Architecture Report

# Using Malware Analysis Results to Improve Security Requirements on Future Systems

# 

Table of Content

Introduction

[Intended Audience](#h.3dy6vkm)

[Project Background](#h.2jxsxqh)

[Keywords](#h.z337ya)

[Project Context](#h.3j2qqm3)

[Workflow](#h.2xcytpi)

[Architectural Drivers](#h.1ci93xb)

Architecture Views

[Design Tradeoffs](#h.1pxezwc)

# 

# Introduction

This is the architecture report of the project *Using Malware Analysis Results to Improve Security Requirements on Future Systems.* The document provides a background of the project and the business context, along with the key architectural drivers in the form of high level functional requirements, business and technical constraints and key quality attributes. The report discusses the different possibilities considered in achieving these architectural drivers, decisions made and their tradeoffs. At the end, different perspectives of the architecture have also been provided and discussed in detail.

# Intended Audience

This report is intended for the following stakeholders:

* Instructors
* Mentors
* Studio Team

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

Project Background

This project is a part of an SEI research and driven by two SEI researchers. These researchers are involved in the study of security requirements engineering and malware analysis and detection. According to their study, new software often have same security vulnerabilities that have already been exploited in previous software systems. The security in software is not improving, as it should be. So, the goal of their research is to make future systems more secure by making use of previously known vulnerabilities.

This project is based on the findings of their previous researches. According to them when software development is in the requirements and design phase, many things are taken care of with respect to security, but one area that has historically not been considered is the results of past malware attacks. Whenever malware attack happens on any software system, domain experts and various researchers analyze the attack to discover the vulnerabilities that have been exploited by the malware. The results are documented in the form of malware analysis report and are publically available on internet. Many organizations provide platform (web-based report writing application) for report writers (domain experts, researchers etc.) to write these reports. One of the most widely used report writing application is one provided by Rapid7. Requirements Engineers, Architects and Programmers often visit these reports to learn about the vulnerabilities being exploited so that they can make their future software more secure.

According to a study of the researchers, these malware analysis reports do not provide great value to requirements engineers and architects as they only talk about the vulnerabilities that have been exploited by the malware and not about the overlooked security requirements, which is the underlying cause of the vulnerabilities. So, to make these reports more valuable for requirements engineers and architects and increase the security of future systems, there is a need to include the misuse case, the use case and the overlooked security requirements in the report. The misuse case is the scenario that depicts how vulnerability might be exploited, the use case depicts the ideal scenario i.e. how the system should react to an attack and the overlooked security requirements are the requirements that have been overlooked while designing the software system and the underlying cause of the vulnerability.

# Keywords

CWE: **C**ommon **W**eakness **E**numeration is a community-developed dictionary of software weakness types. For simplicity CWE, can be thought of as a software weakness category.

Misuse case: Misuse case is the description of the exploitation of the vulnerability i.e. how a vulnerability could be exploited.

Use case: Use case is the description of the software system’s ideal reaction to an attack i.e. how a software should respond to an attack.

Overlooked Security Requirements (OSR): These are the requirements that have been overlooked while designing the software system and the underlying cause of the vulnerability.

MUO: Is often used in the report to jointly refer to **M**isuse case, corresponding **U**se case and **O**verlooked security requirements.

# Project Context

The goal of the project is to create a software system that will provide content to the report writing applications so that report writers can write more comprehensive reports and include the misuse case, use case and overlooked security requirements while writing the reports. The system will be referred to as the *SERF System.* The *SERF System* will contain a database of CWEs, Misuse cases, Use cases and Overlooked Security Requirements. The database of CWEs, Misuse cases, Use cases and Overlooked Security Requirements will be built with the help of the contributors, who are responsible researchers and domain experts who will be invited to write the Misuse cases, Use cases and Overlooked Security Requirements. Once the MUOs are written by the contributors, they will be reviewed by the reviewers, who are also domain experts and researchers but posses higher level of expertise in the area. Reviewers will have the ability to accept or reject the MUOs written by the contributors. Once accepted, the MUOs will be saved in the database and will be available for the report writing applications. The SERF System will expose an Application Programming Interface for report writing applications to interact with it and access the CWEs and MUOs. The system will also provide a web interface for the the contributors and reviewers where they can write and review the MUOs.

