JSON content, a small performance review was done. An open source and free software called mgodatagen was used in order to generate dummy data for the MongoDB server. The JSON documents was made to have similar size to the documents used in PLED environments. Most of the content was random strings of various lengths, to simulate the metadata documents in production.

The most realistic scenario ran first with a count of 100 000 documents, generated various content. The size of the collection after generating the documents was 4 kB and about 300 MB of RAM was used on the instance as a result of the process. Searches in the database from the API was not affected at this point and queries returned results within one second.

Increasing the document count to 1 000 000 impacted the instance by using about 2 GB of RAM as a result of the process, and the collection size was 28 MB. Searches done by the API was slightly impacted, but still acceptable. A query with two filter options (date and name) returned a result of 1003 documents in about 2 seconds.

At 10 000 000 documents, the instance ran out of RAM and stopped responding.

As the document database is never assumed reach a number of 1 million documents, PLED is confident that database search performance wont be a problem.

8.3 Functional testing

After deploying a clean stack, functional testing was done manually by confirming that services and functionality worked as intended. A checklist was created to help the group systematically confirm everything. This uncovered several small errors in for example scripts, PHP files and general configuration. In the bash scripts especially, special attention was needed on using absolute paths, escaping special characters, and running specific tasks as specific users. The test was executed multiple times by redeploying the stack, and it was redone with fixes implemented from the last failed test.

Below is a check-list used in our testing.

- manager-server
  - Cron jobs for backup.
  - Add SSH keys to relevant machines for credential-less login over SSH.
  - Volume attached and mounted.
- dbreplica - MongoDB
  - vulnRetriever added to cron, if enabled in Heat.
  - MongoExpress started with basic authentication, if enabled in Heat.
  - Database restored at start-up.
  - Volume attached and mounted.
  - Create *backupbot* user, with home folder and sudo permissions.
  - Password parameter passed to docker-compose file for MongoDB.
  - Enabled password authentication in ssh_config.
- web-api - DreamFactory
  - Volume attached and mounted.
  - Create *backupbot* user, with home folder and sudo permissions.
  - Enabled password authentication in ssh_config.
  - Access DreamFactory over HTTPS.
  - Create role and API-key in DreamFactory.
  - DreamFactory can access MongoDB
- Database web-interface
  - Access first time use configuration page.
  - Connect to DreamFactory and Swift for retrieving database content.
  - Authenticate with basic authentication using DreamFactory user.
  - Add, modify, remove and view database content.
  - Search for content.
- Docker Registry
  - Log in to front end with basic authentication.
  - Log in with *docker login* and pull/push images.
  - Check if connected to Swift storage.
- Balancer - HAproxy
  - Generate certificates and move them to the appropriate folder.
  - Copy and supply configuration file with correct IP-addresses.

9 Discussion

9.1 Evaluation of the result

A system ready for deployment and functioning out of the box is what PLED ended up with. All of the project results have been met, as well as most of the requirements (including criticalities below mandatory), with the additional operational functionality like for example deployment with Heat. The project owner and PLEDs supervisor has throughout the project been agreeing with most of the choices made and has been positive to the results that has been achieved.

Project effects has yet to be measured to see if the desired effects of the finished product have been fulfilled. To best accompany this fact, a handover-process will be conducted upon delivery, in order to properly prepare the employer Danny L. Murillo and NCR to take over further development and operations of PEMA/PLED.

The pre-project Gantt schema provided little guidelines for the group in the later stages, as much of the requirements were developed during the first couple of months of the project, making it somewhat obsolete. As development started, some tasks took longer than expected, vice versa, but it did give some general sense of time consumption and workload distribution at the least. Meaning it was not too far off to be completely dismissed. At the end, the group was finished with a product ready for submission, and a proper report.

All things considered, the group is pleased with the results, and is hoping for a smooth transfer to NCR for further development and usage.

9.1.1 Evaluation of Docker

Using Docker for running the applications on PLED has made dependency handling easy and the configuration through yaml files made the deployment efficient and easy. One setback with using Docker is that not every application is available in Docker and not every Docker application can be tailored to fit our needs, so some altering had to be done.

9.1.2 Evaluation of Vulnerability Retriever

The vulnerability retriever uses methods that are hard to scale and is inefficient. Considering this service as non-critical the way it was implemented satisfies the needs for PLED. The service was launched once and inserted over 3000 applications to the database. Using a collection dump from MongoDB it is not needed to run the full script again when deploying the service. For this reason, the long time to finish when running the script from scratch is accepted. In the future a more reliable way would be preferred, i.e an API from ExploitDB.

9.1.3 Evaluation of DreamFactory REST API

Taking into account that the group had no previous experience using the software, it has proved itself to be of great use for simply creating a REST API towards your storage.

It was simple to connect to and featured most necessary operations towards MongoDB, in addition to having appreciated features for user management and role-based access. There was absence of some needed functionality, and some of these were behind a pay-wall, but it was not a huge problem to either fix using custom scripting or to find ways around in the back end.

9.1.4 Evaluation of Database web-interface

As a proof of concept, the database web-interface provides a structured overview of some possibilities for uses of the DreamFactory REST API. All features mentioned in the requirements are fulfilled and implemented. As the interface is later to be implemented into PEMA, the results from this project can easily be used as a template or guide for the coordination and implementation.

9.2 Evaluation of the group work

9.2.1 Introduction

All in all, members were satisfied with what's been done and a mostly balanced work load has been achieved. The evaluation has been done through feedback within the group, discussions among group members and looking at the workboard maintained throughout the project, as seen in appendix B.

9.2.2 Organizing

The group members are situated in the same house, thus making it easy to reach each other and coordinate work. A dedicated room was established as a study room, where most of the labor was done. The group had a flat organizational structure, but members had some main responsibility areas, which made sense as different skill sets resides in each of the group members.

A schedule for when to work with the project was used at first, but it quickly became redundant as there was a general consensus between group members on how much time had to be spent on the project. It was rather used as a calendar for when group members were absent or busy with other courses, as it quickly became apparent that planning was dependent on being aware of each other's schedules.

9.2.3 Distributed workload

In the beginning, it was a known factor that some group members were more familiar with certain technologies that others were not, making some tasks more suited for select group members. Being already aware of this, it was an acceptable factor and the work load was generally distributed evenly. There was always something else to be done, e.g. filling out sections in the report, if other current tasks didn't fit a group member.

Kanban worked in PLEDs favor of making tasks visible and keeping them somewhat organized. As all members of the group lives together, the Kanban board was not used in its full potential, both because it was feature lacking [9.4.1] and much of the coordination was done orally in addition to writing on a white board in the study room. But it did come in handy for tracking progress and logging work to some degree.

9.2.4 Project as a form of study/work

While traditional education is structured and follows a set path with a lower risk of getting stuck or going down the wrong path, a project such as this is often more motivating than traditional education. It introduces elements that are relevant to both the Programming course and IT-Operations. Having the chance to help sculpt the assignment increased motivation as relevant technologies could be introduced.

A project such as this is more similar to a real-life scenario one would encounter at a future workplace, and can be used better face new challenges in a potential future workplace.

9.3 Evaluation of learning

For this project the group learned that when a problem occurs, reaching out might not always provide the project group with an answer, if any, that satisfies as an answer, thus finding alternatives and evaluating them has been an important process. For example when trying to find what storage technologies should be used, see reference [3.3].
Not everything can go as planned from the start, in the start of the development phase some features had to be altered to work with the expected functionality. An example can be the vulnerability retrieving script, where it first was intended to use an API from ExploitDB, but this API was not available and PLED had to evaluate the need for such an API and explore other alternatives
Learning how to structure the building of a service has been valuable for this project. In the beginning some of the sub services of PLED was intended to be developed before others, but while working and planning it was realized that a different approach was needed to be able to get the components working.

9.4 Evaluation of choices and technologies

9.4.1 Phabricator

As previously mentioned in [1.8.4], Phabricator has extensive functionality and an adaptation of Phabricator is currently being built by NTNU. PLED used several of the available services offered by Phabricator like the Kanban workboard, git repository, the wiki which was used for both meeting logs and describing PLED functionality and additional guidelines. It worked out well enough in this project and it did not prevent PLED from working efficiently with the assignment.

Although the workboard was somewhat lacking and at times tedious in regard to default options when creating new tasks, which had to be changed at every new task. Sub tasks was also not implemented well enough for us to properly utilize them, compared a much better solution from e.g. Jira. The markup language for the git repository in Phabricator was, again, lacking compared to other more commonly known service providers like BitBucket or GitLab, but it was more of an annoyance than a problem. It would also have been useful to have template functionality when creating meeting logs in the wiki section.

Other functionality like the chat, issue tracker or a file upload section was not used. Either because the services offered by Phabricator was lacking in terms of functionality or

was redundant compared with existing solutions already in use by PLED or the product owner.

9.4.2 Overleaf

Using Overleaf to write the report has made it easy to collaborate on the project, while some features are harder to implement, when using the ready-made template from NTNU minimal adjustments was needed.

9.4.3 Discord

Discord provided PLED with quick and easy communication with the other bachelor groups working on related projects, as well as communication with the project owner.

9.5 Evaluation of Docker Swarm

Initially it was intended to use Docker Swarm to run the DreamFactory service to accommodate for availability and redundancy by utilizing multiple nodes in the swarm. Turns out however it was problematic to get persistence storage with volumes, together with the Docker image provided by Bitnami. The default Docker volume driver caused problems regarding availability across the swarm. This is because a volume is only attached to one node, while the swarm consists of several nodes which is supposed to utilize the same volume. A another volume driver called Rexray was tried out, but also ended up causing problems regarding permissions. The services launched by docker-compose could not write to the attached and mounted volumes, even though permissions were explicitly set by chowning the folder on which the volume was mounted to. Thus when the services tried to create new folders/files, it failed. Several days in, with little progress, it was decided to use a single larger machine with Docker compose and the default volume driver.

Unfortunately the same problem was identified later on the database server nodes. MongoDB replica was difficult to implement in a Docker Swarm, again as Bitnami's Docker image was primarily meant to be run on a single server. The group could not figure out how to get around this within a reasonable time frame. The group were able to place the primary, secondary and arbiter on the separate labeled nodes (servers) named dbreplica-1, dbreplica-2 and dbreplica-3, but there was a problem when secondary and arbiter nodes tried to communicate with the primary node. It was believed that the problem was the same as the previous one, shared volume across the swarm. Rather than having to deal with this problem, a single server running with an external storage seemed like a lot less time-consuming alternative.

9.6 Future Work

9.6.1 Database web-interface

There was not sufficient time to implement the database web-interface to PEMA. The conveyed impression from PEMA was that importing a website to WordPress was not difficult, if one had some WordPress experience.

Filter

A feature that would come in handy is a filter search, filtering content based on pre-defined values. For example, without searching for a specific word in the search bar, one would be able to filter CTF-challenges of the type *attack-defense* with difficulty level *adept*. A more granulated searching functionality would lead to a better user experience as it will lead to more meaningful and substantial results when searching for files.

