



UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Escola Tècnica Superior d'Enginyeria
de Telecomunicació de Barcelona



SecDevOps modeling for web services and applications

Master Thesis
submitted to the Faculty of the
Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona
Universitat Politècnica de Catalunya
by

Enric Carrera Aguiar

In partial fulfillment
of the requirements for the master in
Advanced Telecommunication Technologies

Advisor: Silvia Llorente
Barcelona, 2022



Contents

List of Figures	4
List of Tables	4
1 Introduction	8
1.1 What is DevOps?	8
1.2 DevOps principles	8
1.3 Security in DevOps (DevSecOps & SecDevOps)	10
1.3.1 DevSecOps:	10
1.3.2 SecDevOps:	11
2 State of the art	13
2.1 Model analysis	13
2.1.1 SAMM	13
2.1.2 BSIMM	16
2.2 Model comparison	18
2.2.1 Type of model	18
2.2.2 Measurability approach	19
2.2.3 Structural comparison	20
2.2.4 Conclusion	20
3 Methodology / project development:	22
3.1 Governance	23
3.1.1 Strategy & Metrics	24
3.1.2 Policy & Compliance	26
3.1.3 Education & Guidance	28
3.2 Design	28
3.2.1 Threat Assessment	29
3.2.2 Security Requirements	31
3.2.3 Security Architecture	32
3.3 Implementation	33
3.3.1 Secure Build	34
3.3.2 Secure Deployment	35
3.3.3 Defect Management	37
3.4 Verification	39
3.4.1 Architecture Assessment	39
3.4.2 Requirements-Driven Testing	42
3.4.3 Security Testing	43
3.5 Operations	44
3.5.1 Incident Management	45
3.5.2 Environment Management	46
3.5.3 Operational Management	47
4 Results	49

4.1	Web Application	49
4.2	Validation and Testing of the Application	52
5	Budget	59
6	Conclusions and future development:	60
6.1	Model Comparison:	60
6.1.1	Validation	60
6.1.2	Governance	61
6.2	Practical work:	61
	References	63

List of Figures

1	DevSecOps development cycle [6]	11
2	SAMM structure [10]	14
3	SAMM structure from [10]	15
4	BSIMM model [13]	17
5	Questionnaire example for the initial assessment using SAMM	22
6	SAMM questionnaire scorecard	23
7	OWASP Top 10 mapping from 2017 to 2021 version [17]	27
8	CWE top 25 Most Dangerous Software Errors List [20]	28
9	Technical architecture of a Django web application	40
10	Application functional architecture diagram	40
11	Web Application login page screenshot	49
12	Web Application contract list page screenshot	50
13	Web Application detailed contract page screenshots	51
14	Web Application new contract entry form page screenshot	52
15	Trigger configuration for CodeQL	53
16	Programming language configuration for CodeQL	53
17	Results of the Static Application Security Testing (SAST) performed with codeQL	54
18	Results of the Dynamic Application Security Testing (DAST) performed with ZAP	54
19	Detailed information on the Missing CSP header vulnerability	55
20	List of middleware components located in the main configuration file for the web application	55
21	Directives configuration for the CSP middleware component	56
22	Blocked components by the CSP header	56
23	New alerts reported in the DAST analysis	56
24	Detailed information on the CSP wildcard directive vulnerability	57
25	Detailed information on the no HttpOnly vulnerability	57
26	HttpOnly Header configuration	58
27	Results of the analysis with ZAP after fixing the HttpOnly header vulnerability	58
28	Detailed information on the timestamp disclosure vulnerability	58

List of Tables

1	Overall comparison between SAMM and BSIMM	21
2	Table with the threat categories of the STRIDE model [23]	31
3	Information about the dependency used to implement the web application	35
4	List of possible defects (some of them) that can affect the application in development	38
5	Staff budget	59
6	Hardware budget	59

Acronyms

ASVS Application Security Verification Standard. 27, 28

BSIMM Building Security In Maturity Model. 13, 16, 17, 18, 19, 20, 22, 60

CSP Content Security Policy. 55, 61

CSRF Cross Site Request Forgery. 52

CWE Common Weakness Enumeration. 27, 28, 34

DAST Dynamic Application Security Testing. 4, 42, 43, 44, 53, 54, 60

DDoS Distributed Denial of Service. 41

DevOps Development Operations. 8, 9, 10, 11

DevSecOps Development Security Operations. 10, 12

GDPR General Data Protection Regulation. 47

OWASP Open Web Application Security Project. 13, 15, 20, 22, 27, 28, 42, 44, 52

SCA Software Component Analysis. 34, 35, 60

SIEM Security Information and Event Management. 19

S-SDLC Secure Software Development Life Cycle. 18

SAMM Software Assurance Maturity Model. 13, 14, 15, 17, 18, 19, 20, 22, 23, 29, 60, 62

SAST Static Application Security Testing. 4, 34, 35, 43, 44, 53, 54, 60

SDLC Software Development Life Cycle. 8, 13, 16, 22, 26, 38

SecDevOps Security Development Operations. 11

SSDL Secure Software Development Life-Cycle. 18

W3C World Wide Web Consortium. 55

XSS Cross Site Scripting. 31, 51, 52

Revision history and approval record

Revision	Date	Purpose
0	06/03/2022	Document creation
1	30/08/2022	Document revision

DOCUMENT DISTRIBUTION LIST

Name	e-mail
Enric Carrera Aguiar	enric.carrera.aguiar@estudiantat.upc.edu
Silvia Llorente	silvia.llorente@upc.edu

Written by:		Reviewed and approved by:	
Date	30/08/2022	Date	30/08/2022
Name	Enric Carrera Aguiar	Name	Silvia Llorente
Position	Project Author	Position	Project Supervisor

Abstract

The concept of Web Applications and software in general is very present in our everyday lives, and from the consumer perspective it might seem simple, as we only see the final product. But, to develop this software, there is a lot going on behind the scenes that we do not see, in fact, building software continuously and with a certain level of quality is a very complex task and takes a lot of effort. In this project I aim to analyse and compare existing application development security models and tools, focusing on how they are applied to DevOps (Software Development and IT Operations). From the comparisons, I will choose the best model according to my opinion and I will apply the model to the implementation of web services and applications. After the implementation of the use case. At the end, I will discuss possible improvements and changes for the approach used to develop the application.

1 Introduction

1.1 What is DevOps?

Web applications are a big part of today's society. They have become a very common and worldwide tool for providing digital services of all kinds, but the process behind the development and maintenance of them is not simple. There are two main groups or teams involved in the creation of such applications, the IT (information technology) team and the software development team which worked separately in older work cultures or models. In essence, the software development team was in charge of writing the application source code, and the IT team was the one in charge of the deployment and support tasks.

Around 2007 a different approach called DevOps (Development Operations) gained popularity, aiming at the integration of the two aforementioned disciplines into one continuous process. The main objective of developers and IT operations teams working collaboratively through the whole software development lifecycle (SDLC), is to improve the speed and quality of the software development and deployment.

1.2 DevOps principles

Though DevOps is a practical methodology, it is also fundamentally a mindset and cultural shift in an organization. Several key principles or ideas characterize this philosophy [1]:

- **Automation:** Automate everything, such as workflows, testing new code, and how your infrastructure is provisioned to avoid overworking.
- **Iteration:** Write small pieces of code at a time to support releases and sub-releases that increase the speed and frequency of deployments.
- **Continuous improvement:** Continuously test, learn from failures, and act on feedback in order to optimize performance, cost, and time to deployment.
- **Collaboration:** Unite teams, communicating and breaking down the barriers between development and IT operations.

These principles support an agile style of working with a great focus on automation and the usage of tools to improve the efficiency of the development process. Meanwhile, its iterative workflow allows for continuous development and integration of software. These features help businesses to achieve a faster delivery of new features or software products that are of higher quality, since they can release simpler or more early working versions that have no errors at all and improve them along the way while the actual application is already functional.

NOTE: *Agile methodologies are iterative approaches to project management and software development. Instead of delivering everything in one big release, agile workflows deliver work in small, but consumable, increments. Requirements, plans, and results are evaluated continuously so teams have a natural mechanism for responding to change quickly. The most well known methodology is Scrum, but there are also others such as Kanban. [2]*

The DevOps workflow [3] can be different depending on who is implementing it but the basic processes that usually take place in some way or another are:

- **Planning:** In this process, DevOps teams conceive, define and describe the features and functionality of the applications and systems they will build. Tracking progress both broadly and in detail, from single-product tasks to tasks that span portfolios of numerous products. Creating backlog logs, tracking bugs, managing agile software development and visualizing progress are some of the ways DevOps teams plan with agility and visibility.
- **Development:** The development process includes all aspects related to programming like writing and testing as well as compiling that code into artifacts that can be deployed in various environments such as .jar files for Java based software. DevOps teams seek to innovate quickly without sacrificing quality, stability or productivity. To do so, they use highly productive tools, automate day-to-day and manual steps, and iterate code in small increments through automated testing and continuous integration.
- **Release & Deployment:** This stage refers to the process of deploying applications into production environments in a consistent and reliable manner. The release phase also includes the implementation and configuration of the fully governed core infrastructure that constitutes those environments such as container environments. Also in this stage, teams define a release management process with clear manual approval phases, as well as establishing automatic gates that move applications from one phase to another until they are available to customers. Automating these processes makes them controlled, scalable and repeatable. In this way, DevOps teams can deliver with ease without unexpected mistakes.
- **Operation:** The operation process involves maintaining and monitoring the released applications, as well as troubleshooting potential problems in production environments. Teams work to ensure reliability and high availability, with the goal of zero system downtime, while strengthening security and governance. DevOps teams seek to identify issues before they impact the customer experience and mitigate them quickly as they arise. Maintaining this vigilance requires very comprehensive metrics, actionable alerts and full visibility into applications and the underlying system.

As a whole, the DevOps model has many benefits, such as the collaboration and trust between the different teams involved in development, thanks to it being a communication based approach. It also provides a faster and simpler approach to releasing software thanks to the focus on using automated tools and predefined processes, which helps providing more consistent results, and at the same time it facilitates the resolution of issues by making the response time lower than other more ad-hoc methods of working.

1.3 Security in DevOps (DevSecOps & SecDevOps)

As applications become more complex they tend to process more data, which has advantages such as services being more customized, but at the same time it creates a big target for cyberattacks, since retrieving or stealing private information is one of the objectives of cyberattackers. For this reason, security plays an important role in the development process of an application, but it is not enough with applying patches to existing vulnerable applications, instead, security has to be taken care of and applied throughout the whole life-cycle of the application in each stage and process.

The agile workflow of the DevOps model helps having up-to-date secure applications and processes, since changes can easily be made in different sprints and any new vulnerability detected can be patched or fixed efficiently in a short period of time. This is a very important feature since security vulnerabilities and other related aspects are in constant evolution, for example, a development team might be used to working with or implementing a third party library for their applications, but a new version is released by the software provider and it contains a security bug that can consequently affect the final product released if it's not tested previous to its use.

1.3.1 DevSecOps:

This integration of security processes, active security audits and security testing into the agile development workflow of DevOps is named DevSecOps [5], and rather than applying security to the end product (application or software), aims to have security built into the product by applying the concept of security by design, which means that security is taken into consideration from the very beginning. From the tools that are used (adapting the tool stack to introduce security features) to the design and implementation of the software, security will be present throughout the whole life-cycle of the software we are developing.

The main objectives or propositions of this work model are:

- **Accelerate the deployment frequency:** When security is integrated into the development pipeline, eventually the development will be much more agile due to less errors/mistakes happening that can delay the deployment of products.
- **Decrease time to remediate critical vulnerabilities:** Due to security being spread out throughout the whole SDLC, errors can be caught earlier and fixed before they are harder to fix, like in pre-deployment environments where the whole application is already built and changes are harder to implement.
- **Improve the security posture:** Including security as a feature from the start saves a lot of time to security teams by eliminating benign issues or false positives thanks to automated processes gating at every step of the SDLC.

DevSecOps has a similar workflow and life cycle as we saw previously for DevOps, but the tasks or activities that each process or stage will take part on are going to be expanded with the new security activities that will take place. Essentially we are adding an extra layer of security activities on top of the DevOps cycle as shown in figure 1.