Since we do not have access to the organizations that provide report writing applications, we will also develop a dummy report writing application in order to test the *SERF System*. The *Report Writing Application* will mimic the already available report writing application, e.g. one provided by the Rapid7. The dummy *Report Writing Application* will have limited functionalities and will be used for testing and demonstration purposes only.

# 

# 

Context Diagram

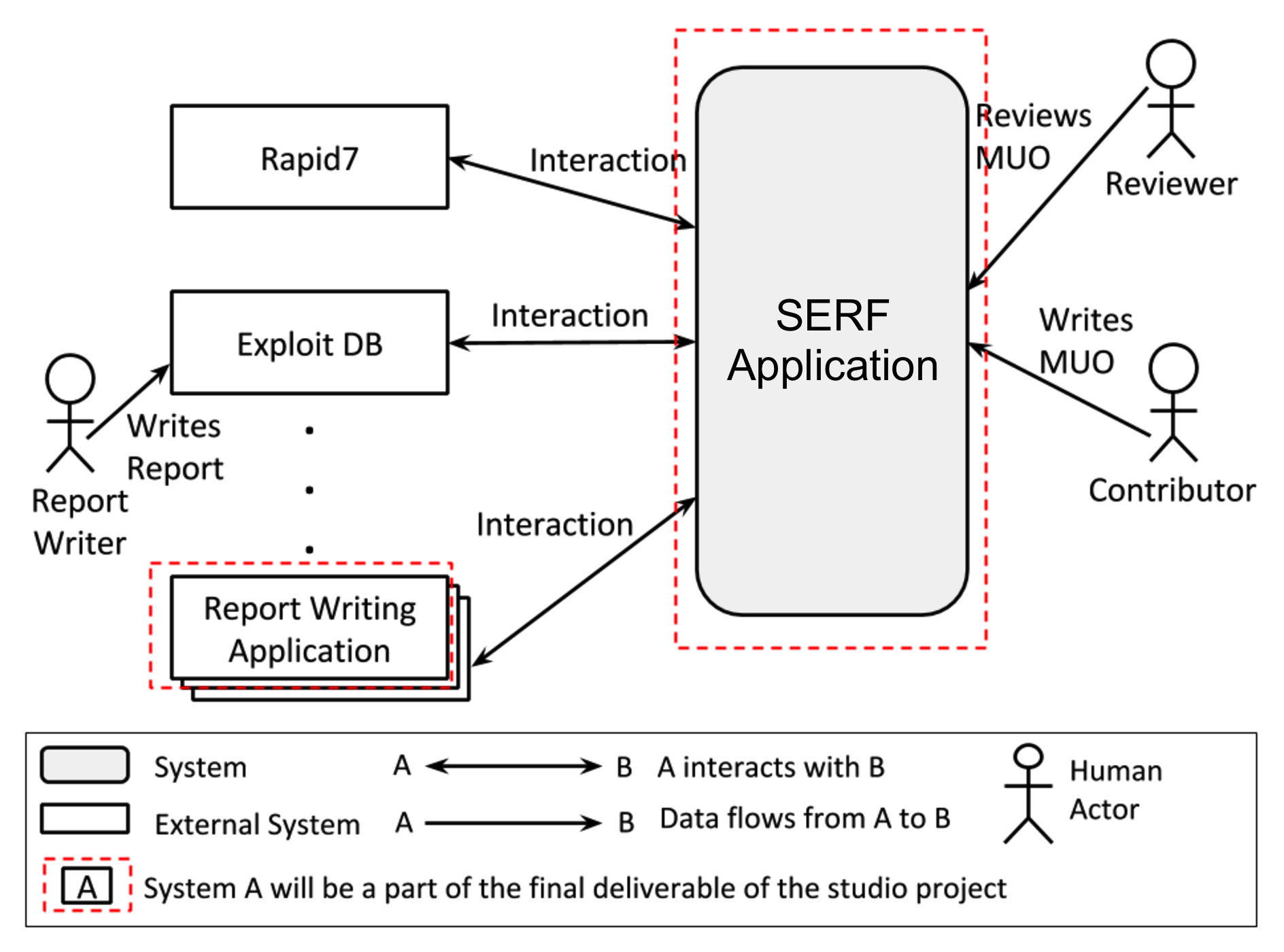


Fig. 1

The interaction between the *Report Writing Application* and the *SERF System* is depicted in detail in Fig. 2. This interaction will happen through REST API that *SERF System* will expose.