Metadata

To ensure best use of the Database web-interface in the future, the metadata options through the insertion and modification forms should be revisited to ensure all necessary data is stored. Both for use in all the different scenarios the vulnerability database can be of use to the NCR, but also for logging and monitoring of data modification.

Malware

The malware documents inserted as of now have limited metadata as the group was unsure of what a malware were to contain, further work on this was postponed until a more certain structure was determined. Such a structure has been added as future work, a figure showing the wished metadata can be viewed in appendix E.

9.6.2 Vulnerability Retriever

The vulnerability retriever relies on the HTML of the ExploitDB website, this is subject to change. Because of this the vulnRetriever is designed to be as adaptive as possible to the change in HTML. In the future a more reliable method of collecting applications is preferred e.g. an API from ExploitDB.
When running in update mode, the script checks with the published date, therefore when no new applications are found the published date is not updated and the next time the updater runs it will check the same ones as last time and the new ones. This causes it to take a few seconds longer to run each time. This is not a serious issue since the retriever runs in the background and does not take up much resources. If it finds an application, it will update the published date.

9.7 Assignment criticism

The assignment was originally just PEMA, but in order to be of more relevance to the group members, meaning an assignment fitting both a programming course and an IT-Operation oriented course, it was spawned as an adapted version of PEMA. While it would take a few days to come up with a new assignment, it made more sense to the group as a whole.

The project owner was very flexible and valued input from the group members when it was requested. It gave PLED a sense of purpose of producing something that would be used for what is was meant for.

What was difficult at times, was understanding PLEDs purpose in the midst of PEMA and the DSL. The different roles of each project were confusing, and the line separating the different assignments was occasionally unclear. Additionally, while waiting for the requirements from the project owner, limited work could be done, prolonging the start of the project. That said, this didn't come as a surprise since it was a branching assignment from the already existing assignment PEMA.

10 Conclusion

This project has proven to become more and more demanding as we progressed, obstacles were encountered along the way that first was not considered. Overcoming these obstacles has given valuable knowledge when it comes to thinking outside the box and develop creative solutions. Having a frequent dialog with the project owner made it possible to tweak the solution to fit their demands and the requirements became somewhat flexible.
The group created a service for storing vulnerable applications, CTF-challenges and malware where data can be added, retrieved or modified using both a database web-interface and a REST API.
The way the applications are stored makes it a very adaptable solution that can be used for a wide range of metadata and file types. The PLED service will make organizing of files and applications trivial for the NCR and help with the deployment of realistic cyber security scenarios. The product that was delivered is believed to be of use in further cyber security scenarios, as well as being improved upon in future projects by the Norwegian Cyber Range.

Bibliography

[1] NSM. Nsms årlige risikorapport 2018. https://www.nsm.stat.no/globalassets/rapporter/rapport-om-sikkerhetstilstanden/nsm_risiko_2018_web.pdf. (Visited March 2019).

[2] Git-scm. Git. https://git-scm.com/. (Visited February 2019).

[3] Phacility. Phabricator. https://www.phacility.com/. (Visited February 2019).

[4] 2011. Iso/iec 25010:2011. *ISO/IEC 25010:2011*, 4. URL: https://www.standard.no/no/Nettbutikk/produktkatalogen/Produktpresentasjon/?ProductID=474621.

[5] Mongodb limits and thresholds. URL: https://docs.mongodb.com/manual/reference/limits/.

[6]

[7] Cloud storage pricing | s3 pricing by region | amazon simple storage service. URL: https://aws.amazon.com/s3/pricing/.

[8] Dreamfactory api management. URL: https://dreamfactory.com/.

[9] Swagger. Swagger api documentation. https://swagger.io/. (Visited February 2019).

[10] DreamFactory. Dreamfactory. https://blog.dreamfactory.com/nosql-no-problem-operation-specifics/. (Visited February 2019).

[11] Api documentation - cve-search. URL: https://cve.circl.lu/api/.

[12] Class s3client. URL: https://docs.aws.amazon.com/aws-sdk-php/v3/api/class-Aws.S3.S3Client.html.

[13] MongoDB. Faq: Replication and replica sets. https://docs.mongodb.com/manual/faq/replica-sets/#what-information-do-arbiters-exchange-with-the-rest-of-the-replica-set. (Visited February 2019).

[14] DreamFactorySoftware. dreamfactorysoftware/df-docker. https://hub.docker.com/r/dreamfactorysoftware/df-docker/. (Visited April 2019).

[15] Linn, N. W. G. J. Rfc1421 - privacy enhanced mail. https://tools.ietf.org/html/rfc1421. (Visited April 2019).

[16] Toxboe, A. Progressive disclosure design pattern. http://ui-patterns.com/patterns/ProgressiveDisclosure. (Visited May 2019).

[17] Amazon. Amazon s3 pre-signed url with aws sdk for php version 3. https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/s3-presigned-url.html. (Visited May 2019).

[18] python. Python configparser. https://docs.python.org/3/library/configparser.html. (Visited May 2019).

[19] DreamFactory. Dreamfactory security guide whitepaper. https://blog.dreamfactory.com/security_whitepaper/. (Visited April 2019).

[20] DreamFactory. How to integrate elk with dreamfactory. https://blog.dreamfactory.com/configure-an-elk-stack-with-dreamfactory/. (Visited April 2019).

[21] W3. Token based authentication. https://www.w3.org/2001/sw/Europe/events/foaf-galway/papers/fp/token_based_authentication/. (Visited May 2019).

Glossary

auth-token Allow users to enter their username and password in order to obtain a token which allows them to fetch a specific resource - without using their username and password. Once their to ken has been obtained, the user can offer the token - which offers access to a specific resource for a time period - to the remote site [21]. 40

CVE-ID Common Vulnerabilities and Exposures is a list of common identifiers for publicly known cyber security vulnerabilities. 1

CVSS Common Vulnerability Scoring System is a metric for reflecting the severity of a vulnerability. 30

Document Oriented Database A nonrelational DB designed to store semistructured data as documents, often represented in JSON. 24

ethical hacking An act of trying to penetrate systems/networks with the intent of finding and reporting vulnerabilities and flaws in said systems. The goal is to improve the security of the system/network. All of this is done after signing an agreement with the owner of the system. 1

HAProxy HAProxy is a free load balancing, and proxying service for TCP and HTTP-based applications [1]. 35

heat Implements an orchestration engine to launch multiple composite cloud applications based on templates in the form of text files that can be treated like code. 37, 56

NoSQL Storage/Retrieval of data that uses other relations than tabular relations (which would be a relational database). 24

object storage Storing data as containers, referred to as objects, as opposed to single files in a hierarchy. The objects has a globally unique identifier instead of a file name and a file path, and the objects includes the data itself as well as its associated metadata. 21

openssl A toolkit for the Transport Layer Security (TLS) and Secure Sockets Layer (SSL) protocols. It is also a general-purpose cryptography library [2]. 39

Security Champion Security Champions are active members of a team that acts as the "voice" of security for the given product or team.[3]. 2

[1] http://www.haproxy.org/
[2] https://www.openssl.org/
[3] https://www.owasp.org/index.php/Security_Champions

SkyHigh NTNUs adaptation of OpenStack. It's a private cloud platform which runs on NTNU servers using OpenStack [4]. 5

SQA Means of monitoring the software engineering processes and methods used to ensure quality in the developed software. 2

unshard Sharding is a database partitioning technique that splits large databases or objects in to smaller ones. Unsharding is the joining of these shards. 27

vulnerability Vulnerability is a term that refers to a weakness or flaw in a system/application that can leave it open to an attack. 1

[4] https://www.openstack.org/software/

A Meeting logs

Meeting150119

This document was moved from /w/pled/meetingsummaries/meeting150119/jan/.

Meeting summary

- Attendees: Adrian, Askil, Karoline
- Agenda: Questions for Erik and other attendees
- Questions:

Put the assignment on paper?
What are the requirements?
What technologies should be used?(mandatory)
What bachelor Theses would you recommend for us to read?
Jungle
Previous PEMA theses/documents
Read PEMA task by Vetle and Adrian
What are we not supposed to do?
Any news on exploit.db?
Make structure for future meetings?

- Notes from meeting:

Meeting with PEMA and Mikal next week
Other thesis's to read: Configuration management system, Audun
Tools based on the certificate transparency concept
Pre-project less important
Internal and external sensor (grading)
Clear assessment of our choices
RESTAPI

Meeting220119

This document was moved from /w/pled/meetingsummaries/meeting220119/.

Meeting summary

- Attendees: Adrian, Askil, Karoline, PEMA, Basel, Erik, Danny
- Agenda: Ask quesitons
- Questions:
- Notes from meeting:

Remove the API?(Danny) - only query the db directly, OR API that would connect the registry with the repo of vuln(Basel) Regestry and vuln app as two dbs.
Metadata ontology around object will be provided by Danny and Mikael?
Owal language
Begin with the API functions and interface
For collaboration with PEMA
Basel optinion: separate PLED into a separate components
Categories, binaries, text files, know what object it is
Using labels perhaps
Send project report before Monday 28 to Erik

Meeting290119

This document was moved from /w/pled/meetingsummaries/meeting290119/.

Meeting summary

- Attendees: Adrian, Askil, Karoline
- Agenda:
- Questions:
- Notes from meeting:

Pre-prosjekt: Security champion og sikkerhet i hvert ledd
Docker container applications
TOSCA, Cloudify og Open TOSCA
Skriv i rapporten: Definer hva som er en god API, evt hva som utgjør en dårlig API.
Faglig utfylling rettet mot vårt prosjekt, ikke for generelt. Skriv kort om alternative valg av teknologier.
https://docs.docker.com/registry/spec/api/

Meeting080219

Meeting summary

- Attendees: Adrian, Askil, Karoline, PEMA
- Agenda: Discuss API
- Questions:
- Notes from meeting:

Autentisering:
PEMA should have an API key
A manual way of adding keys
Type of info available through GET: CVSS, CVE, name, platform, type/category...

Meeting120219

Meeting summary

- Attendees: Adrian, Askil, Danny, Erik, Karoline was in another meeting
- Agenda: Questions
- Questions:

How to get the vuln applications: docker hub, exploitdb, vulnhub, ctf repos
OpenStack project

- Notes from meeting:

Figure out how to store the cyber weapons,
Email basel CC danny om exploit DB.
OpenStack api is useful resource for writing API

Meeting190219

This document was moved from /w/pled/meetingsummaries/meeting190219/.

Meeting summary

- Attendees: Adrian, Askil, Karoline
- Agenda:
  - Display the requirements documents and use cases.
  - Danny and Erik can approve the requirements so that we could start the development.
  - Started implementing configurations for Dreamfactory and mongodb in openStack.
- Questions:
  - Status exploit db(Basel)
  - Integrate with LDAP(to authenticate agains teachers/students)
    - In addition to using API-keys
- Notes from meeting:

Meeting050319

This document was moved from /w/pled/meetingsummaries/meeting05032019/.