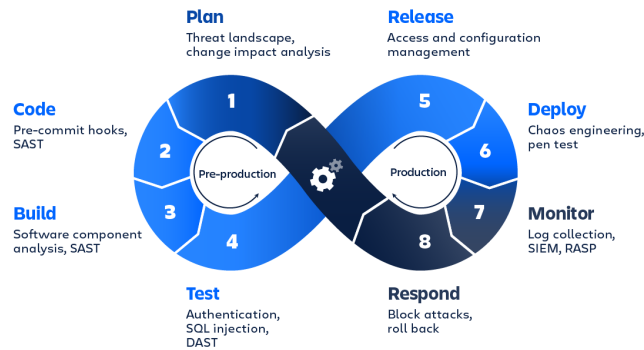


Figure 1: DevSecOps development cycle [6]

These security activities refer to all the tools and techniques needed to design and build software that resists attacks, as well as to detect and respond to vulnerabilities (or actual intrusions) as quickly as possible. Some examples of these activities can be:

- During the planning phase, a threat modeling process can be implemented to be able to know what are the possible threats the software can be faced against, and properly create a good strategy to defend against those possible vulnerabilities during the coding stage.
- Another process that is very important is, even after the testing stage of the application, a test environment simulating the real one can be set up and execute pen testing strategies to try and find any possible mistakes or vulnerabilities that were not taken into account previously. This way organizations can be more certain that it is indeed hard to attack the software and it's ready to be released to the public.

1.3.2 SecDevOps:

Another approach to security in DevOps is SecDevOps [8]. This other security model is slightly different in the way they intend to secure the software development life-cycle, because, instead of introducing security activities as a new feature to the DevOps model and have them like a separate team that still works collaboratively with the others, in this model security comes first, and all decisions taken are from a security perspective. This means that speed of release is not a priority anymore which can have a negative effect on the development process due to focusing too much on security and not on the product being released.

SecDevOps aims to ensure that security becomes a shared responsibility between all members in the DevOps team across the entire application life-cycle, describing a mindset where everyone involved in DevOps understands security. The difficult part about this, is that finding people that are good at development but also have all the knowledge in security is much harder to find. This is the drawback of not having a specialized team in security, which is able to coordinate and guide the other teams on the security efforts that need to be made. It is true that ultimately it would be ideal for every member of the

DevOps team to be knowledgeable in their security practices, but to expect that from the beginning without some guidance from experts in the matter, is much more complicated to achieve, reason why DevSecOps is more popular.

2 State of the art

2.1 Model analysis

In this next section I am going to introduce and analyze two of the most important models used by corporations and companies to structure and improve the way in which they approach their software development from a security standpoint.

The two models I am going to analyze are the Software Assurance Maturity Model (SAMM) [10] by OWASP and the Building Security In Maturity Model (BSIMM) [12] by Synopsys, and later I am gonna make a comparison on some of the most important features they have to see what are the differences between their approaches [15].

2.1.1 SAMM

Before starting with the model itself, I am going to talk about OWASP and how they came to the point of developing this model. Open Web Application Security Project (OWASP) is a non-profit organization that aims at improving the security of software through several open source projects such as SAMM.

SAMM (Software Assurance Maturity Model) is an open framework that provides a simple, effective and measurable way to analyze and plan the security initiatives and infrastructure of all types of companies regarding their software. This initiative brought up by OWASP wants to help organizations analyze their current software security practices and build a properly secure Software Development Life Cycle (SDLC) in an iterative manner, allowing for a progressive improvement that comes from the use of new or improved security-related activities. And, because it is an open framework, anyone who wants to use it can obtain the full benefits of the model without paying, opposed to other models.

The framework is descriptive, meaning that the organizations that are using this model are only receiving recommendations and are not being forced to follow their scheme to the last detail, since in this area of work, there is not a perfect single solution that works for every use case. This is why SAMM was developed with flexibility in mind, to allow their users to define what are their priorities in terms of security and adapt the model to what they specifically need or want to focus on. Once they have designed their own action plan, they can follow the recommended activities that lie in the different security practices and areas of the model. To create a flexible model they opted for a generic description in the different activities that does not necessarily depend on the specific technology that different companies will use to perform them. But, although being generic, these solutions or activities proposed, are provided with enough details and easy enough to follow even for non-security personnel, with the objective of having a cohesive team that understands the importance of these activities within the development of software.

Following the principle of flexibility that I was talking about, this model is made so that it is not necessary to get the highest scores possible in every area or category, since some activities may be more important than others for different organizations and in some cases it might even not make sense to implement some of them either, because the resources are limited or because the development cycle they have in place does not benefit from it.

An example could be a company that only uses self developed components in their code, versus another that does use third party components such as libraries. In the first of these cases, it will not be as important to have a specific process to keep up with the updates of these third party components, meanwhile, the second company will benefit from it to make sure the software they are using from other sources is indeed secure and no bugs or updates are reported. This goes to show that the model is built to be fitted to the needs of the customer no matter the circumstances.

Structure:

The scheme of SAMM has 3 main levels to represent the different areas and practices in which the proposed activities lie within. At the top and most general level we find Business Functions, they represent the general aspects of the Secure Software Development Life Cycle such as Design, Verification (basically testing) or Operations which is the area in charge of maintenance and monitoring tasks. On a second level which they call Security Practices, we have subgroups or subareas within each business function, such as Threat assessment which is a security practice carried out in the design business function, and there are three of these for each top level group. Finally, Security practices have the activities grouped in logical flows and divided into two streams. Streams cover different aspects of a practice and have their own objectives, aligning and linking the activities in the practice over the different maturity levels.

Maturity levels are ranges in which the sets of activities of every Stream are divided into, in such a way that activities in lower levels are normally the most basic or easier to execute as well as requiring less formalization, in order to build a base for more mature activities that come in superior levels. For each Stream in each Security Practice, SAMM defines three maturity levels. Each level has a successively more sophisticated objective with specific activities, and more strict success metrics.

In the end, the structure of this model is composed of 15 Security Practices grouped into 5 different Business Functions. And, to obtain a measurable model, a maturity level approach has been used for the framework, which evaluates the maturity within each of the streams of activities.

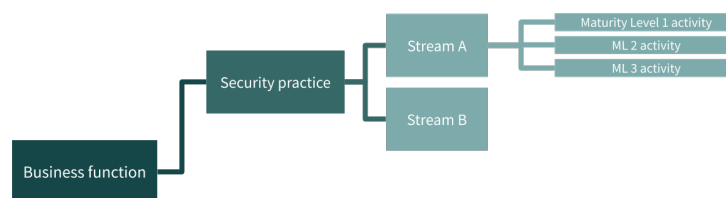


Figure 2: SAMM structure [10]

In the following image the overall structure of all the different business functions, security practices and streams is represented in a diagram from OWASP's SAMM presentation page.

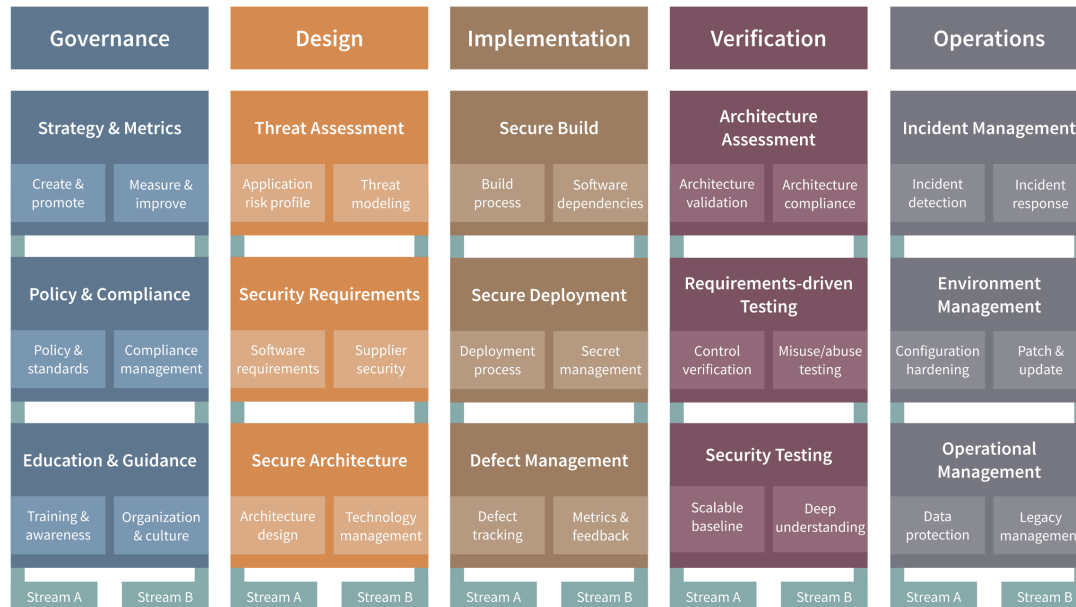


Figure 3: SAMM structure from [10]

In that top and most generic level, the five main Business Functions defined are: Governance, Design, Implementation, Verification, Operations.

- Governance:** This function refers to all the processes and activities related to the management of the software development plan. These processes include the definition and management of metrics, the decisions regarding the strategies to be followed by the development teams, or the definition and compliance control of policies at both organizational level and industry level between others.
- Design:** This concerns everything related to how an organization defines goals and creates software within development projects. Normally, these processes include the gathering of security requirements for development projects, high-level architecture specifications and assessing the possible threats that the software being developed may face, to define what specific countermeasures can be used.
- Implementation:** The focus of this function is on the processes and activities related to how the software products are built and deployed, as well as all the logistics involved with the management of the possible defects. The actual software developers are the ones most impacted by the activities and processes within this function, including for example, the coding of the software and the integration of all the dependencies, which at the same time need to be monitored in case a vulnerability arises regarding one of them. The goal of this function is mainly to be as consistent as possible in producing reliable and secure software and at the same time having the capacity to react fast against defects.

- **Verification:** In this group of processes we find everything related to how an organization checks and tests artifacts produced throughout the software development, such as the final executable for an application, to try and find any possible mistakes or vulnerabilities that were not accounted for in earlier stages of the development. This typically includes quality assurance work, but it can also include other review and evaluation activities such as compliance with the guidelines imposed by the organization or the industry.
- **Operation:** This last business function, groups all the activities necessary to ensure that confidentiality, integrity, and availability are maintained throughout the operational lifetime of the software and the data managed. This includes from the management and monitoring of the production environments, to the adaptations and actions that need to be carried out when disruptions and other changes in the operational landscape happen. And last but not least, this group of activities also includes the management of the legacy software and the secure retirement of their active services.

Overall, the three main characteristics that this model is trying to achieve are to be measurable, versatile and actionable as they mention. The measurability aspect comes from the maturity model approach used, which gives certain levels of maturity depending on what processes and how are they being performed, and based on that, we can extract statistics of maturity in different areas of the SDLC. Next up, to tackle the versatility aspect, the activities and processes proposed in every branch of the security model are independent of the technologies or workflows that each organization may use, allowing the model to be used by anyone, and finally, due to the clearly defined pathways obtained with the use of streams of activities, paired with the maturity model, it is very easy to define strategies and plans to improve the security efforts that are put into the Software Development Life Cycle (SDLC).

2.1.2 BSIMM

BSIMM stands for Building Security In Maturity Model and it is an annually updated framework developed by a Tech Company named Synopsys. The idea for this secure development model started in 2008 with the intention of studying the security efforts that were being put up by multiple organizations across many different industries, in order to design a model that would help other organizations no matter the size, to start planning and executing their own security initiatives.

As mentioned, the method they use to define and update the model in an annual basis, consists essentially of partnering with some organizations, with the intention of studying and observing what are their security initiatives and strategies to approach security in the SDLC. Then, after the information gathering and documenting phase is finished, they use this insight information to build their framework and the security activities that are contained in it, therefore, making this a descriptive model, since they are basically quantifying these activities carried out by their partner organizations and representing them in what can be called a measuring stick to compare yourself against. Essentially, this model is not focused on giving a roadmap designed by them for their users, instead, they

aim at providing information and ideas with activities that come from real world security initiatives, so that they can make their executive decisions based on those options.

When they began back in 2008 only 8 organizations were part of this observation stage, but with time, in each iteration of the model more organizations have come along (around 130 in BSIMM version 11) until the last and present version, when according to their web page more than 200 companies from areas like healthcare, financial services, software vendors, and more have been studied.