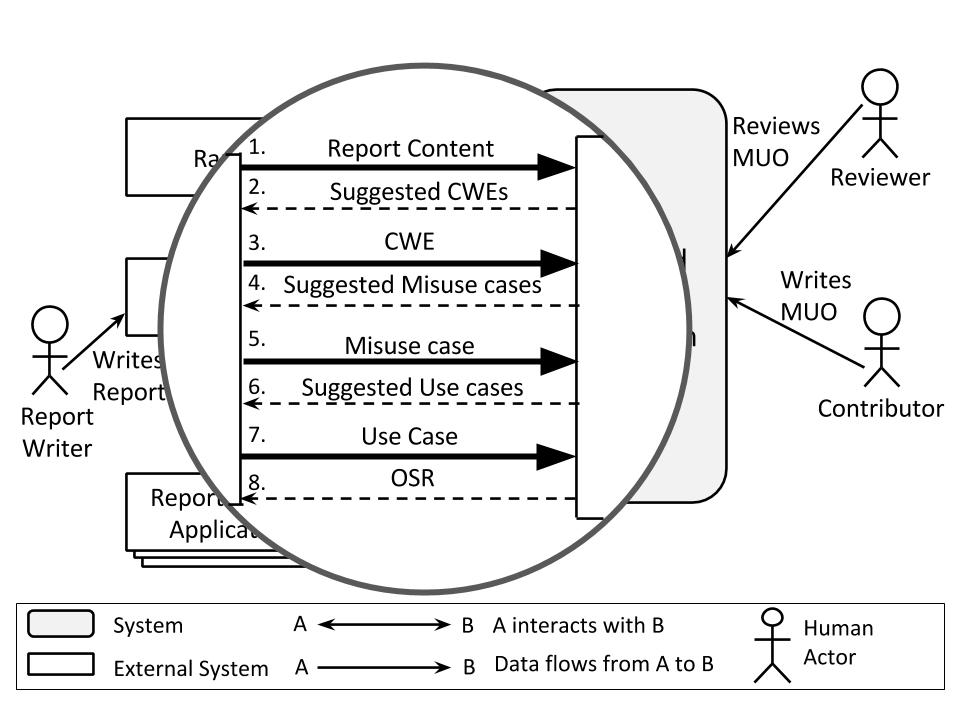


Fig. 2

# Workflow

The typical interaction between the Report Writing Application and the SERF System is as follows:

1. Report writer writes the description of the report using *Report Writing Application*.
2. Report writer request the related CWEs. *Report Writing Application* would make use of a web service from the *SERF System* that takes the description of the report in request, perform an algorithm on the text to find out the related CWEs and respond to the report writing application with the list of related CWEs.
3. Report writer selects a CWE that is most relevant for the scenario he/she wrote and requests for the related misuse cases. Again in the same way the *Report Writing Application* sends a request to the *SERF System* along with the selected CWE. The *SERF System* will respond with the related misuse cases for the CWE.
4. Report writer selects a misuse case that is most relevant for the scenario he/she wrote and requests for the related use cases. The *Report writing Application* sends a request to the *SERF System* along with the selected misuse case. The SERF System will respond with the related use cases for the misuse case.
5. Report writer selects a use case that is most relevant for the scenario he/she wrote and requests for the related overlooked security requirements. The *Report writing Application* sends a request to the *SERF System* along with the selected use case. The *SERF System* will respond with the related overlooked security requirements for the use case.
6. Report writer selects the relevant overlooked security requirements and saves the report. The report is saved in the *Report writing Application*.