Meeting summary

- Attendees: Adrian, Askil, Karoline, Erik
- Agenda: General questions and status update.
- Questions:
- To store dockerfile or build and store image in registry ?
- ExploitDB API status
- Alternatives to exploit db for vuln retrieval
- Notes from meeting:

We have developed the database web interface where one can: upload an application with metadata, metadata can be manually filled or supply a CVE-id to autofill the form. Application is stored in Swift, metadata is stored in MongoDB.

Not sure how to retrieve applications from exploitdb and populate our database with it, automatically.

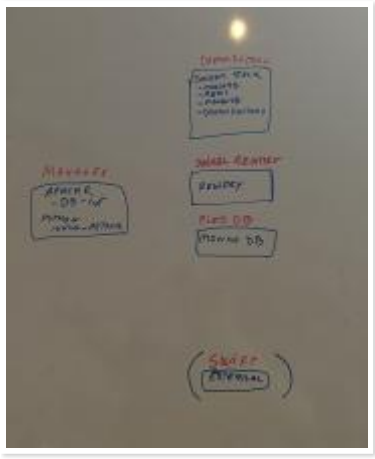
Erik asks us if we have a plan for if we can't get access to an API towards ExploitDB. Our plan is to combine several methods in order to fill our DB: do it manually (database web interface); use Docker-Hub to search for Docker Images with the docker command "docker search vaas cve-id"; try out a search engine for exploitDB MDEOUS/exploitdb (git).

Meeting120319

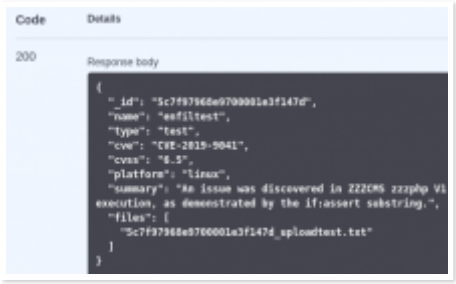
Meeting summary

- Attendees: Adrian, Askil, Karoline
- Agenda:
- Questions:

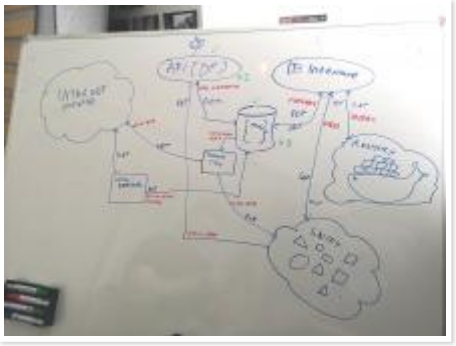
Vulnerability retriever: is in alpha, working for testing purposes. Is written in PHP but want to rewrite to python since it doesnt need to be in php and python has less dependencies.
Scrapes exploitdb for vulnerable applications, uses quite a long time so not to get blocked, thoughts on this approach ?
Server layout: comments ?



PEMA: How does PEMA handle login, should it use the same database for login as PLED? How does PEMA want the application/files delivered?



Erik: In regards to the MongoDB containing documents with metadata, should this be placed in a Docker Swarm with 3 servers for availability? Or is it enough with just one DB server, as there is few writes per day, just one large initial write?
When we want to load balance the API (DreamFactory), do this in a separate Docker Swarm cluster?



- Notes from meeting:

Meeting190319

Had meeting the previous day about PEMA, PLED and the DSL project.

- Test out API, try to combine PLED and PEMA to see how the API is used
- Scarping works for now

Meeting with Erik

- Try to develop the scraper with future change in mind
  - Make change as easy as possible
- First draft of bachelor report due 11.04.19
- Try to deploy a vulnerable app to see the steps necessary
- Talk with DSL developers about metadata needed

Meeting250319

Meeting summary

- Attendees: Adrian, Askil, Karoline, Rune
- Agenda: Database
- Questions:

Situasjonen vår - cve dokument med embedded document som inneholder applikasjoner.

Er det et problem at embedded documents kan gro uhemmet? Skal vi derfor bruke document refrences i stedet?

Vurdere bruk av relational fremfor document oriented database?

- Notes from meeting:

BASE > ACID
Flexible Data Model
Consistency
Fokuser på hvordan vi kom frem til valget (ta til betraktning det vi vet i dag)
Beskrive bruken av databasen (brukere, størrelse, bruksmønster), ytelse er i mindre grad interessant i vårt bruksområde(?)
Vurdere simulering av database-brukt, python script fyller db med 100k docs o.l. og monitorer ytelse; oppfører parametere seg annerledes ved feks 500 documenter og 500k documenter.

Meeting260319

Meeting summary

- Attendees: Adrian, Askil, Karoline, (Lars Erik), Mihkal
- Agenda:
  - API key Swift
  - Metadata for vulnerabilities
- Questions:
- Notes from meeting:
  - Ansible role for vulnerable applications (ikke dockerfiles). - github.com/geerlingguy/ansible-role-ntp
    - Installer ansible - fort gjort å lage skjellet til rolle, Ansible Role NTP
      - Tasks mappe og default
  - Vi legger til at det kan legges til roller i vårt interface, zip file eller github repo, (med dependencies)
  - Docker registry
    - Mihkal kan teste deployment fra de

Meeting290319

Meeting summary

- Attendees: Adrian, Askil, Karoline, PEMA, Michael, Danny and Basel
- Agenda: UML Drawing session, general discussion
- Questions:
- Notes from meeting:

Include malware-collection.

5. april - Geir Olav supplies Basel with info for NCR meeting
6. april - Geir Olav shows up on NCR meeting

Sequence diagram (dynamic) a general overview, UML focus on one component.

Meeting020319

Meeting summary

- Attendees: Adrian, Askil, Karoline, Erik
- Agenda: Weekly progress meeting
- Questions:

What we have done so far:
Renewed schema for easier searching/filtering, and other API operations on our database.
Reviewed report chapters, like requirements/design/introduction.
Created more models: high level use case diagram of PLED and more specific sequence diagrams to better explain our components.
Reviewed Danny's notes on our report and changed it accordingly.
Updated Wiki for API usage.
Had a meeting with Rune to get better argumentation for our choice of database.
Meeting with PEMA and Mikael to coordinate high level information, expectations and so on.
Talked with Lars Erik about usage of credentials when using the API towards OpenStack Swift.

What we want to ask:
Security chapter related questions...

- Notes from meeting:

Meeting090419

Meeting summary

- Attendees: Adrian, Askil, Karoline, Erik, Danny(?)
- Agenda: General questions, progress
- Questions:

What should be in deployment and what should be in implementation?
For example, DreamFactory is currently under implementation, but it also describes how its deployed in PLED, at the same time.
Same goes with MongoDB, we use someone else's Docker Image.

- Notes from meeting:

Need to read report, will get from report feedback

Meeting300419

Meeting summary

- Attendees: Adrian, Askil, Karoline, Erik
- Agenda: General question, progress
- Questions:

Heat Template, how to handle SSH keys on new instances when pulling from Phabricator:

- Move project to GitHub public repo?
- Use same private key across all instances? 😟
- Some ninja method we dont know about? 😏

To what degree must the project be complete when we deliver the project, in regards to finishing touches?

Worked on the Heat template (creates instances, network, installs docker/docker-compose, adds cronjobs, creates/attaches/mounts volumes).
Reviewed report regarding the feedback from Erik, e.g. added an Operations chapter to the report. It contains the subjects backup, logging, monitoring, upgrading and bug tracking.
GUI is coming along.
Retrieving files is now possible, custom CURL in DreamFactory.
We are now using a service user in OpenStack, instead of our own accounts, for communication with Swift.

- Notes from meeting:

Arugmenter for valgene tatt i utvikling av GUI, mtp fargevalg, biblioteker, verktøy.
Malware

Meeting070519

Meeting summary

- Attendees: Adrian, Askil, Karoline, Erik
- Agenda: Questions about Heat, Puppet, Consul, progress
- Questions:
  - Enable communication between machines using ssh_authorized _keys, manager is booted before the workers.
  - Eks: Backup needs to communicate over SSH with web-api server for pulling the files for backup
  - Use puppets authorized key resouce for this ?
  - If puppet we need hostnames, best way to do this ?
  - Use consul for service discovery / DNS ?
- Notes from meeting:

Gå over intro igjen, må være god. A scrutiny of an abstract?
Generere brukere på maskinene, og kjøre ssh-copy-id uname@host med brukernavn/passord.
Volum images som backup?
Noe som ikke skal driftes kontinuerlig over lang tid. Burde være tilpasset å kunne tas opp og ned "ofte", fremfor at feks puppet skal vedlikeholde systemet.
Vurdere å lage backup server på utsiden av stack, evt volumene på utsiden av stacken?
Puppet, alle andre enn master må ha master i etc/host.
Prøv