On top of the actual results of an assessment using the framework, other features provided with the model include an annual report, with the analysis of the security initiatives executed by the participating organizations. This feature is very useful, since the organizations that have less mature security initiatives can compare themselves to other more mature companies, to determine specifically what are the areas they should improve in and how, according to what the rest of organizations in their same industry are doing.

Structure:

The structure of this model is composed of 121 total activities that are divided into 12 security practices which are also grouped into 4 major domains.

Similarly to what SAMM presents, there's two greater levels in which activities are distributed, and to guarantee the measurability of the model they also make use of a maturity model approach.

Their approach for the distribution of the activities in their corresponding maturity levels is based on the frequency of observation for each of them during the study of the security initiatives of the collaborating organizations, meaning that activities that are performed with more frequency are going to be in lower maturity levels and the ones with less frequency are going to be in higher levels of maturity, due to less frequent activities being treated as harder to implement.

Governance	Intelligence	SSDL Touchpoints	Deployment
Strategy and Metrics	Attack Models	Architecture Analysis	Penetration Testing
Compliance and Policy	Security Features and Design	Code Review	Software Environment
Training	Standards and Requirements	Security Testing	Configuration Management and Vulnerability Management

Figure 4: BSIMM model [13]

The four main domains as can be seen in Figure 4 above are Governance, Intelligence, SSDL Touchpoints (SSDL is the same as S-SDLC which is the more commonly used abbreviation for Secure Software Development Life Cycle) and Deployment, each of them with three different security practices that hold their corresponding activities.

- **Governance:** This governance domain has no major differences with respect to SAMM, and as I explained in the previous model, includes all practice areas and activities related to the management and organization of security initiatives as well as being in charge of the development and guidance of the staff.
- **Intelligence:** Intelligence includes all the activities aimed at developing a set of artifacts or material with the baseline security knowledge, that is going to be used during the software development process. These artifacts can be things such as attack models, which can be used to determine what threats a product can have in order to think of security by design, instead of approaching security as another layer on top of the software when it is already built. These artifacts also include proactive security guidance for the developers, so that they understand the importance of applying the security processes to their workflow.
- **SSDL (Secure Software Development Lifecycle) Touchpoints:** This domain groups all the practices in charge of the testing and analysis of the software products before their deployment/release. Here we can find processes such as static and dynamic code reviews or even quality assurance testing amongst others.
- **Deployment:** This last domain includes those practices that are involved with traditional network security and software maintenance, after the software is deployed into a real environment. Tasks such as penetration testing, environment configuration or defect/vulnerability management are really important to ensure the product is properly running and vulnerability free during its whole lifespan.

2.2 Model comparison

2.2.1 Type of model

The first thing we can differentiate between the two models is the type of model or the approach that they use to help the users of these frameworks. As I explained in their respective descriptions, BSIMM uses a descriptive model, while SAMM opted for a prescriptive model. In terms of which of the two is best there's no right answer, they have different ideas and purposes when it comes to guiding their users towards establishing future security initiatives. The prescriptive model established by BSIMM is more helpful when trying to compare against something that has already worked in the past, since the activities proposed have been extracted from real organizations and do not extend further than what has already been done. On the other hand the model proposed by SAMM tries to guide the user towards a set of continuous goals and activities, that not only propose what has already been proven to be working, but they strive towards proposing and studying further possibilities that could improve the already existing maturity levels of security.

2.2.2 Measurability approach

Moving on, an area where the two models share a relatively similar approach is in their measurability. Both frameworks are based on maturity models, but even though they both use this idea of maturity models, their approaches are not implemented equally. Starting with SAMM, there's 3 levels of maturity from 1 to 3 and to represent a null maturity in some areas they use the level 0. These levels are given to each stream in the model scheme, so all the activities distributed within the different levels will be connected or lie in the same security practice area. At the same time, this means that all the activities in a certain stream will be directed towards the same process goal, but in lower levels this goal will be achieved with more simple or less mature activities and as we go up in level the complexity of the activities increases, improving also the results and performance of the process. This is why each level is given a general meaning that go as follows:

- **Level 0:** No process is performed in the practice subarea (stream)
- **Level 1:** There's a basic understanding of the process and it is performed in an ad-hoc manner, meaning that it is not yet standardized in any way.
- **Level 2:** Increase efficiency and/or effectiveness of the security practice with the use of standardization or formalization of the process.
- **Level 3:** Mastery of the security practice and further improvements such as automation of some processes.

To see an example of what these maturity levels mean, we can take a look at the Configuration Hardening Stream from the Environment Management security practice. The overall goal of this stream is to set up secure configurations for all the components of the software environment. In the first level of maturity the activity recommended is to identify which are the key components and use standard ready to use configurations to secure them. On the second level, the activity evolves into a more custom configuration that is applied using some guidelines that need to be defined within the organization to meet the exact requirements of their environment. And finally, the third level proposes an automatic monitoring of these configurations with things such as SIEMs, to check if they are complying with the security guidelines.

Note: A *Security Information and Event Management (SIEM)* collects event log data from a range of sources, identifies activity that deviates from the norm with real-time analysis, and takes appropriate action. Essentially, it gives organizations visibility into activity within their network so they can respond swiftly to potential cyberattacks and meet compliance requirements

Now if we take a look at the approach used for BSIMM, they also use 3 maturity levels, but, something important to notice is that their distribution of activities on the different levels of maturity is based solely on the frequency of occurrence, this is based on the thought that less implemented activities are going to be more complex or difficult to implement, hence the lower rate of occurrence. Even though this might be a fairly valid assumption, when talking about secure software development this can be erroneous, since not all companies have the same security requirements and standards. This distribution

approach in some cases will have problems when trying to generalize the model, because if a security activity is very specific for a certain company or industry, it will belong to a high level of maturity even if it is a really simple one. Another thing that I think is not as well thought out as the approach that OWASP used is that the levels themselves are not connected, meaning that activities from lower levels are not necessarily connected or evolve into the ones in top levels due to the way they are distributed, so you can be implementing top level activities without even completing the lower ones in a certain security practice area.

2.2.3 Structural comparison

Structurally they are pretty similar, both have their main functions dividing more or less the areas of management, design, implementation, testing and maintenance. In the approach designed for SAMM we see there are 5 business functions one for each of these areas, meanwhile BSIMM has only four and makes less emphasis on the design and implementation areas by merging them into one that is called Intelligence.

This difference at the top level of their structure continues when we get down to the practices, because SAMM has what they call streams, which essentially add an extra level of granularity compared to BSIMM. On the other hand, while SAMM has about 90 sets of activities, BSIMM in their last iteration presented around 122 activities. In conclusion SAMM has opted for a more generic approach to their activities but with a very thorough distribution, compared to BSIMM that has more activities with less granularity in their division, which might not be as good as an approach when it comes to specific measurability, since in SAMM you can measure down to the stream of activities level, while in BSIMM the maturity measurement is at most calculated for each security practice.

2.2.4 Conclusion

After putting these two models head to head in these different areas, I would say that overall SAMM is going to be the better suited for most cases. This does not mean that the approach used by Synopsis for BSIMM is not good at all, but in my opinion SAMM has more development work behind when it comes to their activities, as they are better distributed and in my opinion better defined for what can be seen, as each of the activities in SAMM not only has the description, it also includes how to evaluate at what maturity degree it is being performed. This fact has an impact on their update periods, since BSIMM is updated on an annual basis and SAMM takes several years to update their frameworks. It is important to mention that this is also a good thing on behalf of BSIMM, since it is possible to have the latest changes faster in the very rapidly changing environment that is software security.

Lastly, I want to mention what to me is the biggest point for BSIMM against SAMM, which is that option to compare against the security efforts of the other partner organizations, which can be very useful, and even though It is not a must have, it can help speed up the planning process behind these initiatives.

Below there is a table that aims at comparing the different main features of each model to put them side by side and have a clear image of what are their characteristics.

	SAMM	BSIMM
Type of model	Prescriptive	Descriptive
Maturity Levels	From 0 to 3 these levels represent different stages of the maturity status of any stream, practice or business function	From 1 to 3 these levels represent whether a certain subset of activities is performed or not
Structure	5 business functions with 3 security practices each (2 complimentary streams of several activities in each practice)	4 domains with 3 practice each and 7-12 activities per practice
Comparison of results	SAMM does not provide a result comparison platform against other organizations at the moment	BSIMM has a database with the results from many organizations that took part in the development of the model and are used to compare against the results of new clients
Availability	SAMM is an open framework so anyone can self-assess their SDLC or pay experts to do so	BSIMM is proprietary to Synopsys. For an assessment you should reach out to them for cost and logistics
Updates	Every 2-3 years	Annual

Table 1: Overall comparison between SAMM and BSIMM

3 Methodology / project development:

In this section I aim to describe a scenario where SAMM or BSIMM could be used, in order to show what would be the process of improving the security efforts in the SDLC.

The description of the scenario I am going to use is as follows:

A consulting company has no security initiatives at all when it comes to developing their own applications and platforms. They want to start implementing security features and processes into their SDLC, and with that they want to create an application that allows their staff to store confidential information about the contracts with their clients securely.

Now, to guide us in this process of securing the SDLC, I will choose one model. In the previous section I compared two very well known models by putting them side by side, and the conclusion I got from it was that SAMM is a better model than BSIMM in most aspects in my opinion. For this reason, I have chosen SAMM to perform this practical section. On top of that, SAMM being an open framework has more detailed information about the processes and activities which will help in their implementation.

Before even starting to think about anything related to building applications, in a real scenario the first step would be to take the test prepared by OWASP to evaluate the different areas of the SDLC according to the guidelines defined in SAMM. This test has a series of questions that will determine in a level from 0 to 3 the maturity of each category. In this practical case, because we assume that nothing regarding security is being considered at all, the levels of each category evaluated which will be starting point I want to improve upon are 0, but in a real scenario, this would provide us with a general baseline maturity that would provide us with a clear picture of the weaknesses and strengths of the SDLC to plan the further improvements.

		Design	No	Yes, some of them	Yes, at least half of them	Yes, most or all of them	Notes	Rating
Application Risk Profile	1	Do you classify applications according to business risk based on a simple and predefined set of questions? An agreed-upon risk classification exists The application team understands the risk classification The risk classification covers critical aspects of business risks the organization is facing The organization has an inventory for the applications in scope						0,00
	2	Do you use centralized and quantified application risk profiles to evaluate business risk? The application risk profile is in line with the organizational risk standard The application risk profile covers impact to security and privacy You validate the quality of the risk profile manually and/or automatically The application risk profiles are stored in a central inventory						
	3	Do you regularly review and update the risk profiles for your applications? The organizational risk standard considers historical feedback to improve the evaluation method Significant changes in the application or business context trigger a review of the relevant risk profiles						
	1	Do you identify and manage architectural design flaws with threat modeling? You perform threat modeling for high-risk applications You use simple threat checklists, such as STRIDE You persist the outcome of a threat model for later use						
	2	Do you use a standard methodology, aligned on your application risk levels? You train your architects, security champions, and other stakeholders on how to do practical threat modeling Your threat modeling methodology includes at least diagramming, threat identification, design flaw mitigations, and how to validate your threat model artifacts Changes in the application or business context trigger a review of the relevant threat models You capture the threat modeling artifacts with tools that are used by your application teams						
	3	Do you regularly review and update the threat modeling methodology for your applications? The threat model methodology considers historical feedback for improvement You regularly (e.g., yearly) review the existing threat models to verify that no new threats are relevant for your applications You automate parts of your threat modeling process with threat modeling tools						

Figure 5: Questionnaire example for the initial assessment using SAMM

In figure 5, you can see the excel sheet that is used to perform the maturity questionnaire, where each category will have different questions with a set of predefined responses. Once

the responses are set, the excel sheet shows the maturity level on a general table shown below in figure 6.