# Architectural Drivers

High Level functional requirements:

* The *Report Writing Application* shall allow report writers to write report and request related CWEs and MUOs from the *SERF System*.
* *SERF System* shall provide an API for the report writing applications to interact with it and query CWEs and MUOs.
* Before using the services of the *SERF System*, the *Report Writing Application* has to subscribe to the services of the *SERF System*. The subscription shall be done by filling an online application form and verification. Once verified, the report writing application will be issued an unique alphanumeric key, which will be mandatory in every service request from the report writing application.
* The *SERF System* will contain the database of CWEs and MUOs.
* The *SERF System* shall provide a web user interface for *contributors* and *reviewers* for writing and reviewing the MUOs.
* The *SERF System* shall have a notification mechanism that notifies the respective users whenever an event occurs that is of interest to them. E.g. *reviewers* must be notified whenever a *contributor* submits MUO for review. The notifications can be of two types: In-app notification or an email notification.
* The *SERF system* shall provide keyword search algorithm, which will be used to find out the related CWEs based on the description of the report.
* Contributors and reviewers shall be authenticated when accessing the *SERF System.*

Business Constraints:

* This project has to be finished by mid December 2015 because that’s the time till when resources are available.
* No access to the organizations that provide report writing applications*.*
* Only five team members are available for the project and cannot be increased for the whole project duration.

Technical Constraints:

* A third party keyword-search algorithm is to be used to suggest the related CWEs based one the description of the report.
* Use of free technologies only, whether open or closed source.

Quality Attributes:

Modifiability

Since there are high chances that the CWE search algorithm/methodology will be replaced with another one if a better one is found in the future, it shall be possible to use a new *CWE search algorithm* with zero code change in the modules that depends on CWE search module and no change in the existing data model. Replacing the algorithm should not take more than 2 man-hours of work.

|  |  |
| --- | --- |
| Stimulus | Request to change the CWE search algorithm |
| Source of Stimulus | Customer |
| Environment | Development/Maintenance /Production |
| Artifact Stimulated | CWE Search module |
| Response | The CWE search algorithm is replaced |
| Response Measure | The algorithm is replaced within 2 man-hours with zero code change in dependent modules and no change in existing data models. |

Security (Brute-force Attack):

When a malicious user tries to access the SERF API without a valid authentication key and fails for 10 consecutive times within a minute, the system shall block the IP address of the user for 24 hours within 20 milliseconds after 10th unsuccessful attempt.

|  |  |
| --- | --- |
| Stimulus | 10 consecutive unsuccessful attempts within a minute to access SERF API |
| Source of Stimulus | Malicious user |
| Environment | System is in production and running |
| Artifact Stimulated | SERF API |
| Response | All the subsequent requests from the IP address is blocked for 24 hours. |
| Response Measure | The IP address is blocked within 20 milliseconds after 10th unsuccessful attempt. |

Security (Denial of Service):

A malicious user somehow accessed the valid API key of a Report Writing Application. He soaks the system with rapid requests for suggesting the related CWE through SERF API. The system shall notice the barrage of requests and if more than 100 requests are received within 1 minute, block the API key and notify system administrator within 20 milliseconds receiving the 100th request.

|  |  |
| --- | --- |
| Stimulus | 100 requests to suggest related CWEs from the same API key within a second |
| Source of Stimulus | Malicious User |
| Environment | System is in production and running |
| Artifact Stimulated | SERF API |
| Response | The API key is blocked and system administrator is notified |
| Response Measure | The API key is blocked and system administrator is notified within 20 milliseconds of receiving the 100th request |

Performance:

When a request for suggesting the related CWE is received at the *SERF System*, it shall not take more than 5 seconds to generate a response with the list of related CWEs for a report with a maximum of 150 words and a maximum of 5 concurrent users.

|  |  |
| --- | --- |
| Stimulus | Request to suggest related CWE |
| Source of Stimulus | Report Writer |
| Environment | The report description contains 150 words, and a maximum of 5 concurrent users |
| Artifact Stimulated | CWE Keyword search component |
| Response | Request is processed and response is generated with the list of related CWEs |
| Response Measure | The response is generated within 5 seconds of receiving the request. |