B Kanban Work Cards

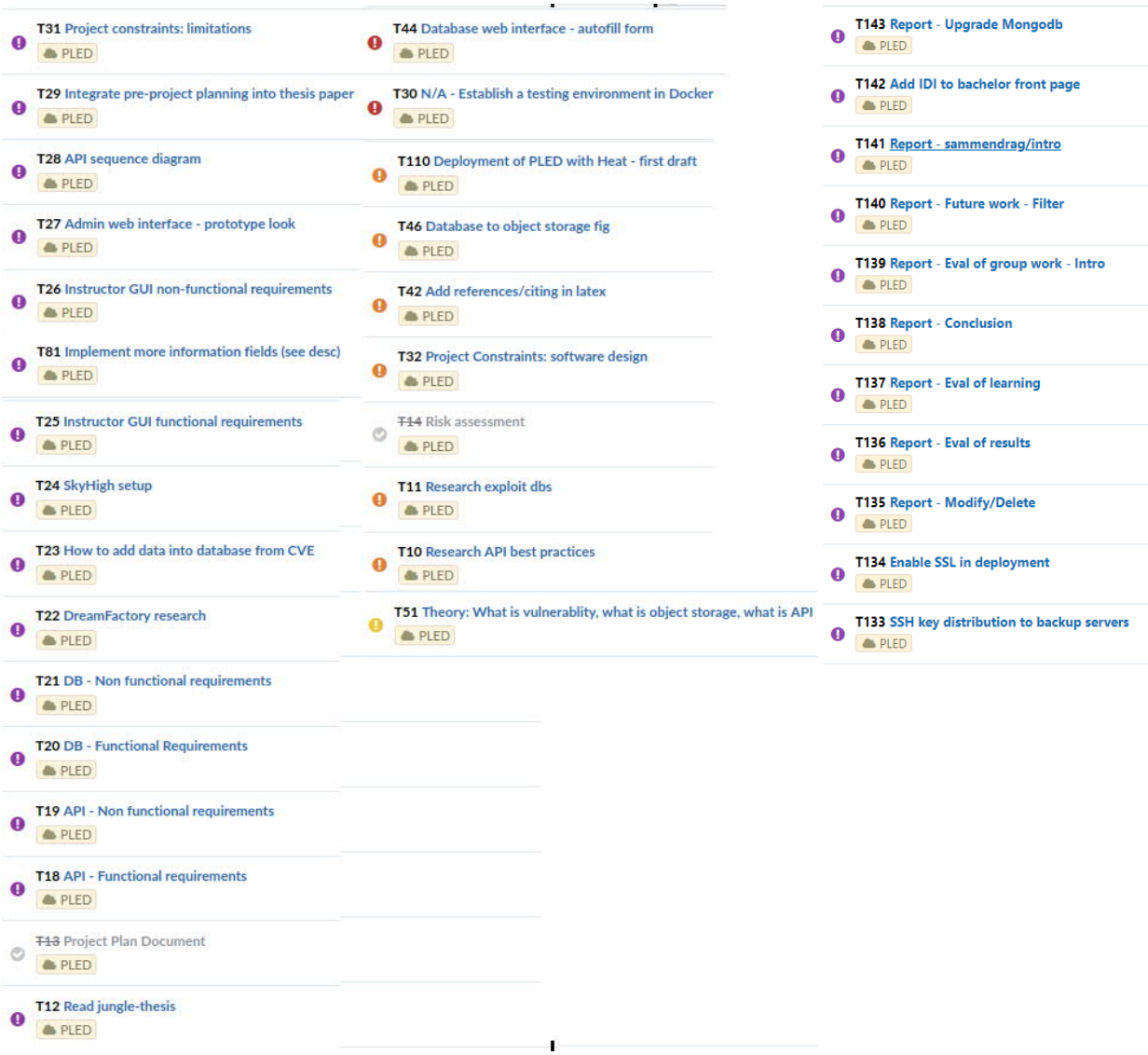
Unbreak Now! (2)

- **T37** Short definition on whats a good/bad API, in regard to our project — PLED
- **T17** DB use cases — PLED

Needs Triage (98)

- **T132** Delete objects in swift when object is deleted in mongodb — PLED
- **T131** Error handling in db web-interface — PLED
- **T130** Ensure db web-interface is described as proof of concept or something, you know what I mean — PLED
- **T129** gui change category to type — PLED
- **T128** Look into duplication in retrieving data to be viewed in frontend using twig — PLED
- **T127** Create view for deleting db content in front end — PLED
- **T126** Fix search res for empty fields/dynamic results — PLED
- **T125** Create modify view in front end — PLED
- **T124** Add/change cve information stored from front end — PLED
- **T123** Erik Feedback - FOOTNOOTS — PLED
- **T122** 3.3 Too dependent on OpenStack when we use Swift? (desc) — PLED
- **T121** Erik feedback - 3.3.1 "PLED has decided..." needs more reasoning — PLED
- **T120** Erik feedback - section or chapter on IT-operations (desc) — PLED
- **T119** Erik feedback - evaluate whats written about lets encrypt (see desc) — PLED
- **T118** Erik feedback - vectorize our figures/schemas (export as PDF) — PLED
- **T117** Erik feedback - Eval requirement criticality on add/upd/del content — PLED
- **T116** Erik feedback - Bullet list on what needs to be done - ch 1.2 — PLED
- **T115** Erik feedback - Diqs how much Phabricator was used — PLED
- **T114** Security - DreamFactory — PLED
- **T113** Arguments for NOT choosing other development frameworks — PLED
- **T112** Change how we explain why we use incremental development — PLED
- **T111** Introduction - Development framework - Include assignment uncertainty — PLED
- **T109** Implementation - Docker Swarm — PLED
- **T108** Enable SSL/TLS for DreamFactory — PLED
- **T107** Implementation - HAProxy loadbalancing — PLED
- **T106** Update db schema based on Dannys ER diagrams from Discord — PLED
- ~~**T105** Explain GridFS~~ — PLED
- **T104** Database model see comment — PLED
- **T103** Replace vulnerable applications with a more general term better describing mangfoldet — PLED
- **T102** Link tables — PLED
- **T101** Label tables — PLED
- **T100** Update CTF form in db GUI — PLED
- **T99** Add chapter on top right of pages in latex — PLED
- **T98** Update vulnRetriever with new database format — PLED
- **T97** Update Wiki with API usage — PLED
- **T96** Update Database use cases in requirements — PLED
- **T95** Update glossary - Internal termonology — PLED
- **T94** Link and label requirements — PLED
- **T93** Database Design - Rework argumentation — PLED
- **T92** Diagram: vulnRetriever sekvensdiagram — PLED
- **T91** Diagram: API sekvens diagram — PLED
- **T90** Diagram: PLED happy path — PLED
- **T89** Diagram: hele system sekvens diagram — PLED
- **T88** Diagram: database modellering — PLED
- **T87** Diagram: use case hele systemet — PLED

T86 Database Design - Include BASE PLED	~~T69~~ Design - Event-Driven VS. Request Response PLED	T53 Implementation: Getting vulnerabilities PLED
T85 Requirements - subT82 PLED	T68 Requirements - Explain why we use ISO25010 PLED	T52 Database web interface - search page PLED
T84 Write how the report is organized PLED	T67 Requirements - define low/med/high criticality PLED	T50 Database web interface - upload file PLED
~~T83~~ Fix the requirements | Duplicate PLED	T66 Fix ref and glossary mixup PLED	T49 Database web interface - submit form to mongoDB PLED
T82 Review and change report in regards to Danny's comments in the actuall report PLED	T65 Introduction: Terminology PLED	T48 Mer om decoupling og adskilt system i :system architecutre, introduction PLED
T81 Implement more information fields (see desc) PLED	T64 Introduction: Project group background PLED	T47 Introduction - Scope PLED
~~T80~~ More argumentation for choice of DB-model PLED	T63 Working tools and methods: testing PLED	T45 Design - Admin web interface features PLED
T79 Oppsett av lastbalanserer PLED	T62 Working tools and methods: Version control PLED	T43 Deploy local CVE search database PLED
~~T77~~ Basis oppsett av Docker Swarm - Database PLED	T61 Working tools and methods: programming language PLED	T41 Admin = DreamFactory ? Endre Admin GUI til Database management PLED
T76 Basisoppsett av Docker Swarm - DF PLED	T60 Working tools and methods: Docker PLED	T40 Discuss storage mechanisms PLED
T75 Fullfør dokument-utforming til database PLED	T59 Storing data: docker images / other files PLED	T39 Does design satisfy requirements PLED
~~T74~~ REMOVED PLED	~~T58~~ Database web interface - other files PLED	T38 Cyber Weapon storage analysis PLED
T73 Vuln retriever PLED	T57 Database mgmt gui PLED	T36 System Architecture: Admin GUI PLED
T72 LDAP testing PLED	T56 Implementation: Admin interface PLED	T35 System Architecture: Introduction PLED
~~T71~~ LDAP configuration/setup PLED	T55 Implementation: API development PLED	T34 PEMA desc description PLED
~~T70~~ LDAP integration - DB mgmt interface PLED	T54 Impmementation: Storing vulnerabilties PLED	T33 Project Constraints: Subject Area PLED


T31 Project constraints: limitations
PLED
T29 Integrate pre-project planning into thesis paper
PLED
T28 API sequence diagram
PLED
T27 Admin web interface - prototype look
PLED
T26 Instructor GUI non-functional requirements
PLED
T81 Implement more information fields (see desc)
PLED
T25 Instructor GUI functional requirements
PLED
T24 SkyHigh setup
PLED
T23 How to add data into database from CVE
PLED
T22 DreamFactory research
PLED
T21 DB - Non functional requirements
PLED
T20 DB - Functional Requirements
PLED
T19 API - Non functional requirements
PLED
T18 API - Functional requirements
PLED
T13 Project Plan Document
PLED
T12 Read jungle-thesis
PLED
T44 Database web interface - autofill form
PLED
T30 N/A - Establish a testing environment in Docker
PLED
T110 Deployment of PLED with Heat - first draft
PLED
T46 Database to object storage fig
PLED
T42 Add references/citing in latex
PLED
T32 Project Constraints: software design
PLED
T14 Risk assessment
PLED
T11 Research exploit dbs
PLED
T10 Research API best practices
PLED
T51 Theory: What is vulnerablity, what is object storage, what is API
PLED
T143 Report - Upgrade Mongodb
PLED
T142 Add IDI to bachelor front page
PLED
T141 Report - sammendrag/intro
PLED
T140 Report - Future work - Filter
PLED
T139 Report - Eval of group work - Intro
PLED
T138 Report - Conclusion
PLED
T137 Report - Eval of learning
PLED
T136 Report - Eval of results
PLED
T135 Report - Modify/Delete
PLED
T134 Enable SSL in deployment
PLED
T133 SSH key distribution to backup servers
PLED

C Pre-project report

NTNU
Norwegian University of
Science and Technology

Pre-project

Bachelor assignment 2019

PLED

February 11, 2019

Contents

1 Project information

Title:	PLED (Pentesting Lab Environment Deployment)
Projectnr:	47
Participants:	Adrian Jacobsen Moen (BITSEC), Askil Amundøy Olsen (BITSEC), Karoline Moe Arnesen (BPROG)
Project owner:	NTNU, Norwegian Cyber Range w/Danny Lopez Murillo
Coordinator:	Erik Hjelmås
Contact person:	Adrian Moen

2 Goals and limits

2.1 Background

The PLED(*Pentesting Lab Environment Deployment*) project is a subproject of the PEMA (Pentesting Exercise Management Application). PEMA is a modular scalable virtualization platform, its purpose is to deploy virtual cyber scenarios for use in ethical hacking, penetration testing and cyber security competitions. PEMA is meant to give instructors and students an interface to deploy, log and submit cyber security exercises.
PLEDs purpose is to create a platform that collects and stores vulnerable applications and systems, that can be deployed by the instructor at will. PLED will provide a graphical interface where an instructor can select vulnerable systems to be deployed, as well as a RESTapi that will be used by the PEMA project. Additionally the instructor should be able to give the vulnerable machines a token or flag that is ment to be submitted when the user has completed the exercise.
Since the PEMA project is developed along side the PLED project it will be difficult to integrate properly, for that reason the PLED project will additionally create a service to deploy machines and configure a lab environment using Openstack and Puppet, independently from the PEMA project. In the future the plan is to integrate these projects to one project.
The collected systems and applications should be stored in a database or fileserver along with all metadata for each vulnerability e.g. CVE and CVSS score.