Current Maturity Score					
Functions	Security Practices	Current	Maturity		
			1	2	3
Governance	Strategy & Metrics	0,00	0,00	0,00	0,00
Governance	Policy & Compliance	0,00	0,00	0,00	0,00
Governance	Education & Guidance	0,00	0,00	0,00	0,00
Design	Threat Assessment	0,00	0,00	0,00	0,00
Design	Security Requirements	0,00	0,00	0,00	0,00
Design	Secure Architecture	0,00	0,00	0,00	0,00
Implementation	Secure Build	0,00	0,00	0,00	0,00
Implementation	Secure Deployment	0,00	0,00	0,00	0,00
Implementation	Defect Management	0,00	0,00	0,00	0,00
Verification	Architecture Assessment	0,00	0,00	0,00	0,00
Verification	Requirements Testing	0,00	0,00	0,00	0,00
Verification	Security Testing	0,00	0,00	0,00	0,00
Operations	Incident Management	0,00	0,00	0,00	0,00
Operations	Environment Management	0,00	0,00	0,00	0,00
Operations	Operational Management	0,00	0,00	0,00	0,00

Functions	Current
Governance	0,00
Design	0,00
Implementation	0,00
Verification	0,00
Operations	0,00

Figure 6: SAMM questionnaire scorecard

Now that we know what is our starting point, the goal that I want to set for this section of the thesis is to try and achieve the maturity level 1 in as many areas as I can. Since in this scenario I do not have an actual company to organize, I will be mainly focusing on the activities more closely related to the actual development of the software (specially the more ad-hoc activities that are gathered in this first maturity level generally), aiming to end up with a web application that has been developed using the guidelines from SAMM.

I am going to start off by going through all the security practice areas, to see what are the actions that need to be performed in order to achieve the desired level of maturity. I also need to take into account that for this practical case I will be going through all these areas in order, resembling more or less the steps that are taken in the development of software, from design to maintenance, but in a real scenario, these areas are not thought as a linear process. Instead, the model is actually designed to work with agile methodologies, meaning that all these activities can be performed simultaneously for different projects during the software development life-cycle.

3.1 Governance

The first business function to address is Governance, which has three different practices: *Strategy & Metrics*, *Policy & Compliance* and *Training & Guidance*. This function is probably the least "actionable" one for me in this practical example, because it is the one most related to the organizational area of a company, the policies in place, strategical decisions, etc. But, I will go through all these different practice areas and explain which are the required activities needed to achieve our goal.

3.1.1 Strategy & Metrics

As I explained in section 2.1.1, this practice area focuses on the planning and decision making regarding the security strategy of the organization, as well as the further measurement of success of these decisions. It is a very important area since this security program/plan will be the foundation of the whole security initiative. Metrics are also a very important part of this area because they will guide the organization into making the right decisions and fixing the things that are not helping towards the end security goal (basically what is called a metrics-driven approach to design and modify the security plan).

The two differentiated areas of this security practice make for an easy separation in the two respective streams. In stream A we will focus on the actual definition of objectives and the strategy to achieve them, meanwhile, stream B is dedicated to the measurement and the definition of security metrics that will help improve the security initiatives.

Stream A: Create and promote

To reach the maturity level 1 in this stream, the general criteria is to understand what is the security posture of the organization. This entails gathering information regarding your organization and the software in development or active, to investigate which are the security risks that are being faced by means of interviews with the involved stakeholders or other documentation that may exist about it. Once all the information is gathered, it is meant to be stored in a document where all these risks are displayed in a way that shows their importance based on the companies priorities.

This information is then used to establish the possible risks related to the industry and the organization, determining which would be the impact if the organization is compromised, as well as the opportunities coming from the security features of certain applications (For example if a secure version of an app can help differentiate the company within a market). All this data extracted from the information should be documented in a prioritised manner where we can see what is the organizations most important factors or assets. The document produced from this activity will essentially be used to help create risk profiles for the different applications or software (low, medium or high for example), and based on them different approaches could be applied.

In our example the organization is supposed to be a consulting firm, and the web application is intended to store all the basic data from the different deals they have with their clients, so this information should be categorized as sensitive, therefore is a high risk case, because we do not want any of the information to be leaked. In this case, a more thorough posture is applied, meaning that we will not allow for as many vulnerabilities to arise, specially the more critical ones.

In less sensitive cases maybe having a few non critical vulnerabilities could be allowed to make the product launch faster (obviously as long as these vulnerabilities are not harmful to the user or the company).

In terms of opportunities, the fact of having a very secure infrastructure to store all confidential data helps potential clients to see that in fact the company is trustworthy

and they are invested in their security initiatives, so being able to ensure the confidentiality of their agreements could help potentially to get more clients onboard.

Finally, the processes that should have been performed according to the model are:

- "Capturing the risk appetite of your organization's executive leadership", which refers to what are the risks that the organization is willing to take and which not. A risk could be taken for example if it is marginal or not important, or if the resources of the company are not enough to address them.
- "The organization's leadership vet and approve the set of risks", refers to the approval of the security posture.
- "Identification of the main business and technical threats to your assets and data", referring to the information gathered to build the security posture document.
- "Document risks and store them in an accessible location". These documents with all the information, are then stored in locations where all the relevant personnel such as developers and security engineers can have access to, in order to ensure that they are being taken into account in the actual development of applications.

To evaluate if this is properly done, within the maturity level we can achieve a certain percentage of it, so there are 4 stages in which these criteria could be set to:

- Not done.
- Only performed for general risks.
- Performed with specific risks to the organization.
- Performed for specific risks and opportunities.

Therefore, to achieve the full level of maturity, all the criteria should be met for both organization-specific risks and opportunities.

Stream B: Measure and Improve

This second stream at level 1 has the goal of measuring the effectiveness and efficiency of the security program. To achieve this goal, we have to define a set of metrics or measurements that will provide insight information about the security program, these need to indicate or bring out relevant points on the activities and processes of the SDLC, which together indicate the efficiency and improvement of the security program.

According to the model, the measurements should cover three categories: Effort (i.e. number of applications scanned for vulnerabilities, hours spent fixing vulnerabilities, etc.), Results (i.e. number of vulnerabilities identified, number of fixed problems, etc.) and Environment, that measures the complexity of the work (Contextual metrics such as number of applications existing, total lines of code, etc.).

When defining these metrics we need to focus on several characteristics that are brought up by the model, these being:

- Measuring the values of the metrics consistently to avoid misalignment of the results.

- Easy or inexpensive to gather.
- Expressed in defined units that can be expressed both in cardinal numbers or percentages.
- Together, metrics should help explain the tendencies of the results, for example, maybe the number of vulnerabilities has dropped by a lot but the reason behind it is that less vulnerability scans have been performed.

In conclusion, if we want to achieve the level 1 maturity, the previous guidelines need to be followed in the definition process for all three categories, as well as documenting the set of metrics with all the information needed to understand and use them. Furthermore, the document has to be published so everyone in the organization that needs to know any of the results can.

For the example in our use case, the metrics I am going to use to measure the performance of my software development are the number of vulnerabilities I encountered after coding the first version, the number of hours involved in the coding and the fixed vulnerabilities. Due to obvious reasons, these will not have as much relevance as it would in a real scenario because we are not using the agile methodology nor doing more than one application to compare the improvement of the SDLC, but it will allow me to have a rough estimate of the work that will have gone into the development and the mistakes that I have made. In later stages I will also record the number of tests and type of tests I perform on the application code and endpoint, to evaluate if all possible ends of the software are validated.

3.1.2 Policy & Compliance

In this second practice, the focus is on understanding and meeting external legal and regulatory requirements while driving internal security standards to ensure compliance in a way that is aligned with the business purpose of the organization.

These policies will ensure the proper methodology is used in the development of the applications as well as enabling efficient audits that can be performed via automated controls.

The two streams that we find in this practice are Policy & Standards and Compliance Management. The first stream takes care of the definition and identification of the policies and standards, while the second one, is in charge of setting up or defining which are the compliance requirements for the Software Development Life Cycle (SDLC) processes.

Stream A: Policy & Standards

To achieve the maturity we want, the activity consists of putting together a set of policies and standards that will serve as guidance in all the areas of the software development. To do this, we can focus on already existing industry standards to extract the most relevant sections that align best with the objectives of the organization.

The policies should refer to non-technology-specific aspects, focusing on more general objectives of the organization to achieve security and integrity in the SDLC. Meanwhile,

the standards should incorporate the requirements set by policies and establish more technology-dependent guidelines.

If these defined policies and standards are applied to all the applications in the development life cycle, the level 1 maturity in this stream will be achieved.

For this practical scenario, since this is a more organization oriented activity, I am going to be leaning on standards developed by OWASP, since it is one of the leading organizations in application security projects and standards. Some of the standards I am going to use as reference for some of the later stages include the OWASP Top 10 [17], OWASP Cheat Sheets or the Application Security Verification Standard (ASVS) [18] (which will help guide us in the testing stage of the development).

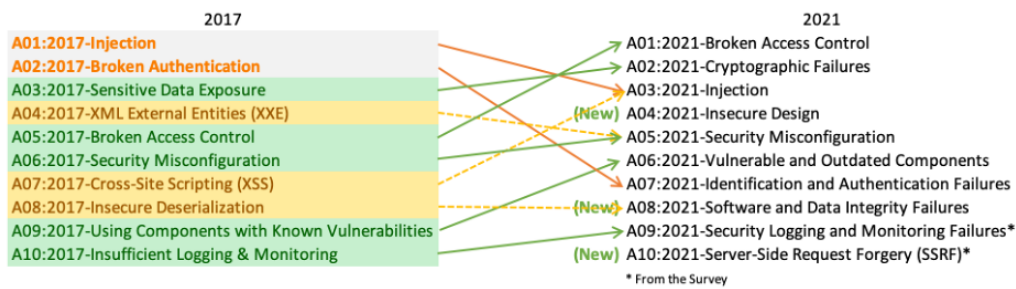


Figure 7: OWASP Top 10 mapping from 2017 to 2021 version [17]

Another standard that I am interested on following is the Common Weakness Enumeration (CWE)[19] Top 25 Most Dangerous Software Errors[20] which compliments well with the Top 10 vulnerabilities list from OWASP.

Note: *CWE is a community-developed list of software and hardware weakness types. It serves as a measuring stick for security tools, and as a baseline for weakness identification, mitigation, and prevention efforts.*

Rank	ID	Name	Score	KEV Count (CVEs)	Rank Change vs. 2021
1	CWE-787	Out-of-bounds Write	64.20	62	0
2	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	45.97	2	0
3	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	22.11	7	+3 ▲
4	CWE-20	Improper Input Validation	20.63	20	0
5	CWE-125	Out-of-bounds Read	17.67	1	-2 ▼
6	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	17.53	32	-1 ▼
7	CWE-416	Use After Free	15.50	28	0
8	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	14.08	19	0
9	CWE-352	Cross-Site Request Forgery (CSRF)	11.53	1	0
10	CWE-434	Unrestricted Upload of File with Dangerous Type	9.56	6	0
11	CWE-476	NULL Pointer Dereference	7.15	0	+4 ▲
12	CWE-502	Deserialization of Untrusted Data	6.68	7	+1 ▲
13	CWE-190	Integer Overflow or Wraparound	6.53	2	-1 ▼
14	CWE-287	Improper Authentication	6.35	4	0
15	CWE-798	Use of Hard-coded Credentials	5.66	0	+1 ▲
16	CWE-862	Missing Authorization	5.53	1	+2 ▲
17	CWE-77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	5.42	5	+8 ▲
18	CWE-306	Missing Authentication for Critical Function	5.15	6	-7 ▼
19	CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer	4.85	6	-2 ▼
20	CWE-276	Incorrect Default Permissions	4.84	0	-1 ▼
21	CWE-918	Server-Side Request Forgery (SSRF)	4.27	8	+3 ▲
22	CWE-362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	3.57	6	+11 ▲
23	CWE-400	Uncontrolled Resource Consumption	3.56	2	+4 ▲
24	CWE-611	Improper Restriction of XML External Entity Reference	3.38	0	-1 ▼
25	CWE-94	Improper Control of Generation of Code ('Code Injection')	3.32	4	+3 ▲

Figure 8: CWE top 25 Most Dangerous Software Errors List [20]

Stream B: Compliance Management

In this second stream, the objective will be to define which are the requirements that need to be met for certain specific applications. The risk type of the application and the data handling will play a part in knowing which are the requirements to comply with. These compliance requirements are going to be mostly related to security best-practices, therefore they might already exist in the list of policies and standards created in the other stream.