Note: The customer has not not decided where they’ll host the system, the performance scenario will be tested on the machine with following configuration:

* 1 GB RAM
* 1 CPU Core
* Intel E5
* 40 Gbit Network In
* 125 Mbit Network Out

Prioritization of Quality Attributes

Modifiability is the highest priority quality attribute because it is almost certain that the future teams will work on extending the system. There are very high chances that the existing CWE keyword search algorithm will be replaced with a different one because future focus of the customer is to improve the accuracy of the results and it is expected that multiple algorithms will be tested to find one with the better results. Since the system is about security, it is expected from this system to be secure in general terms. The system will be demonstrated to multiple stakeholders like authorities from Department of Defence, organizations that deals with security, domain experts, researchers, students, requirements engineers, architects, programmers etc., some of them may try to break the system in order to verify the security of the system. However customer do not anticipate orchestrated attacks on the system because it is a proof-of-concept and this version will not be accessed by the large volume of users or public users. Security of the system in general is important because it may damage the reputation of the customer who deals in the software security and are proposing a system to improve the security in software systems. The idea of security in the context of this project is to prevent the attacks that can be executed without much efforts. So, the priority is to prevent the easy attacks first hence, the Brute-force scenario takes higher priority than the Denial of Services scenario because in order to do denial of service attack on the system, user first have to request the API key so that they can send valid requests to the system that the system will respond to. Also it is easy to trace the denial of service attack because in that case the requests are coming from a source with valid API key. The least priority quality attribute is *Performance* because the customer’s concern is that the system should not take much longer to respond. Performing poorly while demonstration to the stakeholders will leave bad impression on stakeholders which may in turn inhibit the prospects of the adaption of the system. So the priority of the quality attributes is as follows:

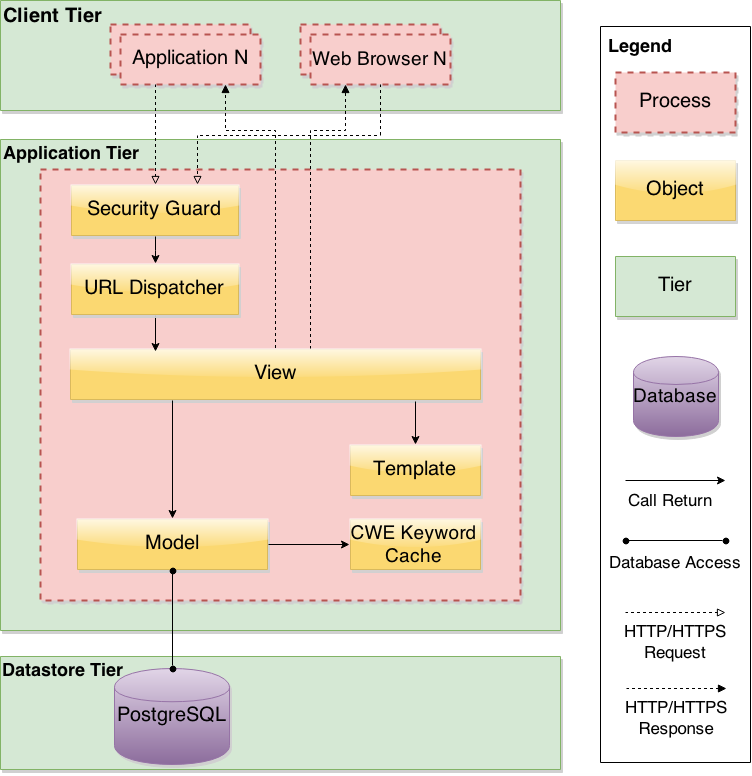
1. Modifiability
2. Security (Brute-force)
3. Security (Denial of Service)
4. Performance