2.2 Project goals

The goals of this assignment is to create a solution for the Norwegian Cyber Range to store vulnerable applications and a method to receive and return vulnerable applications through a API to be used in the PEMA project.
The main goals to achieve are the following:

- Create a vulnerability database
  Translate the given vulnerability description ontology into a database schema, design and deploy it.
- Categorize the vulnerable applications
  Capture The flag, Penetration Testing, Attack and Defence etc.
- Provide a graphical interface
  Allows instructor to manually add vulnerable applications with its corresponding metadata.
- API interface to the database.
  (From graphical interface)Add vulnerable applications.
  Query database for available vulnerable applications with/without information.
  Return specific vulnerable applications meeting given criteria.
- Deploy the vulnerable application using the API interface.

2.3 Limitations

The final result of this assignment is to be used for educational purposes only and is meant to be a tool to be used in the course Ethical hacking. It's meant to educate students in fields of ethical hacking/white hat hacking, pen-testing and academic writing. For use by the Norwegian Cyber Range primarily.

3 Scope

3.1 Subject area

This project is a part of the Norwegian Cyber Range project from NTNU. The goal of this project is to educate within the subjects of cybersecurity both offensive and defensive.

3.2 Problem scope

PLED is only a part of PEMA, and will be developed independently. There will be some coordination between PLED and PEMA, in order to avoid overlapping, but also to work towards the same goal of completing a fully working PEMA.

3.3 Problem description

The PEMA project needs a platform for storing vulnerable systems and applications as well as deploying them in a safe environment. PLED should be able to collect vulnerable systems from sites such as *exploit-db* and *seebug.org*, and store the needed metadata to a corresponding database.

4 Project organization

4.1 Assigned responsibilities and roles

Our group (PLED) consists of three bachelor students, two from IT-Operations and Information Security and one from Programming - Applications. Danny Lopez Murillo from NTNU Norwegian Cyber Range is the project owner, while Erik Hjelmås is our coordinator.

- Adrian Jacobsen Moen (BITSEC) - Developer, Communication and secretary, Security Champion.
- Askil Amundøy Olsen (BITSEC) - Developer, Security Champion.
- Karoline Moe Arnesen (BPROG) - Main API/backend developer, Database engineer.

4.2 Internal policies and routines

Routines: Each Tuesday PLED will meet with our coordinator for evaluation and general discussion about the progress on our work. A meeting summary is written each meeting, using the template in OneNote. All group members are to write up how many hours spent working on the project each day.

5 Planning, follow-up and report

5.1 Project modules

The project is divided into the following modules:

- Defining the API
- Design and deploy database(s)
- Download and insert vulnerable applications into database
- Create Database API
- Create graphical interface
- Deployment of vulnerable applications

5.1.1 Software development framework/process

We have chosen the iterative and incremental method in agile development, PLED needs to be able to adapt to requirement changes down the line, plus some requirements are not yet established. Iterative delivery was chosen in order for our coordinator and project owner to efficiently assess the work that has been done since our previous meeting. This allows for early feedback on our components. Along with our chosen process, PLED will also use Kanban board to distribute workload among the group and use it to keep track of pending tasks and prioritize accordingly. Kanban also provides us with a visualization of our progress from start to finish. The members of PLED have used agile and Kanban in previous projects, making it a preferred method.

5.1.2 Methods and approach

The method used will be to develop and document continuously, where everything that is done is roughly documented, so that it can be further improved when completed. The development will be done incrementally so that one part that can be completed before going to the next component. This will only work if every part of the development is carefully planned on beforehand. The actual development and testing should be trivial when basing it upon the plans already made. This also counts for the documentation as it should be done alongside in each process.

5.2 Planning of project status meetings

The weekly status meetings will be used to assess work done, whats planned for the rest of the week(s) and general questions from PLED.

6 Quality Assurance organization

6.1 Documentation, standards and source code

The system will be documented using the wiki on Gitlab, where a repository will also be created to store and version the source code. PLED will use standard documentation methods where each code-module will be visualized and explained in the wiki by a standardized format. All source code will be documented using its appropriate Programming Language Documentation.

6.2 Configuration management

The project will be version controlled with the use of Git. The report will be written in a shared LaTex document on Overleaf. This allows for reverting to older versions and see changes made in real time. In Git there will be a testing branch and production branch, this way all the development and testing will be done in testing and when all tests are passed it can be merged into production.
PLEDs goal is to integrate all software through a CI/CD pipeline to make development as consistent and repeatable as possible, this also helps PLED automate the process as much as possible.
The goal for change in the project is that it is seamless and efficient.

6.3 Risk assessment

6.3.1 Technology

Risk	Analysis	Treatment
Not being able to extract data from vulnerable repositories	This means PLED will not be able to deploy the solution to its full extent, as there wont be any applications to deploy from the database.	Implement the surrounding technology, excluding the database content (vulnerable applications), could populate database with dummy-applications.
Not be able to integrate with PEMA	The solution may work separately but if the application cannot be integrated with PEMA it will be difficult to use it for what was originally intended.	Clearly defined interfaces on the API that both groups have agreed upon
Improper database design	If the database is inefficient or the badly designed, it will cause problems when using the API	To save future hassle and have more efficient API calls, the design of the database will be thoroughly considered before implementing

6.3.2 Business

Risk	Analysis	Treatment
Downloading vulnerable applications to NTNU servers	If the vulnerable applications are not quarantined in a secure environment, the vulnerabilities might cause issues for the outside network.	Hardening, make sure incoming connections are only from trusted sources. When the systems is live, consider if network connectivity is needed
Application delivered not as requested	If the projects requirements is not clearly established and communication with project owner is scarce, then the delivery may not be as expected from the clients perspective	Clearly defined goals and requirements, open communication with project owner and coordinator. Continuous review of requirements
Application not finished in time	If the time is not managed properly or the project becomes larger than planned, the application might not be complete when the deadline is due	Follow GANTT schedule, clearly define the work to be done, define scope

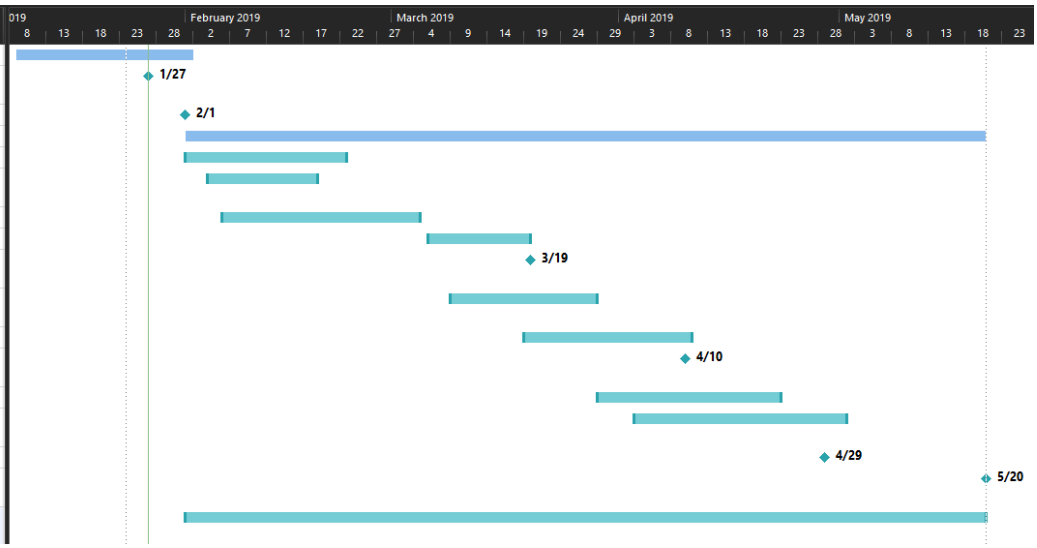
6.3.3 Project group

Risk	Analysis	Treatment
Not being able to keep up with the set time frame because of unforeseen problems with the development.	Lagging behind on work and not being able to deliver components in time creates more problems down the line because of work-buildup.	Reach out for help when problems occur, pivot if problems can't be solved.
Illness in PLED, leaving one or more group member(s) unable to work for an extended period of time (1 week or more).	If a member of the group is unable to work, progress will be slowed down. The situation gets worse if the group member is working on technology that the rest of the group is less familiar with.	Prioritize tasks, or evaluate the situation and consider leaving out functionality.
Uncoordinated work	Members of the group work separately on the same modules with out knowing. This results in duplicated work and lost efficiency.	Use Kanban board cards to delegate tasks between the members, and keep to given tasks. Check Git to see what other members are currently working on.

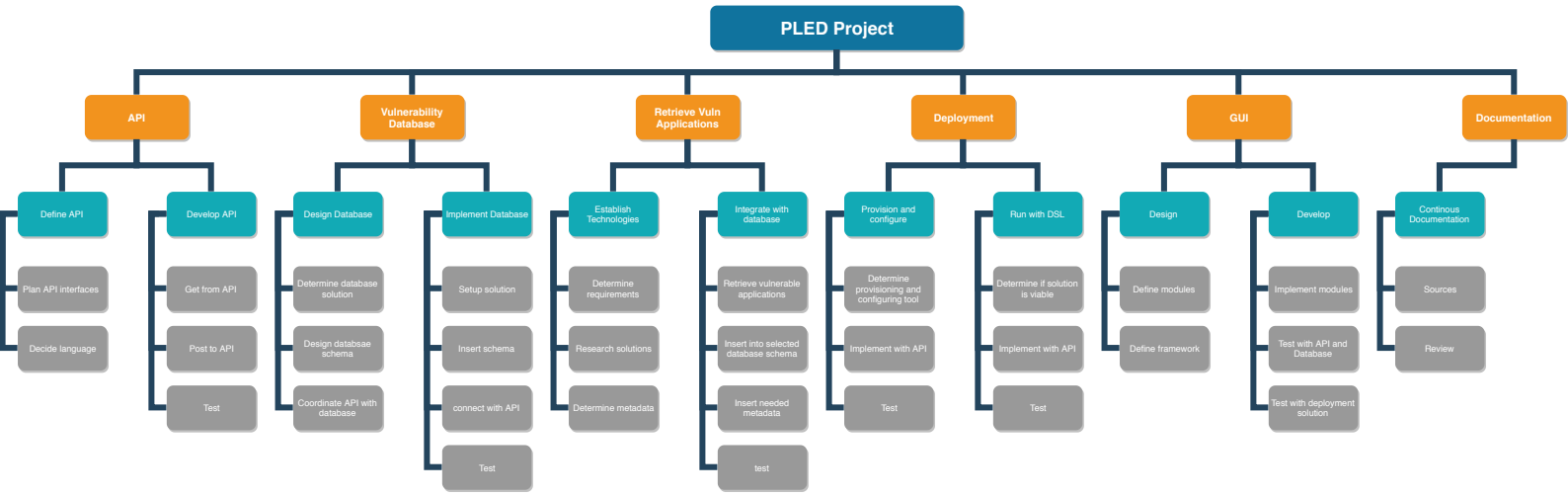
7 Progress management

7.1 GANTT

Task Name	Duration	Start	Finish
Pre project	18 days	1/9/19	2/1/19
Deliver pre project to Erik Hjelmås	0 days	1/27/19	1/27/19
Deadline pre project	0 days	2/1/19	2/1/19
Bachelor PLED Thesis	77 days	2/1/19	5/20/19
Define API	16 days	2/1/19	2/22/19
Research online vulnerability repositories	11 days	2/4/19	2/18/19
Design database	19 days	2/6/19	3/4/19
Create database	10 days	3/6/19	3/19/19
Completed database development	0 days	3/19/19	3/19/19
Retrieve vulnerabilities online	15 days	3/9/19	3/28/19
Create API	17 days	3/19/19	4/10/19
Completed API development	0 days	4/10/19	4/10/19
Deploy solution for testing	17 days	3/29/19	4/22/19
Create a GUI - for DB administration	21 days	4/3/19	5/1/19
First draft of report	0 days	4/29/19	4/29/19
Bachelor Thesis hand-in deadline	0 days	5/20/19	5/20/19
Rapport building and documentation	77 days	2/1/19	5/20/19