To achieve the desired maturity I will use as my policy and standard library the set of requirements from the previously mentioned standards. The idea is that for coding and building the application, I comply with the CWE Top 25 and OWASP Cheat Sheets that provides guidelines focused more on coding, and for testing that the implementation is correctly executed, I will be looking into ASVS which is focused on that.

3.1.3 Education & Guidance

In this practice the main objective is to guide and educate the development teams in secure methodologies, and due to it not being directly involved in the actual process of development I will not conduct any of these activities since I obviously do not have any staff or team to educate.

3.2 Design

In this function we can find activities that will help in the process of defining and preparing the security controls that will need to be implemented in the different software develop-

ments of a company. The three practice areas differentiated in this function are: *Thread Assessment*, *Security Requirements* and *Security Architecture*.

3.2.1 Threat Assessment

In the security practice of Thread Assessment, the aim is to understand the inherent threats of the software we are building and its functionalities, to be aware of how they can affect the organization. This information will help develop security mechanisms during the coding phase, as well as determining which are the specific tests that need to be conducted to validate the security controls put in place to mitigate these threats.

The two streams defined in this security practice are Application Risk Profile and Thread modeling. The first one identifies the risks of every software development project as well as determining their level of importance. This helps to later on automatically perform certain actions depending on the level of risk without having to create development plans for each and every project individually. Meanwhile, Thread Modeling helps with the evaluation and mitigation of threats by defining the security controls and tests that are performed during the different iterations of the design phase and in the later security assessment.

Stream A: Application Risk Profile

The main objective of this stream at level one is to establish a way of classifying the different applications in different risk levels. As mentioned before, this helps automatize the development process, since some actions should be essential for high risk applications but not useful for low risk software. To reach the desired level 1 maturity, the premise or minimum work that has to be done is to have a system that classifies the applications/-software into the different risk levels and then, apply this assessment process for all the applications that are being developed or in development.

As it is mentioned in Software Assurance Maturity Model (SAMM), a basic manner of classifying the risk of certain applications is to create a questionnaire (5-10 questions long) that helps us create a profile that will put the application in one of the three possible levels of risk (low, medium, high).

For the purpose of this example I will create 5 questions that will ultimately determine the level of risk of the application I am coding.

- Question 1: Does the application deal with confidential/sensitive data?

Response: Yes, the app deals with contractual data from the clients of the company.

- Question 2: Is the application open to the public or is it only for internal use?

Response: The app is for internal use, only staff of the company who deals with contracts will have access to it.

- Question 3: Does it have any forms to collect data?

Response: Yes, the app collects data when the user wants to add a new contract to the database.

- Question 4: Are there restricted sections where user management is key?

Response: The application endpoint can be accessed by anyone in the company, but only the ones with a user/password pair granted to them by the administrator can have access to the data and the functionalities.

- Question 5: Does the application use third party libraries that could be subject to vulnerabilities?

Response: To code the application the libraries that are going to be used are mainly the ones derived from the Django application framework from python, but for some functionalities other libraries may need to be used.

As we can see by the responses, we should probably classify this application in the high risk category based on the fact that four out of 5 questions reflected a possible risk or vulnerable functionality. To summarize, the app is critical mainly due to it dealing with information that is confidential, and specially because it also serves as data manager, since data can be entered and deleted from the database. Fortunately, the app is not for open use to the general public and it is expected to be used in a private network that is only accessible from within the company, so it will not be as exposed. But, this is not an excuse to avoid securing the application, since an attacker could enter the network and then try to attack the application.

To finish off, I want to reflect that in a real situation, this list of questions would be longer in order to better understand the risks and functionalities of the software, also, these questions should be aligned with the organization's security objectives to customize and have more information about the aspects that are more important to them.

Stream B: Threat Modeling

In this stream at level 1 maturity, the aim is to identify, evaluate and manage the possible threats that all critical level applications may have. Note that at this lower level of maturity, only critical applications are the ones which need to be evaluated, and it essentially helps us understand what can go wrong in the security of an application and develop mechanisms to fix or avoid these situations.

To simplify the threat modeling process, it is best to use simple checklists such as STRIDE [21] or PASTA [22]. These models help us identify possible threats using different strategies and for this use case, my focus will be on STRIDE since it is the one I know best. In the STRIDE model they identify the following threat categories, that represent the different areas that an application could be vulnerable to:

Threat	Property	Threat Definition
Spoofing Identity	Authentication	Pretending to be someone other than yourself
Tampering with Data	Integrity	Modifying something on disk, network, memory or elsewhere
Repudiation	Non-repudiation	Claiming that you did not do something or were not responsible; can be honest or false

Information Disclosure	Confidentiality	Providing information to someone not authorized to access it
Denial of Service	Availability	Exhausting resources needed to provide service
Elevation of Privilege	Authorization	Allowing someone to do something they are not authorized to do

Table 2: Table with the threat categories of the STRIDE model [23]

The idea for using STRIDE is that in this stage previous to coding the application, we want to make sure that these general threats that our application could face are taken into account during our app architecture design process.

For our use case, all of them could be potential threats, even though DDoS will be less probable since the application is not open to the public internet. Meanwhile, the ones that I expect to be more critical are Information Disclosure, Elevation of Privileges and Spoofing, since the worst that can happen, is that someone logs without authorization and does something with the confidential data that is being used by the application.

During the testing phase, we will come back to this model to test if these threats are actually mitigated or could still lead to vulnerabilities within the web application.

3.2.2 Security Requirements

This security practice as the name says, is in charge of the definition of requirements towards secure software, taking into account the functional objectives of the given application. There is two categories of requirements that will be taken care of, the first type comprises software-related requirements made to protect the core of the application and the data used in the service. On the other hand, we have requirements that are imposed to outside suppliers in case we have software being developed by another organization. Note: the outsourced development does not account for third party libraries (Those will be taken care of later on).

Stream A: Software Requirements

For this first stream, in the maturity level 1 we aim at defining a set of relevant security requirements, that reflect the security objectives towards the application in development (i.e. inputs need to be secured against possible code injections). The requirements should not state how the objective is achieved, only the functional result (i.e. not mention the specific technology). To define them, we can think of the application from the attacker side and think about all the ways the application could be abused.

In our case I am going to define four requirements that I think are essential for the security of our web application.

1. Checking all the input values to make sure XSS or other similar injection vulnerabilities cannot be exploited.

2. Users should only be managed (created/eliminated) by an administrator.
3. The data used in the application has to be secured both in and out of the database.
4. The login should include a limited number of tries to avoid brute force attacks.

These defined requirements need to be measurable, so later on when the development is more advanced, they can be checked easily or even build automated controls in more mature stages, which will allow for a faster and more secure development. To help them being measurable, it is best to be as specific as possible, since it will make it easier to establish a certain control over it because it will not be ambiguous.

An example of a badly defined requirement could be, "The application should protect against the OWASP Top 10", since it is too general, it will not relate to the application as we want. Let's say that one of the Top 10 vulnerabilities is not actually applicable to our use case, then would we be able to "tick" this requirement from the list at all? It is unclear, and that is why simple and specific is the way to go.

Stream B: Supplier Security

In this use case since I am doing all the development of the web application myself, there is no outside suppliers so this section of the model does not apply.

In case we had to, the aim of this section would be to perform assessments of the third party vendors in order to assess their security policies. This is done to establish certain requirements for the third party agreement in order to align the organization's risk "appetite" and meet the requirements in that area. For this we need explain what we expect from them explicit and discuss it clearly.

3.2.3 Security Architecture

This security practice has the objective of tackling all the security regarding the tools and components of the application architecture. The first stream (Architecture Design) deals with the components that are going to be used as a foundation of the application and the second stream (Technology Management) is focused on the technologies that are going to be supporting the development of the application and their security properties (i.e. operating systems or development stacks/tools).

Stream A: Architecture Design

In the first level of this stream, teams are trained to use basic security principles during the design of the software architecture. For this, we must define a set of security principles that will form a short checklist that the technical staff will be able to follow. For this particular situation, I focused on some of the main principles of software developments like **simplicity**, **securing weakest links**, **secure failure** and **least privilege**.

- **Simplicity:** Thinking of simplicity, I decided to use as my base component to develop the application the Django framework for python. Since it is a very straight forward library, that already implements many security controls without requiring

to manually implement them, it will help avoiding mistakes in the implementation of those controls or security functionalities.

- **Securing weakest links:** The weakest link of the application in my opinion are the forms used to input and store the contracts in the database. To ensure that section is secure, I intend to implement sanitation controls for those inputs that look for possible malicious code or formats that are not expected in order to avoid storing them as code in the database and risking a breach.
- **Secure Failure:** Secure failure consists of trying to make a possible failure of the application as secure as possible, so in my case, to avoid the standard error messages of the application where all the info is shown about the files and other app information, I aim to block those messages and instead a generic error should appear to avoid information disclosure. This also applies to the login, where I do not want to specify if the password or username are wrong, instead I will just use the message "the introduced credentials are not correct" or a similar message that reflects a neutral response.
- **Least Privilege:** To ensure that least privilege is taken into account I will only allow admins to create new users in the application. That way we ensure that even if an employee has a company account, unless the admin of the application creates their account into this particular app, they will not be able to access the contracts to modify them.

Stream B: Technology Management

This stream has the aim identifying and evaluating the risk introduced by the important technologies and frameworks used to develop the application. In this regard, the main technologies I use for the development of this web application are, python/Django framework to code the application, GitHub to store the code files and application structure, as well as maintaining a good version control and perform some security tests on top of them, and finally I am gonna be using other applications such as ZAP to perform some security tests in later stages.

In terms of vulnerabilities, the two most sensible components are Django/python and GitHub, since they can be prone to having new vulnerabilities that could directly affect how the application runs. In the case of Django, the risk for vulnerabilities can be more sensitive since the application is running on that platform and any problem with it also affects the application. Meanwhile, GitHub is basically storing the whole code infrastructure of the application, therefore, if anything is leaked it would be easy to know how the app works and prepare custom attacks that may not have been protected against.

3.3 Implementation

After the design phase of the model we come into the Implementation stage. In this phase we will be implementing and building the code of the application following the design and guidelines that we have previously defined. On top of that we will also be taking

care of and managing the defects that this code could have. The three security practices introduced in this domain are *Secure build*, *Secure Deployment* and *Defect Management*.

3.3.1 Secure Build

In this first practice of the Implementation domain the general purpose is to standardize the software build process. This will make the coding process much more repeatable in a consistent manner to avoid possible human errors during the code writing. It obviously must be done using secure components and libraries since we do not want insecure standardized code.

The first of the two streams in this practice focuses on the automation of the build process with the use of automated build pipelines that include security checks along the way to make sure no vulnerabilities appear in the process of building the application. Some checks that can be performed can include SAST that evaluates the code in a static state to detect possible vulnerabilities matching with databases such as CWE, but generally in this process we are looking toward more functional testing, to ensure the application is working so later we can perform more thorough testing. In cases where flags are raised, we can stop the build process and fix the problems before going further into the construction of the application.

Meanwhile, the second stream is in charge of overseeing the security of the software dependencies used in the code of the application. To do this, there are Software Component Analysis (SCA) checks that will report alerts when the security status of a component is affected.

Stream A: Build Process

The goal at this first level of maturity will be to formalize the steps of the build process so that any developer or ultimately an automated tool can follow them and perform the build in a secure manner. Several things we need to take into account when making this build pipeline are, that any tools featured in any of the steps have to be actively maintained to avoid future security issues, and we also need to set up values or checks that allow us to keep track of the integrity of the generated artifacts. Specifically in our use case, this step is not as important since we are just doing a one time development of an application and we will not need to build any more applications due to this not being an actual organization, but, if I had to define a pipeline for the building process of the code for applications using the same frameworks as I used, it would be something like the following.

Since Django uses a pretty straight forward system for the views, I would set up different views and functions that served as baseline for different types of pages (i.e. lists, login, logout, forms). These examples would be examples that should be modified for the task in mind and would already include the security features, therefore the only thing a developer would need is to change the view to match the application in hand.

Once the code is written, it would be pushed into the corresponding test Git branch. In this branch I would activate a pipeline that consisted of several tests, first some unit

testing to evaluate the functional aspects of the application using tools such as Sonarqube, and then some security tests involving SAST and SCA. If any of these tests reported a vulnerability above low risk, I would cancel any further build processes. In case the test came out successful, the next step would be to generate an artifact containing the application itself so that it can be taken into deployment stages later on. A very popular tool to define and execute these pipelines is Jenkins [24], which allows us to integrate different programs or applications to perform the different steps automatically.