# 

# 

# Architecture Views

# **Dynamic View**



**Static Views:**

Model-Template-View (MTV) Architecture

Fig.4 shows the overall partitioning of the implementation units of the system. We are using a variation of MVC pattern, which is Model-Template-View where the code is split into Model, View and Template. Models contain the data access logic. Views describes the data that gets presented to the user. It’s not necessarily how the data looks, but which data is presented. Templates on the other hand describe how the data looks like and how it is presented. There is no direct architectural driver for choosing this pattern and was mainly influenced by the framework we will be using for developing the project, which is Django. Still, separating the system in this way allows decoupling among the code modules and changes in the modules are localized. For example, if one wants to change in the code related to the presentation, they would just need to change the pertaining module in the Templatewithout any need to change in the other modules e.g. Views or Models. Similarly if any code change is needed related to the data access classes, the code change will be localized in the Modelsmodules and no change is needed in the ViewsorTemplates*.*

Other than Models, Views and Templates, the CWE Search module contains all the classes that contain code related to the CWE search algorithm, which is explained in detail in the next section.

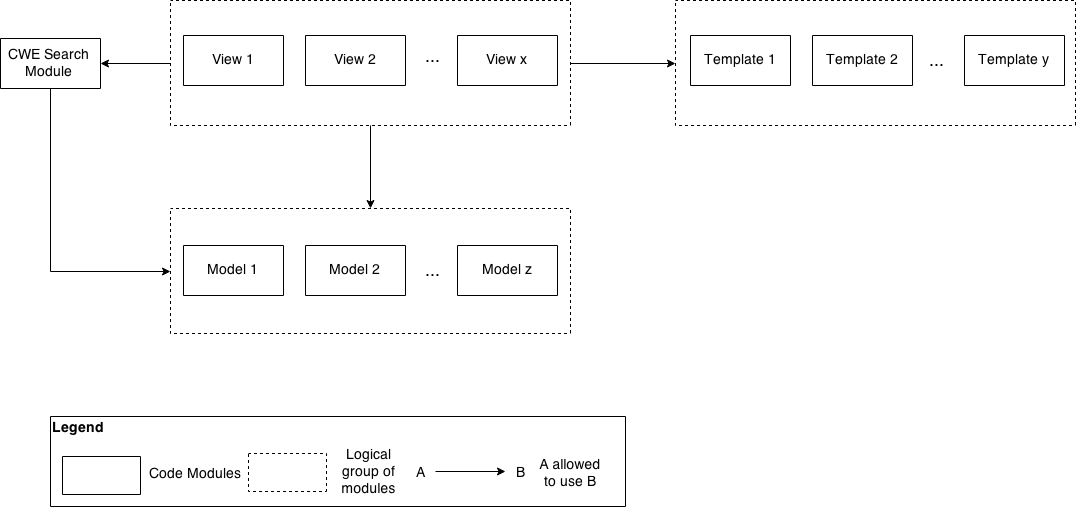


Fig. 4

CWE Search Module

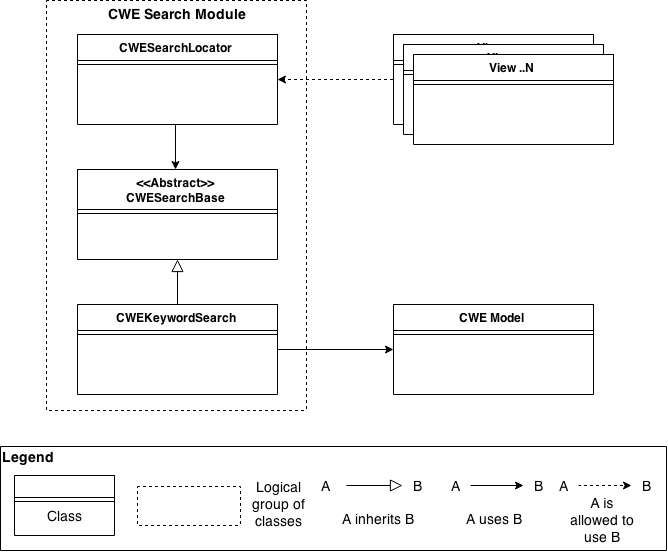


Fig. 5

As the modifiability quality attribute stated, there are high chances to do more work on the CWE keyword search algorithm in the future and replace it with better algorithms. To reduce the cost of change in the future, and given that future maintainers of the system are not the same team who developed it, the goal is to allow adding new CWE search algorithms and replacing the old one without changing the code of any of the already deployed modules and data model.