7.2 Work Breakdown Structure


PLED Project
API
Define API
Plan API interfaces
Decide language
Develop API
Get from API
Post to API
Test
Vulnerability Database
Design Database
Determine database solution
Design databsae schema
Coordinate API with database
Implement Database
Setup solution
Insert schema
connect with API
Test
Retrieve Vuln Applications
Establish Technologies
Determine requirements
Research solutions
Determine metadata
Integrate with database
Retrieve vulnerable applications
Insert into selected database schema
Insert needed metadata
test
Deployment
Provision and configure
Determine provisioning and configuring tool
Implement with API
Test
Run with DSL
Determine if solution is viable
Implement with API
Test
GUI
Design
Define modules
Define framework
Develop
Implement modules
Test with API and Database
Test with deployment solution
Documentation
Continous Documentation
Sources
Review

7.3 Milestones

Following is our defined milestones, these may change, when the project is defined further.

- Define API interfaces
- Complete database
- Retrieve vulnerable systems successfully
- Complete GUI
- Deploy in test environment
- Integrate with PEMA

7.4 Time and resource scheduling

Our time scheduling is illustrated in the gantt diagram. The members of the PLED project is using a shared calendar, where all other tasks are scheduled, this is to know when to set off time for project work. For task and resource distribution Trello is used along with a Kanban template to help manage tasks to be done.

D Group Agreement

NTNU

Norges teknisk-naturvitenskapelige universitet

Vår dato Vår referanse

Prosjektavtale

mellom NTNU Fakultet for informasjonsteknologi og elektroteknikk (IE) på Gjøvik (utdanningsinstitusjon), og

NTNU, IIK, NORWEGIAN CYBER RANGE/
DANNY LOPEZ MURILLO (oppdragsgiver), og

ADRIAN JACOBSEN MOEN
ASKIL AMUNDØY OLSEN
KAROLINE MOE ARNESEN (student(er))

Avtalen angir avtalepartenes plikter vedrørende gjennomføring av prosjektet og rettigheter til anvendelse av de resultater som prosjektet frembringer:

1. Studenten(e) skal gjennomføre prosjektet i perioden fra JAN 2019 til 20 MAI 2019.

   Studentene skal i denne perioden følge en oppsatt fremdriftsplan der NTNU IE på Gjøvik yter veiledning. Oppdragsgiver yter avtalt prosjektbistand til fastsatte tider. Oppdragsgiver stiller til rådighet kunnskap og materiale som er nødvendig for å få gjennomført prosjektet. Det forutsettes at de gitte problemstillinger det arbeides med er aktuelle og på et nivå tilpasset studentenes faglige kunnskaper. Oppdragsgiver plikter på forespørsel fra NTNU å gi en vurdering av prosjektet vederlagsfritt.

2. Kostnadene ved gjennomføringen av prosjektet dekkes på følgende måte:
   - Oppdragsgiver dekker selv gjennomføring av prosjektet når det gjelder f.eks. materiell, telefon/fax, reiser og nødvendig overnatting på steder langt fra NTNU på Gjøvik. Studentene dekker utgifter for ferdigstillelse av prosjektmateriell.
   - Eiendomsretten til eventuell prototyp tilfaller den som har betalt komponenter og materiell mv. som er brukt til prototypen. Dersom det er nødvendig med større og/eller spesielle investeringer for å få gjennomført prosjektet, må det gjøres en egen avtale mellom partene om eventuell kostnadsfordeling og eiendomsrett.

3. NTNU IE på Gjøvik står ikke som garantist for at det oppdragsgiver har bestilt fungerer etter hensikten, ei heller at prosjektet blir fullført. Prosjektet må anses som en eksamensrelatert oppgave som blir bedømt av intern og ekstern sensor. Likevel er det en forpliktelse for utøverne av prosjektet å fullføre dette til avtalte spesifikasjoner, funksjonsnivå og tider.

4. Alle bacheloroppgaver som ikke er klausulert og hvor forfatteren(e) har gitt sitt samtykke til publisering, kan gjøres tilgjengelig via NTNUs institusjonelle arkiv hvis de har skriftlig karakter A, B eller C.

Tilgjengeliggjøring i det åpne arkivet forutsetter avtale om delvis overdragelse av opphavsrett, se «avtale om publisering» (jfr Lov om opphavsrett). Oppdragsgiver og veileder godtar slik offentliggjøring når de signerer denne prosjektavtalen, og må evt. gi skriftlig melding til studenter og instituttleder/fagenhetsleder om de i løpet av prosjektet endrer syn på slik offentliggjøring.

Den totale besvarelsen med tegninger, modeller og apparatur så vel som programlisting, kildekode mv. som inngår som del av eller vedlegg til besvarelsen, kan vederlagsfritt benyttes til undervisnings- og forskningsformål. Besvarelsen, eller vedlegg til den, må ikke nyttes av NTNU til andre formål, og ikke overlates til utenforstående uten etter avtale med de øvrige parter i denne avtalen. Dette gjelder også firmaer hvor ansatte ved NTNU og/eller studenter har interesser.

5. Besvarelsens spesifikasjoner og resultat kan anvendes i oppdragsgivers egen virksomhet. Gjør studenten(e) i sin besvarelse, eller under arbeidet med den, en patentbar oppfinnelse, gjelder i forholdet mellom oppdragsgiver og student(er) bestemmelsene i Lov om retten til oppfinnelser av 17. april 1970, §§ 4-10.

6. Ut over den offentliggjøring som er nevnt i punkt 4 har studenten(e) ikke rett til å publisere sin besvarelse, det være seg helt eller delvis eller som del i annet arbeide, uten samtykke fra oppdragsgiver. Tilsvarende samtykke må foreligge i forholdet mellom student(er) og faglærer/veileder for det materialet som faglærer/veileder stiller til disposisjon.

7. Studenten(e) leverer oppgavebesvarelsen med vedlegg (pdf) i NTNUs elektroniske eksamenssystem. I tillegg leveres ett eksemplar til oppdragsgiver.

8. Denne avtalen utferdiges med ett eksemplar til hver av partene. På vegne av NTNU, IE er det instituttleder/faggruppeleder som godkjenner avtalen.

9. I det enkelte tilfelle kan det inngås egen avtale mellom oppdragsgiver, student(er) og NTNU som regulerer nærmere forhold vedrørende bl.a. eiendomsrett, videre bruk, konfidensialitet, kostnadsdekning og økonomisk utnyttelse av resultatene. Dersom oppdragsgiver og student(er) ønsker en videre eller ny avtale med oppdragsgiver, skjer dette uten NTNU som partner.

10. Når NTNU også opptrer som oppdragsgiver, trer NTNU inn i kontrakten både som utdanningsinstitusjon og som oppdragsgiver.

11. Eventuell uenighet vedrørende forståelse av denne avtale løses ved forhandlinger avtalepartene imellom. Dersom det ikke oppnås enighet, er partene enige om at tvisten løses av voldgift, etter bestemmelsene i tvistemålsloven av 13.8.1915 nr. 6, kapittel 32.

12. Deltakende personer ved prosjektgjennomføringen:

NTNUs veileder (navn): ERIK HJELMÅS

Oppdragsgivers kontaktperson (navn): DANNY LOPEZ MURILLO

Student(er) (signatur): _____ dato 11/01-19

_____ dato 11.01.19

Karoline Moe Arnesen dato 11/01/19

_____ dato _____

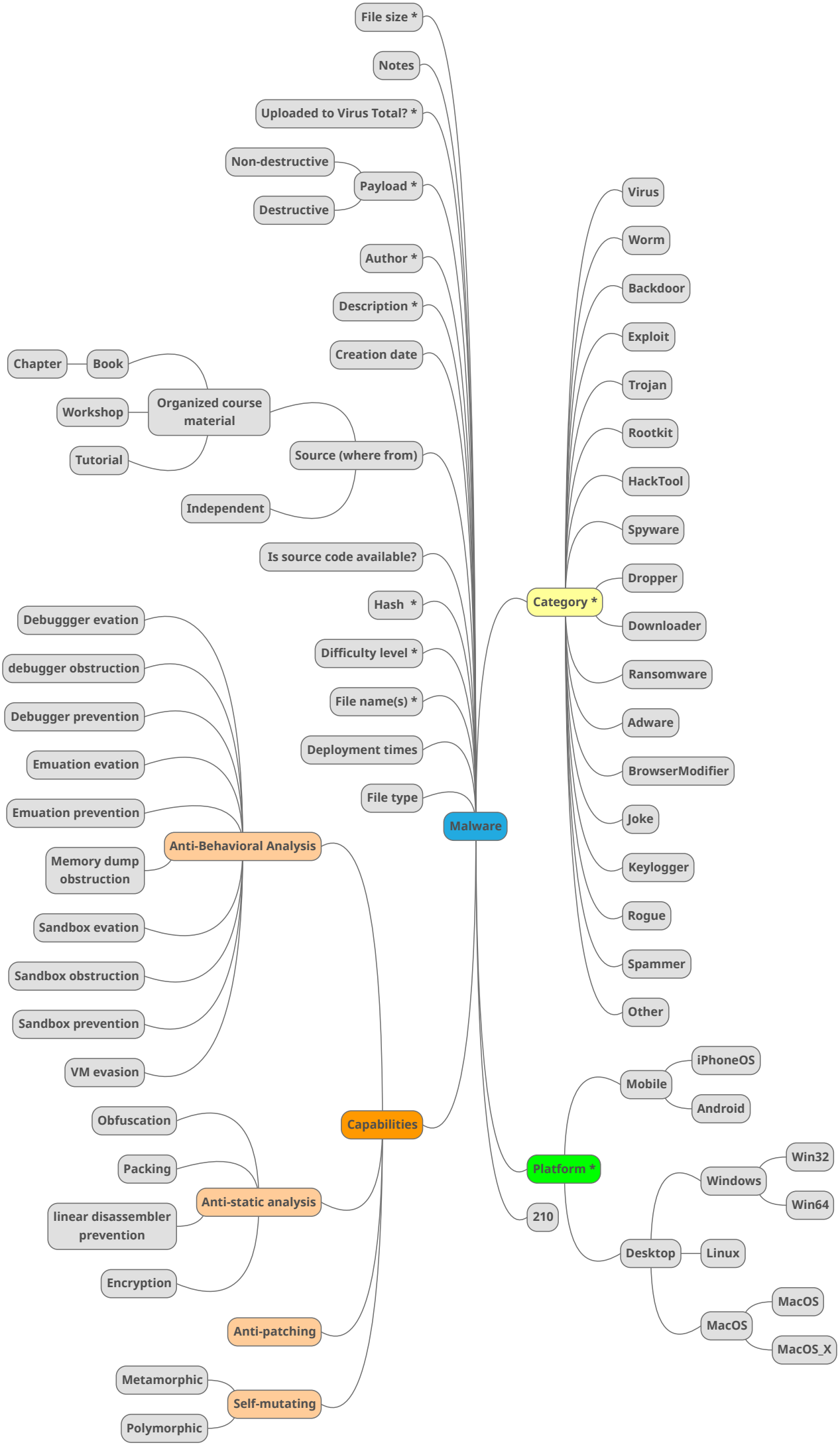
Oppdragsgiver (signatur): Danny López M. dato 15-1-2019