Stream B: Software Dependencies

For the second Stream the aim at level one is to gather and keep a record of all the dependencies used in the application in its production environment. To do it, we will create a list with the following information for each dependency:

- Name of the dependency
- Version
- What is it used for?
- Source Information (repository, author, etc.)
- Status

In this case the only dependency we really need to monitor and make sure that it is indeed secure is the Django module from Python, and the information about the version I used is as follows.

Name	Django
Version	4.0.5
What is it used for?	This dependency is the framework used to implement the web application
Source Information (repository, author, etc.)	Repository: https://github.com/django/django Author: Adrian Holovaty, Simon Willison
Status	Secure (All the major security vulnerabilities are addressed by the security features of the dependency)

Table 3: Information about the dependency used to implement the web application

3.3.2 Secure Deployment

This stage of the implementation domain takes care of the security during the deployment of the software/application to ensure that the delivered product is secure and has no integrity breaches after its build.

In the first stream we will be focusing on the deployment process and its formalization, aiming at the removal of any manual human error by trying to automatize it as much as possible. To make sure the deployment is successful a set of expected outcomes and checks are also defined. Meanwhile, in the second stream the focus will be less towards

the processes involved in the deployment, instead, we will be protecting the privacy and integrity of the sensitive data that is required for the application to work in a production environment.

Stream A: Deployment Process

As already mentioned, the aim for this first stream will be to limit the risk introduced by possible human errors when carrying out the processes of the deployment manually. As we did for the Build, in this case we have to define step by step all the processes of the deployment, and break it down into a set of activities or instructions that will follow the person in charge of the deployment.

On top of the guide to deploy the software securely, we also need to list the tools used in the process, since securing them and making sure they are up to date will also be a very important activity, that will be tackled in the Operations domain.

In this use case, the deployment to a production environment is not performed since I do not have access to any real servers where the application can be deployed without paying, but some tools that are very popular for this purpose are Jenkins, Atlassian, AWS (Amazon Web Services), GitLab, Ansible (to automatically create the environments to deploy the application), etc.

Stream B: Secret Management

Following the first stream, one thing that is very delicate when deploying applications into production environments is the management of secrets such as tokens, passwords, hashes, etc. This is very important to guarantee the integrity of the application, and in this first maturity level the objective is to define the access controls to these production secrets.

Some things we need to have in mind when defining these secrets and their access rules is that they should be separated and independent of any secrets used in the testing environments, since these are usually a lot less secure and not as much care goes into securing these secrets. On top of that, all production secrets should be stored in a secure location with the proper access controls (i.e. RBAC), there are several tools that help implement the management and control of secrets, such as Vault which has been developed by a company by the name of HashiCorp [25].

The importance of a proper access control is reflected in the security principle named *minimize the attack surface*, which refers to the fact that if a developer for example has access not only to development environment data but production secrets too, if the account is compromised and the production secrets are accessible, the problem can be much larger than it should. Instead, ideally the developer should not have access to these secrets, to avoid unwanted accesses and manipulations, therefore, applying the least privilege security principle.

Further up in higher levels of maturity these secrets will be automatically generated to avoid possible issues coming from the manual generation of these.

In this particular use case I am tackling, the production environment is out of my scope since I do not have any public server where I can deploy my application or free cloud service. But in terms of secrets, the only ones generated by me are the actual users that can access the application in its test environment, which essentially is a virtual environment in my local computer. Since the admin user is the only one that can manage the users and their passwords because there is no registration functionality, there was no real need for using vault, because the database of the application is the one that manages these.

Vault would be useful if for example we had an API service that for security reasons required a certain key to access the data. In that case, this key could be stored in Vault and accessed directly by the application so it does not have to be hard coded.

3.3.3 Defect Management

This last practice of the implementation domain deals with all the processes related to the collection, analysis of the defects and its management for all the software projects of the organization. These are more often activities that are performed when a version of the application is already in the production environment and a new vulnerability has risen in one of the components for example, even though if any vulnerability or defect is detected in previous steps, it is also dealt with by the team in charge of this practice too.

One thing to be noticed is that this practice does not account for the fix of the vulnerabilities or defects found in applications, this job is dedicated to the developers.

The practice's first stream is focused on the process of handling and managing defects to ensure released software has a given assurance level. On the other hand, the second stream focuses on enriching the information about the defects and deriving metrics to guide decisions about the security of individual projects and of the security assurance program as a whole.

Stream A: Defect Tracking

To achieve the level 1 maturity in this first stream we have various objectives. Firstly we want to have a listing of the possible security defects and the ways to identify them. This will help when performing the right security tests to have a guide of things to test for. On top of that, we also should define a classification of these defects to rank them in terms of risk, which is important to manage these defects once they are reported, because if a higher risk defect is detected, it should be given priority.

To create this list I used the OWASP Top 10 vulnerability list and CWE top 25 most dangerous weaknesses to take the ones that are applicable to this web application and rank them in three levels of risk ranging from low to high with a medium level in the middle.

Security Defect	Description	Detection Method	Risk Level
Cross site request forgery (CSRF)	The web application does not, or can not, sufficiently verify whether a well-formed, valid, consistent request was intentionally provided by the user who submitted the request.	SAST, DAST, Pen. Testing	High
Inadequate Encryption Strength	he software stores or transmits sensitive data using an encryption scheme that is theoretically sound, but is not strong enough for the level of protection required.	Manual Review of the code	High
Improper Authentication	When an actor claims to have a given identity, the software does not prove or insufficiently proves that the claim is correct.	SAST, DAST, Pen. Testing	High
Improper Input Validation	The product receives input or data, but it does not validate or incorrectly validates that the input has the properties that are required to process the data safely and correctly.	Fuzzing techniques, DAST, Pen. Testing	High

Table 4: List of possible defects (some of them) that can affect the application in development

Stream B: Metrics & Feedback

As mentioned, this stream strives towards the improvement of the security initiatives put in place for the SDLC. To do so, periodically several metrics such as the number of tests, the number of reported defects, the number of fixed defects, etc, are evaluated to determine the different areas of improvement and prioritize the application of more security activities in those areas.

Hera are some metrics and the information that can be extracted from them as stated in the [OWASP SAMM model definition for this stream](#):

- **Total number of defects versus total number of verification activities:** This parameter could give us an idea of whether we are performing enough tests to find defects or not. If we see that the total number of defects has decreased, but the frequency is also lower it could mean that we are under-performing in intensity of tests. Meanwhile if the tests frequency stays the same and the defect rise, it could mean two things, we are either making more mistakes or the tests are becoming more accurate and therefore finding new defects that were neglected before, but, in either case we need to review the design and implementation phases.
- **The software components the defects reside in:** This indicates where the defect was detected (i.e. configuration of a deployment tool, application code, etc) and allows us to direct our attentions to the area where security flaws might be

more likely to appear in the future again. Knowing this we can focus our resources towards the improvement of the security in those affected areas.

- **The type or category of the defect:** This metric suggests areas where the development team need further training.
- **Severity of the defect:** This can help us in activities like thread modeling and risk profiling since we can observe the applications and the defects that arise with each, helping the team understand the software's risk exposure and improve the design phase of the development life-cycle.

3.4 Verification

In this stage, the model defines all the processes and activities that are related to the validation and testing of the different artifacts that result from the implementation and build of the application before deploying them into a production environment.

Within this Domain we will see three different security practices named *Architecture Assessment*, *Requirements-driven Testing* and *Security Testing*.

3.4.1 Architecture Assessment

This security practice is in charge of making sure that all the proper security and compliance requirements have been followed when building the application and its infrastructure, as well as testing if the identified security threats are properly mitigated or not.

The two streams differentiated in this practice are *Architecture Validation* and *Architecture Mitigation*. In the first stream the focus is towards the verification of compliance with the requirements that have been set up in the Policy & Compliance and Security Requirements practices. Meanwhile, the second stream is more focused on reviewing the effectiveness of the security controls put in place against the most typical threads as well as the ones identified in the Thread Assessment security practice.

Stream A: Architecture Validation

Overall, the objective for this stream at maturity level one is to have a clear understanding of the infrastructure of the applications and its components as well as the relevant security measures that should be applied.

At first to get a good view of how everything is built, having a schematic model of the overall software architecture, is very helpful when trying to see how the software components interact and what functionalities they have.

Once we know the general structure, we can start to test for some of the general security mechanisms such as authentication, data protection, access management, etc, and observe if they are provisioned or not as defined in the requirements. These mechanisms do not only apply to the core application, but also the components surrounding it that provide services that help the application run, such as databases, and other support elements.

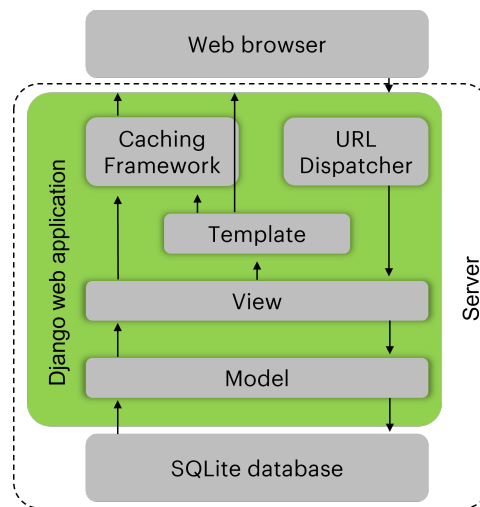


Figure 9: Technical architecture of a Django web application

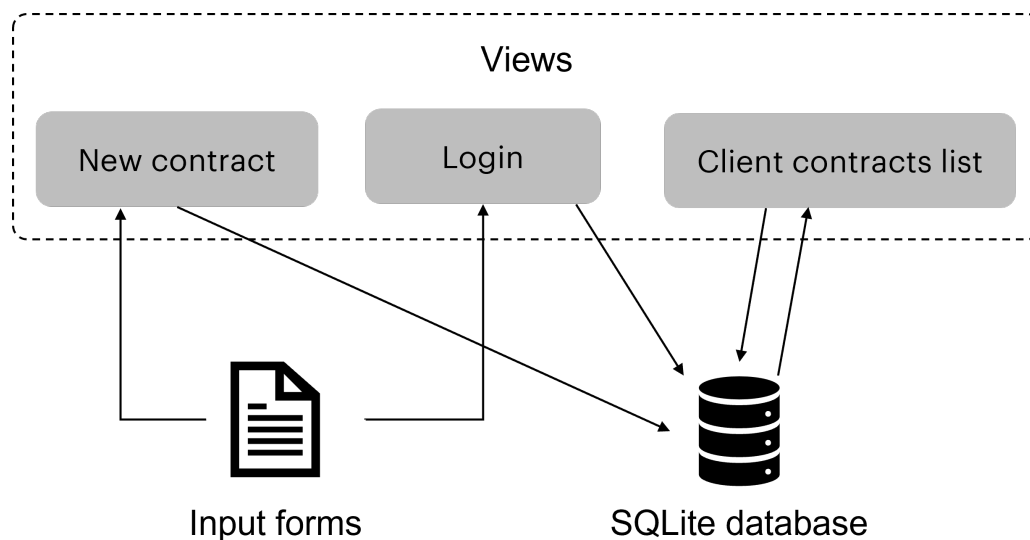


Figure 10: Application functional architecture diagram

In our case we will have the database where users and confidential data is stored, I will also take a look into how the application communicates with it and what are the security mechanisms applied by the Django module to secure this end of the infrastructure. In figure 10 I show how the application interacts with the database and the form templates set up for the different views. The login for example, contains a form used to request access to the application using a set of credentials. This form is protected using the best practices guidelines by Django since it is the main line of defense to gain access to the website. In the new contract view, protecting the form has also been an important feature since we are modifying the database by inserting data, therefore we need to make sure we are secure against any kind of injection attack.

Following the guidelines from OWASP to protect a web application (which I set as compliance standard for this project), I tried to evaluate if the necessary measures have been implemented and their quality.