To achieve this requirement, the CWE Search module has been designed using the Service Locator pattern which provides late bindings of the CWE search algorithms implementations and provides abstractions for selecting the appropriate CWE search implementation in the system. These are the main classes of the module:

**CWESearchLocator:** This is the actual service locator and acts as a simple run-time linker for the search implementation. This is the public class of the CWE module to the rest of the modules in the system to select the appropriate CWE search algorithm in an abstracted and invisible way. It holds an instance of CWESearchBase along with its priority. When a new CWE search algorithm is added to the system, it needs to register itself with the CWESearchLocator and pass a priority that should be higher than the priority of the current registered implementation. If the priority of the new implementation is higher that the original one, then it will be set as the default implementation and serviced to the whole system. Otherwise, the original implementation will continue to be the default one. The current implementation requires restarting the application to register a new implementation as runtime switch is not critical to the client.

**CWESearchBase:** This is an abstract class that provides the basic abstract methods required to be implemented by all search algorithms. It acts as the contract between the algorithm implementations and the system modules who will use these algorithms.

**CWEKeywordSearch:** This is the basic CWE search algorithm that will be delivered as part of this project. It has to be registered with the CWESearchLocator and shall receive minimum priority so that future implementations will replace it. When future implementations are available, they shall also implement the CWESearchBase and register with the CWESearchLocator. If the new implementation registers with a higher priority than the previous one, it will be used as the default implementation. Therefore, we will be able to change the CWE search algorithm without changing the code of any of the previous modules.

# Design Tradeoffs

The biggest design tradeoffs that we faces were between performance and security of the SERF API provided to report writing applications. As explained earlier, security is of a higher concern to the client as the whole project is about enhancing security in future systems. Performance, mainly reflected in the response time of API requests to suggest CWEs in a report, is also of high importance as no report writing application will use the system if it takes a considerable amount of time. Unfortunately, performance and security are inversely proportional to each other and we had to balance between them to satisfy the customer’s requirements and achieve the project’s goals.

The first action that team took before making any decisions and evaluating decision tradeoffs, is to prioritize the quality attributes using customer inputs, specially between performance and security. The results as explained earlier show that security in general has higher priority than performance.

First of all, a key security concern from the client was brute-force attacks to access restricted areas of the system. Protecting logging into the system’s web interface can be simply achieved using captchas and similar techniques. However, such techniques cannot be applied to authenticating REST API requests and prevent brute-force attacks. Therefore, the requirement states that the system shall block API access from the IP from which 10 consecutive failed access attempts are received within a minute and the blocking should be done within 20 milliseconds after the 10th unsuccessful attempt. Implementing this feature does have a minor performance drawback as each failed access attempt has to be recorded and check the total failed attempts in a specific interval to find out whether this IP reached the limit of failed attempts and should be blocked or not. However, we chose to include this feature as security has higher priority than performance and the performance will only be affected in failed access attempts and not the successful ones.

Another key security concern from the client is protecting the system from denial of service attacks. Tactics to protect a system from Denial of Services DoS spans from the ISP, cloud providers, hardware firewalls, software firewalls, web server and lastly the application itself. We are only considering tactics related to the web server and application itself and the rest is considered out of the scope of the project. Even though DoS attacks should be prevented from reaching the application as it is considered already late if these attacks reach the application, still adding a DoS protection layer in the application will promote security and protect against the attacks that seemed normal to the underlying layers. However, this will have huge impact on performance as each request has to be logged and matched with heuristic patterns to detect when DoS attack is taking place. The team in this case had to balance between performance and security requirements and we decided to protect against DoS attacks only for the lengthy and heavy operations in the system, which we limited it to suggesting related CWE through SERF API as explained in the quality attributes section. The reason for choosing this design decision is that it might take tens of thousands of requests per second to bring the system down by querying normal and light operations, which can be easily detected as DoS by the other means explained earlier, such as web server and firewall. However, it might only require few hundreds requests per second to bring the system down if querying heavy operations such as suggesting related CWEs from a report description, which might skip the firewall detection.