Signert avtale leveres digitalt i Blackboard, rom for bacheloroppgaven.
Godkjennes digitalt av instituttleder/faggruppeleder.

Om papirversjon med signatur er ønskelig, må papirversjon leveres til instituttet i tillegg.
Plass for evt sign:

Instituttleder/faggruppeleder (signatur): _____ dato _____

E Malware meta model


Malware
File size *
Notes
Uploaded to Virus Total? *
Payload *
Non-destructive
Destructive
Author *
Description *
Creation date
Source (where from)
Organized course material
Chapter
Book
Workshop
Tutorial
Independent
Is source code available?
Hash *
Difficulty level *
File name(s) *
Deployment times
File type
Category *
Virus
Worm
Backdoor
Exploit
Trojan
Rootkit
HackTool
Spyware
Dropper
Downloader
Ransomware
Adware
BrowserModifier
Joke
Keylogger
Rogue
Spammer
Other
Platform *
Mobile
iPhoneOS
Android
Desktop
Windows
Win32
Win64
Linux
MacOS
MacOS
MacOS_X
210
Capabilities
Anti-Behavioral Analysis
Debuggger evation
debugger obstruction
Debugger prevention
Emuation evation
Emuation prevention
Memory dump obstruction
Sandbox evation
Sandbox obstruction
Sandbox prevention
VM evasion
Anti-static analysis
Obfuscation
Packing
linear disassembler prevention
Encryption
Anti-patching
Self-mutating
Metamorphic
Polymorphic

F Wiki from Phabricator

DreamFactory Setup

DreamFactory setup guide

This is a manual approach of starting the DreamFactory service with docker-compose. Skip to step 4 if the OpenStack Heat template is used, provided in the repository PLED_deployment.

Note, a user in this case could be a service, like PEMA. Remember to enable "Active" on users and apps or whenever it shows up.

1. Clone PLED repository.
2. cd into folder /pled/docker
3. Start the docker-compose file 'docker-compose-df.yml' with:

```
sudo docker-compose -f docker-compose-df.yml up -d
```

4. Enter the public IP-address of the instance running in your web browser to access the DreamFactory admin panel.
5. Create the admin user on the "first time setup page".
6. Go to Services and configure the 'mongodb' service.

Set host, port, databasename, username and password. If you have set up a replication set on different servers, use the "Connection String" to accommodate for the replicated servers.
NB: MongoDB username is *root* and the required password can be outputted by the administrator by running `openstack stack show <STACKNAME>`.

7. Create a Role, under Roles, and define permissions for each service the role is supposed to have.
8. Create a User under Users and assign role(s) to the user. E.g. if a user is supposed to have access to the API, it would be a good idea to grant permissions to also read the documentation.
   A. In DreamFactory you can either import users from xml, json or csv files or create new users with the forms presented in the GUI. You can find both methods in the Users tab (/dreamfactory/dist/index.html#/users), where create is on the top-left side and import on the top-right. Exapmle of json user:

```
{
            "id":123,
            "name":"name",
            "username":"username",
            "first_name":"firstname",
            "last_name":"lastname",
            "last_login_date":"yy-mm-dd hh:mm:ss",
            "email":"email@email.com",
            "is_active":true,
            "phone":"phone",
            "security_question":"securityquestion",
            "confirm_code":"y",
            "default_app_id":null,
```

```
            "oauth_provider":null,
            "created_date":"yyyy-mm-dd hh:mm:ss",
            "last_modified_date":"yyyy-mm-dd hh:mm:ss",
            "created_by_id":"id",
            "last_modified_by_id":"id",
            "confirmed":true,
            "expired":false
    }
```

9. In order to generate an API key, go to Apps and create a new app. Give it a name, description and a role. Leave "App location" on "No storage required". This key is required in the first time setup of the database web-interface.
10. Enable CORS under Config. Fill in the following fields:

```
Path: /api/v2/api_docs/mongodb/_table/*
Description: PEMA CORS
Origins: * (this allows any origin, alternatively use a comma delimited list for allowing
specific hosts to access the API)
Headers: * (defines allowed HEAD-ers)
Exposed Headers:
Max Age: 0
Methods: All
```

For further reading, visit:
https://guide.dreamfactory.com/docs/
http://wiki.dreamfactory.com/DreamFactory/Tutorials/Using_the_REST_API

Exploitdb Application Retriever

vulnRetriever.py setup

NB: The script is for scraping exploitdb for applications and can take up to 11 hours to finish

Automatic setup

If PLED is launched from the **HEAT** template (the recommended way), the database will have a sizeable amount of Exploitdb applciations already inserted (3000+ in time of writing).
If you have inserted applications and wish to run in update mode go to **Manual setup**
With automatic setup the script will be cloned down to the *manager* instance and is ready to use
A setting for enabling the script in Update mode with Crontab can be set in the *pled_top_env.yaml* file

Manual Setup

To manually setup the vulnRetriever script you need to clone down the PLED repository
In the *vulnRetriever* folder:
Run:

```
./install.sh
```

This will install the required python modules.
now you can run the script with:

```
python3 vulnRetriever.py
```

This will launch the script in normal mode, there are in total 4 different modes:

Mode	Argument	Description
Normal	none	Runs the script from start and inserts every application found into the database, if cancelled and restarted it will start from scratch
Quickscan	-q, --quick	Run in "quick" mode. Will store every exploitdb-id scanned in a file, so that if interupted it can start from last scanned it in the file.
ID scan	-i, --id	Requires an INT as parameter, will scan a given exploitdb-id to check if there is a vulnerable application available.
Update	-u, --update	Run in "update" mode. Will get the last published date from the database and only scan exploits that have been published after the given date.

If you previosuly ran the script in Update mode, and want to start from scratch you need to empty the checked.txt file (delete and remake)

Configuration

When starting the script manually, the following values needs to be set in *vulnRetriever/config/vulnRetriver.ini*:

- **Mongo_ip**: the ip of your mongodb host
- **Mongo_user**: Mongodb username (default: root)
- **Mongo_pw**: Mongodb password accosiated with username

Other settings:

- **StartDate**: Published date of applcations to start searching from (Default 2000-01-01)
- **Checkfile**: filename to be used when storing its when running in "Quickmode"
- **Exploiturl**: Baseurl of exploitd on exploit-db.com (Only change if exploit-db layout has changed)
- **Mongo_database**: Database to store applcations (Default: pled)
- **Mongo_collection**: Collection in database to store application documents (Default: vuln_application)
- **Csvurl**: Url to find all exploits on exploit-db.com in CSV format (Only change if layout has changed)

Logging

The script does not write to a log, but by default will print out data.
To get a log when running on a schedule, output all the data from the script to a file

```
python3 vulnRetriever >> <LOGFILE> 2>&1
```

Examples

Example of the script running in Crontab with logs to the home folder:

```
0 0 * * 0 cd /home/ubuntu/pled/vulnRetriever/ && /usr/bin/python3
/home/ubuntu/pled/vulnRetriever/vulnRetriever.py -u >>/home/ubuntu/.vuln.log 2>&1
```

This will run the script in Update mode every sunday at midnight

Example of running with exploit-db id specified:

```
python3 vulnRetriever.py -i <EDB_ID>
```

This will return if the id was found and if it was inserted

Example of running in quickscan mode:

```
python3 vulnRetriever.py -q
```

This will write each id checked to the checked file specified in *config/vulnRetriever.ini*
If interupted you can start again with same command and it will start of where it was interupted

OpenStack Heat deployment

How to deploy PLED

Prerequisites

- Some knowledge of OpenStack
- An OpenStack project with enough allocated resources. Estimated 32 GB RAM and 8 VCPUs, but this depends on flavors used in the Heat template.
- OpenStack CLI (sourced with the API access file from your OpenStack project)
- A security group called default

Deployment

First download the PLED Deployment repo.
Generate a key-pair in OpenStack and put the key in the root folder of the repository. Put the key-pair name in pled_top_env.yaml file (without the .pem extension). Other environment variables will need to be filled out as well, see below for further explanation.

Navigate to the root folder of the repository /PLED_deployment and run the command:

```
openstack stack create NAMEOFSTACK -t pled_top.yaml -e pled_top_env.yaml
```

Errors or status of the stack can be seen by running. This also reveals the password for MongoDB, which is required in the setup process.

```
openstack stack show NAMEOFSTACK
```

To se the event list during the stack creation, run the command

```
openstack stack event list NAMEOFSTACK
```

Environment file:

```
pled_top_env.yaml
parameters:
  key_name:    #Private key name.
  mgoexpress:   #Whether you want MongoExpress included or not.
  vulnRetrieverUpdateMode: #Wheteher you want vulnRetriever activated in UpdateMode or not.
  basic_auth_pw:    #Password for Docker Registry and MongoExpress.
  basic_auth_uname:    #Username for Docker Registry and MongoExpress.
  # For Docker Registry storage in Swift
  swift_username:     #Username for your OpenStack account with Swift accessibility,
possible a service user.
  swift_password:     #Password for said account.
  swift_domain: Default    #Default is for service users, NTNU would be used for a regular
```

```
    user.
      swift_tenantid:     #Tenant ID for the user/service user.
      swift_container:   #Name for the storage container in Swift.
```

Services

Service	Running on instance	Manual setup required
PLED database	dbreplica	username / password for mongodb in env file
Rest API	web-api	Setup in DreamFactory interface see **DreamFactory Setup** and **Dreamfactory Additional Services** for setup guides
vulnRetriever	dbreplica	see **Exploitdb Application Retriever** for setup and usage
Database management interface	db-web-interface	First time config required.
Docker Registry	docker-registry	username / password for basic auth in env file

Access logs - Docker services

To access Docker logs, use common logging commands together with docker logs.

```
docker-compose -f docker-compose-filename.yml logs [options] [servicename] >> output.log
```

Leave out [servicename] to output logs from all services in the compose file.