To put an example of this process, I will explain how I did it for the new contract page, which basically contains the form that will gather the data fields for the new entry in the database. There are 5 fields in this form, the name of the client, the budget, the type of client, the expiration date and a brief description of the contract and the related project. It is obvious that in this situation the most applicable security feature to test is the input validation security, since it is the security feature that can protect the application from injection attacks and other input related attacks. To ensure that only the expected inputs are introduced, in several fields I opted for a closed option input, in which I define the only values that can be introduced and the user selects one of them (i.e. the expiration date is a selection calendar, not written manually or the type of client is either tech or non-tech). In other inputs it was trickier because the description for example cannot be predefined, so here I used character escaping to avoid the insertion of code in the request and only have it as plain text.

Stream B: Architecture Mitigation

In the previous stream the objective was to verify if a certain security mechanism is in place and its quality, but now that we know what are the security controls in place, in this next stream, the objective will be to ensure that the mechanisms in place are indeed protecting against the threats that can affect our application or if there is some part of the architecture that has been left unprotected.

Essentially what I want to achieve in this stream, is to evaluate if the security controls in place are enough or not, and evaluate possible threats that the application architecture is not protected against.

The first thing we will do in this practice is to identify the software architecture model that was previously defined for the particular application. Once we have it, a team of technical staff will analyse the architecture to evaluate if all the threats that could be affecting the application and its environment are being addressed and executed properly. The people in charge of the analysis should be knowledgeable in application security, since they are going to need to identify the possible threats that may have been overlooked by other non-security teams that are offering support such as the architecture or the development teams.

In my case, there is one part that I have actually not taken into account when protecting the application, I have been focusing mainly on the application level, where I can apply the different features provided by Django to protect the application and the communications with the database, but, I have not talked about the communications between the browser and the server on a network level. Some examples of measures that could be applied at this level are the use of a load balancer, a proxy or firewall that protects against attacks such as DDoS, etc.

3.4.2 Requirements-Driven Testing

This practice of the validation domain is focused on evaluating the correct implementation and performance of the security controls that have been put in place by the development team, following the security requirements defined in the *Security Requirements* practice.

In here we will observe what is called negative and positive testing:

- **Positive Testing:** This type of testing will pay attention and verify the compliance of the integrated security controls with the security requirements and validates if they work properly.
- **Negative Testing:** In this other type of testing the aim is not about the correct functioning of the security controls, instead, we are looking to evaluate the quality and performance of the implementation for that control. Essentially we want to find out if the security control has flaws that could lead to a faulty service of the mechanism and therefore cause a security vulnerability/breach.

The positive testing will be addressed in the first stream called *Control Verification*, while negative testing will be performed in the second stream called *Misuse/Abuse Testing*.

Stream A: Control Verification

As mentioned, this stream aims to verify the effectiveness of the security controls integrated into the application, but, for the level one that we are aiming for, we only need to test the standard controls.

The tests that we perform mainly refer to access control, authentication, input validation, encoding, encryption and escaping data (showing the data output securely). To perform these tests we could use manual methods, but using DAST (Dynamic Application Security Testing) tools we can analyze the behaviour of these controls when the application is running in a test environment. A DAST tool will essentially look for vulnerabilities in your running application from the outside without accessing any of the source code.

It is important that these tests are performed periodically to ensure that the application is still secured, but most important when some controls are changed, to make sure the overall control structure of the application has no vulnerabilities that could be potentially exploited.

To perform these tests I used a free open source tool from OWASP called ZAP [26]. This tool as well as doing other types of analysis, also has a DAST functionality that will help us test the behaviour of the standard security controls. The results of these tests will be addressed later on in section 4.

Stream B: Misuse/Abuse Testing

In this second stream we are still trying to evaluate the performance or behaviour of the security controls, but in this case we are going to use what are called fuzzing techniques. Fuzzing in this context essentially means to pass invalid, random or unexpected data to an input within the application. This will allow us to see if the security controls deal

with the use of such data properly or if this unexpected input value can trigger a strange response that might lead to a vulnerable application.

ZAP is also a tool that allows us to perform such testing and for this application I targeted the log in input, since I expect it to be the most dangerous because it is the page where anyone can access. If this layer does not fail the inner application forms should not be as critical in my opinion since the people managing them will already be authenticated users. As in the previous stream, the results of this testing process will also be commented in section 4.

3.4.3 Security Testing

Security testing is also involved with performing tests related to the security controls integrated into the application, but in this case we do not want to just verify if the ones in place are properly working, we also want to detect or find the vulnerabilities that can be found in the application in general, because in the design phase we might have left off other security features that might have a certain importance for the application. Essentially the idea is to double check if the security of the application is complete or has flaws. Mainly we can perform this tests two ways, manually or automatically.

The main benefits of the manual approach is that it will be more in-depth and personalized to the application on hand, but it has its downsides such as being a slow method of testing and requiring an expert in manual security testing. On the other hand, Automated testing is usually more consistent with its results, and even though they might not be as in-depth, the results are also obtained much faster than manually.

In the first stream called *Scalable Baseline*, we will see all the automated related testing, while in the second named *Deep Understanding*, the focus will be towards manual testing and how/when it is performed.

Stream A: Scalable Baseline

The aim of this stream at the first level of maturity is basically to identify or detect the vulnerabilities of the applications that are in both development and production. Since we are still at low maturity levels the evaluation will be basic and will target the main vulnerabilities. To reach this goal, we have to rely on the use of automated tools that perform both Static Application Security Testing (SAST) and Dynamic Application Security Testing (DAST), to evaluate both the source code and the behaviour of the application, which will result in a more efficient security testing process that produces results that are more reliable and of better quality.

In general terms, when performing SAST analysis, we will be looking at the source code of the application and try to find any vulnerabilities that arise from bad practices when coding or the misuse of components. On the other hand, DAST analysis will serve as black-box testing, where we will not be looking or providing any source code to the tool, instead, the tool will perform tests that will try to discover any bugs or unexpected insecure behaviours that the application may have.

Since this use case does not have any budget to acquire any tools that are build for this purpose in a professional environment, I will be using ZAP for the DAST testing, which is an open source tool by OWASP as explained in the previous practice. But, to perform SAST analysis I had to find another tool and ended up deciding to use an open source tool integrated withing GitHub that goes by the name of CodeQL [27]. This tool can be integrated as part of the pipeline when code is pushed into the main branch, meaning that when any piece of code is pushed, an automatic analysis of the code will be performed, according to a set of guidelines that we can program to make a custom test.

Stream B: Deep Understanding

In this stream, similar to the previous one, the aim is to identify the vulnerabilities, but in this case we want to try and dig deeper than what automatic tools can, and discover further vulnerabilities in the most critical components of the application.

Essentially what we want to do is complement the data obtained from testing with automated tools, which are useful but can never replace the full value of a thorough expert manual review. These tests will usually be performed to the components that have been identified as most critical since they will be the most important to secure. Another reason for selective testing is that manual reviews are usually very costly when it comes to time and resources, so being selective in this process will help with the agility of development.

For this testing phase ideally we want to have a set of criteria that is used to review these components, and finally, the results obtained should be addressed in alignment with the defect management policy as well as all the other activities performed in this verification domain.

In this particular use case, since I am not an expert in manual reviews, I decided not to implement this stream, mainly because I think that even if I test these features I think are most critical, the value of the results I could obtain is still similar or less than what the automatic tool ZAP can provide.

3.5 Operations

This last domain is the one that deals with all the security aspects revolving the software in its production state, here we will see activities such as tools and environment hardening, incident management or decommissioning of the applications.

Since this stage requires the application to be in a production stage to perform its activities, I am not going to take it into account for our use-case, because the aim of this section was to build a web application that was secure and ready to be deployed, but I will go through all the practice areas and their streams to put some examples of how these activities could be performed.

The security practices that I am going to go through in this domain are: *Incident Management, Environment Management & Operational Management*.

3.5.1 Incident Management

In this first security practice, the aim is to cover all the activities involved in the detection and response of incidents that can affect the security of our application, to try and maintain the application operative constantly without any problems. Some relevant examples of incidents that are most common are: Denial of Service attacks, that aim to create a bottleneck in the application traffic to reduce or block the service they provide, or injection attacks, that aim at inserting malicious code within the application for other users to execute it whenever they load the application.

In the first stream we will deal with the detection of these incidents, while in the second one we will focus on the actions to be taken as countermeasures, to respond to an incident and try to reduce as much as possible the effects of the incident.

Stream A: Incident Detection

To achieve a level one maturity in this stream, the general objective is to be able to detect the most obvious security incidents. To achieve this goal, we need to focus on capturing as much data regarding the logs of the application and its traffic as possible, this information will help us understand what is the usual workflow, therefore, allowing us to observe any changes from the normal logs that could indicate an incident has happened. This process ideally should be performed using automatic tools to improve the detection time, due to the trouble of analyzing the data manually in big applications with lots of log volume.

Some examples of applications that could be used for this purpose are Azure's monitoring tool or SolarWinds security event manager.

On top of this, one thing to take into account when dealing with logs, is that they should be treated as high priority data, therefore stored in secure locations where only the necessary accesses are granted and following the log data retention guidelines.

Stream B: Incident Response

In this second stream the objective will be to solve the most common security incident as efficiently as possible. The first thing we would need to do is identify who are the stakeholders that are going to be responsible of resolving the different types of incidents. Since we are at a very low level, at first it is not necessary to have a specific team dedicated to this task, but the participants for this activity should be defined beforehand to improve the response time.

Finally, to make the response process repeatable and more analyzable, all actions performed to resolve an incident should be documented, this way we can maintain a certain level of efficiency since incidents of the same origin will be solved much faster knowing how it was resolved in previous situations. This will also help the team understand and learn how to recover from these incidents, as well as improve the efficiency of the solutions applied.

3.5.2 Environment Management

In the Environment Management security practice, the focus is on the maintenance of the elements and components of the operational applications. This is important since regular patches and new versions of these components come out regularly and having the latest most updated versions it is normally the most secure way of acting, since these versions will be the most supported and improved.

Another process that is performed as part of this practice is the secure configuration of the environment elements, as some components might not be secure by default and they are dependant of their proper configuration. On top of that, the teams in charge of this practice should be monitoring the vulnerabilities that may arise in the different versions of these components to make sure the one in use is secure and stable.

The two streams in this practice are called *Configuration Hardening* and *Patching & Updating*. In stream A the hardening of the configurations for the components is performed, while in stream B the focus is on the Patching and updating of the components to the most suitable and secure versions.

Stream A: Configuration Hardening

The aim of this first stream at level one is to have hardened basic configurations for all the components. To achieve this, we would need to apply secure configurations coming from available sources and standardized by vendors. These sources will serve as secure guidelines that will be followed during the configuration definition.

The process to define these configurations will start with the identification of the different stack components and looking for the guidelines (such as OWASP's open project) that will be used to define a secure baseline configuration. With these components we will make a list where all the version info on the components is going to be stored, in my case these would be the Django framework version 4.0.5, and the rest of the components that will be used to run the application like the operating system, which in my case is MacOS 12.0 Monterey or the container tool we might be using to deploy the application (i.e. kubernetes), and even the IDE we use to run the code could be added to the list. As components we could also look at the servers, but they included as part of the Django framework.

In further steps we would use the guidelines and our own knowledge and experience to build a set of custom security configurations for each component, but since we are aiming for level one, we only want to configure these components securely in an ad-hoc manner without standardizing anything or storing any formal configuration process.

Stream B: Patching & Updating

Complementing the first stream, this second one aims at mitigating all well-known issues that can affect third-party components such as operating systems, code libraries, etc. The activities related with this stream are performed in an ad-hoc manner, meaning that they do not need to be detailed and documented processes, instead they are performed for each component stack separately without centralizing the work.

The development teams are the ones who choose when the components should be updated or patched, except in situations where a vulnerability has been published for a certain component and it causes a security issue (In these cases it is required to act on it and patch it or update to another more secure version).

Finally, periodical reviews of vulnerabilities should be performed in order to stay updated on what is going on in the industry and be able to react fast to any problems that may arise in this regard.

3.5.3 Operational Management

In this last security practice of the whole model, the focus revolves around the protection of operational support tasks that involve components such as databases, operating systems and the data itself. These are tasks that are not performed by the application, but that are key for the security of the application, because using old operating systems, or storing the data insecurely in a database could be detrimental for the application's security even if the application itself is secure.

Some of these tasks could be for example, administration of applications and decommissioning, database administration or data backups between many others.