Some useful options:

```
--follow, -f    #stream output
--timestamps , -t
--until    #RFC 3339 date, a UNIX timestamp, or a Go duration string
--since    #RFC 3339 date, a UNIX timestamp, or a Go duration string
--tail 5    #output the last 5 lines from the log
```

See references for date/time syntax.

Docker log reference
Docker compose log reference

Database Setup

MongoDB Automatic setup and install

To automatically deploy and configure the database, refer to **OpenStack Heat deployment**

MongoDB Manual setup and install

Prerequisites

- docker
- docker-compose

The docker files used for Mongodb are from bitnami
The docker files used for Mongo Express are from Mongo-Express

This is a manual setup, if OpenStack Heat is used, found at our PLED Deployment repository, these steps are automated.

Setup

Clone the PLED repository

```
git clone ssh://git@git.ncr.ntnu.no/source/pled.git
```

Enter the *docker* folder.
The docker compose file for the Mongodb setup is called *docker-compose-mongorep.yml*

Configuration

Open up *docker-compose-mongorep.yml* in your favorite text editor (vim of course)
Edit the environment variables to your values
Username and password will be used for DreamFactory and general access to the database.
Replicaset key is used to authenticate the nodes in the replicaset.
Port number must be consistent across the replicaset.

For data persistence a volume is needed:
In PLED this data is stored on an external volume.
To configure this you will need an attached volume to the server you wish to run the database on.

Create volume:

```
sudo mkfs -t ext4 /path/to/volume
```

Mount volume with docker:

```
sudo docker volume create --driver local --opt type=ext4 --opt device=/path/to/volume
mongodb-master-data
```

Run install

If all is configured, the only thing needed to run is:

```
sudo docker-compose -f docker-compose-mongorep.yml -d
```

This will run the service in detached mode

The database should now be accessible on *<SERVER-IP>:27017*
To access the database you need to add the port to your security group in OpenStack

Mongo Express (optional)

Mongo Express gives a easy to use web interface to the database.
Run:

```
sudo docker run -d --network ubuntu_default -e ME_CONFIG_MONGODB_SERVER=<MONGODB-ADDRESS> -
e ME_CONFIG_MONGODB_ADMINUSERNAME=<MONGODB-USERNAME> -e ME_CONFIG_MONGODB_ADMINPASSWORD=
<MONGODB-PASSWORD> -p <PORT>:8081 mongo-express
```

With the serverip, username, password and prefered port.
Specified port must be added in the security group in OpenStack

If another network was specified in the *docker-compose-mongorep.yml* then change out *ubuntu_default* when running the above command.

If successfull the interface will be available at *<SERVER-IP>:<SPECIFIED-PORT>*

Backup and Restore

When deploying the service automatically with **HEAT** a cronjob is added to the backupserver to run dump of the database once per week.
To perform backups manually you can use mongodump the command for this is:

```
mongodump --username USERNAME --password PASSWORD --authenticationDatabase admin --
host MONGODBIP --portMONGODBPORT --db DATABASE --forceTableScan /destination/folder
```

This will create one *bson* file for each collection in the database
with this its also possible to backup single collections with :

```
--collection COLLECTION
```

If not destination is given the dump will go in a folder called *dump* in the working directory

Restoring from dump

To restore from a dump you can use mongorestore :

```
mongorestore --username ROOT --password PASSWORD --authenticationDatabase admin --host
MONGODBIP --port MONGODBPORT --db DATABASE /dump/folder
```

General info

At a minimum 3 different types of nodes are requires:

- Primary node
- Secondary node
- Arbiter node

To read more about mongodb replicaset refer to mongodb docs

Database Web Interface Setup

Get config values for first time setup

Prerequisites:

- Access to the OpenStack project where PLED is deployed

Values:

API key:

- Refer to DreamFactory Setup step 9

API URL

- Floating ip of DreamFactory server or balancer server, check SkyHigh interface or run `openstack server list`

S3 Key and S3 secret

- run `openstack ec2 credentials create`
- Key is field: *access*
- Secret is field: *secret*

S3 Region

- Region where OpenStack swift is running, default is **SkyHiGh**

S3 Endpoint URL

- URL where Swift container can be accessed
- Usually something like *https://swift.skyhigh.iik.ntnu.no/swift/v1/<PROJECT-ID>*

Docker Registry

prerequisites

- docker
- docker-compose

Automatic Setup

The Docker Registry is automatically deployed with the heat deployment.
Username and password for logging into the registry and the registry frontend will be specified in the env file for deployment.
The registry is available on *<docker-registry-ip>:5000*
the registry frontend is available on *<docker-registry-ip>:80*

Manual Setup

clone the PLED repository
move into the docker-registry folder

```
cd pled/docker/docker-registry
```

edit the *config.yml* file with your values, only values like *<OPENSTACK_USER>* are required to be changed.
For the automatic setup PLED used a service account fo SkyHiGh, contact system administrator to get this.
The full documentation for Swift Docker Storage driver is available at https://docs.docker.com/registry/storage-drivers/swift/

When the config is ready you must create the username and password for the basic authentication
First create a directory for the htpassw backup

```
mkdir -p /home/ubuntu/htpasswd_backup
```

Then make sure that you can write to the folder.
Then run the following docker command:

```
docker run --rm --entrypoint htpasswd registry:2 -Bbn <USERNAME> "<PASSWORD>" >/home/ubuntu/htpasswd_backup/htpasswd
```

This will create the basic auth data for the registry.

Now you are ready to start the registry:
run:

```
docker-compose -f /home/ubuntu/pled/docker/docker-registry/docker-compose.yml -f /home/ubuntu/pled/docker/docker-registry/docker-compose.auth.yml up -d --build
```

If successfull:
The registry is available on *<docker-registry-ip>:5000*
the registry frontend is available on *<docker-registry-ip>:80*

How to use

To use this registry you need to enable it as an insecure registry in your docker config.
The documentation for this is available at https://docs.docker.com/registry/insecure/

From the documentation

Edit the `daemon.json` file, whose default location is `/etc/docker/daemon.json`
If the `daemon.json` file does not exist, create it. Assuming there are no other settings in the file, it should have the following contents:

```
{ "insecure-registries" : ["<REGISTRY-IP>:5000"] }
```

Save the file.

Now you need to login with the username and password you configured earlier

```
sudo docker login -u <USERNAME> <REGISTRY-IP>:5000
```

A prompt for the password will appear, enter what you configured.
Will display *Login successfull* if successfull

Now you can pull or push images.
To see available images browse the registry frontend on *<docker-registry-ip>:80*

Pull image

```
docker pull <REGISTRY-IP>:5000/<IMAGE>
```

Push image

Tag image:

```
sudo docker tag <IMAGE> <REGISTRY-IP>:5000/<IMAGE>
```

Push image:

```
sudo docker push <REGISTRY-IP>:5000/<IMAGE>
```

Dreamfactory Additional Services

Configure scripts and external services in Dreamfactory

Two types of services will be explained in this guide:

- Python Scripts
- Remote CURL service

Python Scripts

To create a service in DreamFactory select the **Services** tab.
Select **Create**
Select **Script>Python**
Give the script a name (this will be used in the api endpoint)
Enter label and description.
Under the **Config** tab:
Upload a script or enter the source code in text field.
An example of a script install is provided below.

NB: any additional modules needed for python must be installed on the server

Customsearch service install
To install the *customsearch* service provided by PLED first download the *customsearch.py* script from
https://project.ncr.ntnu.no/source/pled/browse/master/misc/customsearch.py
In **info** tab:
Name: customsearch
Label : Customsearch
Description: Customsearch for retrieving all platforms and types from vuln_applications collection

in **Config** tab
Upload script file or manually paste inn source code

Press save.
Customsearch is now accessible *<dreamfactory-ip>/api/v2/customsearch/<platformORtype>*

DreamFactory REST API usage

Short guide on how to use the PLED Rest API. It is assumed you have knowledge about NTNUs adaptation of OpenStack and basic knowledge around how a REST API works.

1. Get an API-key for you application in the DreamFactory admin panel.
   A. Query PLED system owners or an instructor with administrative access to generate an API-key for your application.
   B. Specific roles/privileges can be assigned to applications wishing to make use of PLED. Depending on what permissions you get, you can use API calls like GET, PUT, DELETE and so on. You as a system owner should have access to the API documentation (Swagger.io) on what you can do with the API.
2. Interact with the API, either through the API Docs page (for testing and getting to know how it works), a web browser, or our GUI implementation found on the database web-interface.
   A. You will be presented with a JSON (default) or XML format request, similar to the following example:


```
{
    "_id": ObjectID("5ca321d14a71d03012a9c76f"),
    "cve_summary": "The lreply function in wu-ftpd 2.6.0 and earlier does not properly
cleanse an untrusted format string, which allows remote attackers to execute arbitrary
commands via the SITE EXEC command.",
    "cvss": "10.0",
    "cve": "CVE-2000-0573",
    "exploitdb_id": "20030",
    "application_name": "WU-FTPD 2.4.2/2.5 .0/2.6.0 - Remote Format String Stack Overwrite
(1)",
    "date": "1999-10-15",
    "type": "remote",
    "platform": "unix",
    "app_url": "https://www.exploit-db.com/apps/652cfe4b59e0468eded736e7c281d16f-wu-ftpd-
2.6.0.tar.gz"
}
```


In the above example, a response with the CVE-ID 2006-0006 is provided by PLED. There is a CVE-document with the CVE description, a CVSS score and an array with embedded documents containing metadata and a link on all applications related to this CVE.

Searching for applications:
Use the API Docs in DreamFactory to get familiar on how the API works, and what the requests looks like. Use SQL-like syntax for filtering what you want to search for, in this example we're searching for all vulnerable applications that can run on the "platform" Windows. And since the "platform" field is inside the embedded document "applications", we have to take this into consideration and use dot(.) notation.

```
http://IPADDRESS/api/v2/mongodb/_table/vuln_applications?filter=platform=windows
```

The code below is a curl example, supplied with an API-key which is needed for authentication.

```
curl -X GET "http://10.212.137.85/api/v2/mongodb/_table/vuln_applications?
filter=platform=windows" -H "accept: application/json" -H "X-DreamFactory-API-Key:
AVERYLONGKEY12345678901337"
```

A. Moen, A. Olsen, K. Arnesen

PLED (Pentesting Lab Environment Database)

NTNU
Norwegian University of
Science and Technology