The two streams I am about to talk about are called *Data Protection* and *System Decommissioning/Legacy Management* and as their name suggest, in stream A the aim will be towards functions protecting data and stream B will go towards the management and decommissioning of applications and systems.

Stream A: Data Protection

The objective in stream A is to understand the sensitivity of the data used throughout the organization as well as the application of some quick measures to secure it.

The first step would be to figure out what are the different levels of sensitivity that the data is associated to in your organization, to do this, we can gather information from all around the organization in case this classification is performed in an ad-hoc manner for each application. Once this is set, the data within each application should be protected using the proper guidelines, according to the highest sensitivity level, without worrying about the lower levels to simplify the protection process.

Remembering from earlier security activities, we defined three levels of risk (low, medium, high) and for this we will follow the same approach for sensitivity, where our data in this use case would correlate to the high sensitivity level.

To secure the data, we could follow standard regulations and guidelines such as the General Data Protection Regulation (GDPR), which for example in Article 32 mentions that personal data should be masked or encrypted when it is processed in any way [28].

Stream B: System Decommissioning / Legacy Management

In this last Stream the aim of the processes performed is to identify the unused software components, to do this at low maturity levels the identification process can be just a

periodical review of the assets or even observations by chance, but when a component out of use is found, it should be noted and a formal process for its decommissioning should be filed, making sure that the development teams are really not using it anymore and are informed that it should be securely decommissioned.

Following this identification, we also need to take into account that in this lower maturity levels we may have many people across the organization using different versions for some components, and it is also the aim of this stream to discontinue or remove the older versions of some of these components that might not be secure enough, and manage the migration to more updated versions.

This stream is obviously not gonna be very useful in our use-case since this is a one time development that will not be released publicly and this is more focused towards a continued development environment such as the one in a real case scenario, where more time will pass and the software used initially may get to the point of being outdated or obsolete.

4 Results

In this section of the thesis I want to explain in depth how the web application turned out, some of the security controls I think are most important in the implementation, and the results of security validation testing stage as well as the actions I took as consequence of those security tests.

If you want to have a look at the code of the application before reading this next section, it is accessible in the following git repository: <https://github.com/enric-carrera/WEBTFMgit>

4.1 Web Application

I will start with the presentation of the Web Application, its functionalities and the different pages it has.

First of all, as I have explained during the design phase of the model in the beginning of section 3, the scenario or objective of the application was to fill in the need of a hypothetical consulting company to have a place for managing and storing the data of the contracts they had with their clients about specific projects.

To fill in this gap, the description of the application that I used as baseline for the design was: *Simple website that could only be accessed by certain people and that would have a place to add and remove contracts from a list.*

Since the application had to be simple, I decided to define only 3 main pages including the login, the contract list display, and the new entry form page.

Login page:

This page its pretty self explanatory, but I do want to mention that Django was very helpful when creating it, since the main functionalities and methods needed to operate this functionality were already set up. The only thing I really had to do in this scenario was to create the endpoint and the HTML page with the form, to link them directly to what they call the registration module or sub-application, that takes care already of all the data management and access control. This was helpful since I did not have to bother in adding any extra security features as the module already had them from scratch.



Figure 11: Web Application login page screenshot

Client/Project list page:

This page as I mentioned before, had the objective of displaying a listing of all the contracts in the database to be able to see their basic information and remove any contract if necessary. Here, I also added a sub-view to observe the more in depth data about a specific contract, since the list would not be very clean looking and clear if all the data was listed in the main page.

Since this sub-view was not modifying the database or had any important functionality, the only feature that I had to address was how to make the query to the database, to make sure that if a malicious entry was leaked into the list of contracts, it would not be able to execute any actions on the database (essentially how to correctly retrieve data from the database and show it without risk of code execution).

On the main list, I also had to make sure the data retrieval and display was performed correctly, and even though this main list could modify the database because contracts could be deleted from here, I could not really do much about it, because if an account is compromised there is no way of knowing for sure if that person is who he or she says without complicating the functional features of the application. One option that could have been implemented if there had been time and resources is a two factor authentication, to ensure that the person deleting the contract is actually them, since they would need some other authentication method in case that the account is compromised (i.e. an SMS with a code).

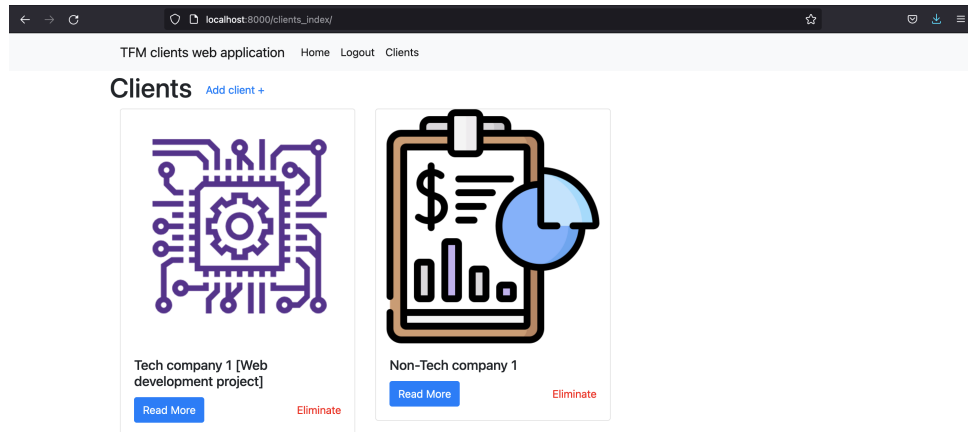


Figure 12: Web Application contract list page screenshot

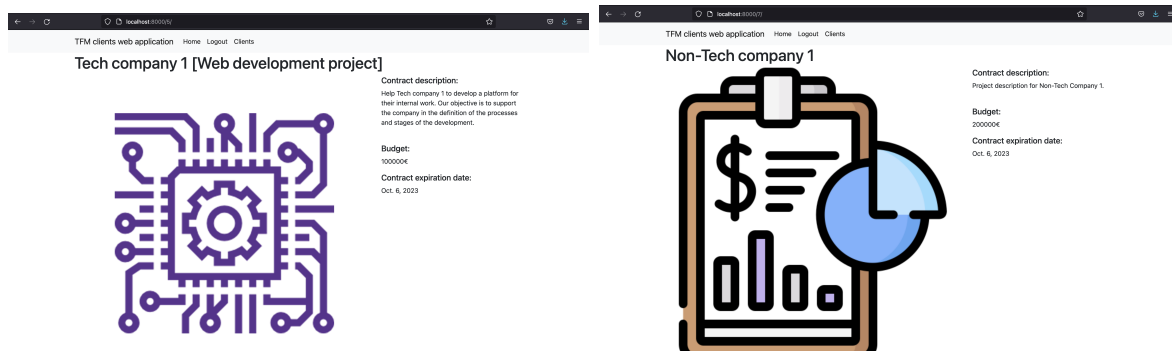


Figure 13: Web Application detailed contract page screenshots

New entry form page:

This page is the one that contains the form to add a new contract to the list. In this case I did really need to think about the security features more thoroughly, since we are going to be adding information to the database, which is a very sensitive action.

In section 3.4.1 I explained in a simpler way what are the different data fields involved in one contract entry, but I am going to try and go a bit deeper here.

- **Client name:** This is a CharField of a maximum length of 100 characters that represents the name of the company that requested the project. I limited the length of this entry following the security principle of minimization, which explains that the size and complexity of the data we want to protect should be the minimum possible.
- **Description of the project:** This is a TextField without a defined maximum length, since there is no way of knowing how much space it will take to describe different projects. In this case, to protect against possible injection attacks, character escaping was necessary, this way we can protect against Cross Site Scripting (XSS) for example.
- **Budget:** This is an IntegerField that represents the available money to spend in the project. In this case, to protect the field I opted for only accepting Integer values, therefore it will be very hard to inject any code through it because it only allows numbers to be introduced.
- **Expiration date:** This is a DateField that represents the predicted expiration date of the project. For this field, I used a similar method as with the budget, to minimize the type of entries that can be made, I limited the values by using a calendar interface where you can only choose a value from the list of dates.
- **Type of contract:** This field was created to differentiate between tech-related projects and non-technological projects, to add an image that helps identifying them. Since there is only 2 types, I limited the field to two options to choose from, that

way I avoid having to protect it in other ways since I am dictating what are the exact entries available.

This field essentially stores depending on the option the locations of the two images linked to each type. Since the images are in a static folder within the application because there was no server to store them dynamically, when the web application is not in debug mode the images will not be accessible by the application for security reasons, but in debug mode they will show.

On top of the security features I explained previously, following the OWASP cheat sheet to protect against Cross Site Request Forgery (CSRF) [29] I used the CSRF Token, which makes it impossible for attackers to create valid requests to the back-end server, since the server will check if there is a token and its validity, but the attacker does not have this value.

On top of it, other cheat sheets from OWASP that I looked at for this section included, the Cross Site Scripting (XSS) related one [30] and the one related to Injection [31]. Finally, following the documentation for the Django framework I also used their documents about forms and input validation [32].

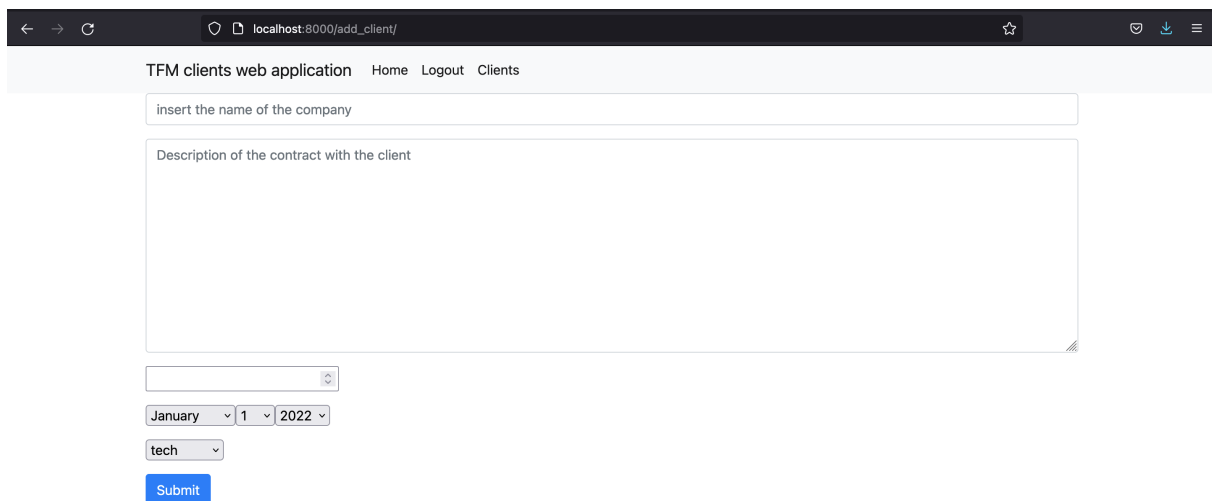


Figure 14: Web Application new contract entry form page screenshot

4.2 Validation and Testing of the Application

Once the application was built, to perform the security verification/test I used mainly two different tools. As mentioned previously in other sections, I opted for an automatic approach to test the security of my application, to simplify the process as well as to make sure that every area of the application was covered and properly tested, since manual testing could produce misleading results in my case due to lack of deep knowledge.

The two tools I used were ZAP (developed and maintained as an open project from OWASP) and CodeQL, which is a tool integrated into the GitHub Security Actions pipeline.

- **ZAP:** To perform the Dynamic Application Security Testing (DAST) I chose this tool mainly because it is an open source tool, and had the ability to customize the tests with some functionalities such as fuzzing testing for inputs. In a real scenario where there is a budget to acquire licenses for other more professional tools, I could have used for example programs such as BurpSuite, Veracode or Fortify [33].
- **CodeQL:** This second tool is focused on Static Application Security Testing (SAST), so we will be checking the actual code for any malformed piece of code or missing security features. The reason for choosing this tool is the same as ZAP, as they are both open source tools that do not require to pay for a license. This second tool comes as a feature within the GitHub security capabilities, where we can set up a pipeline with different tools that perform actions when something happens. In this case, I set it up in a way that every time a piece of code is pushed into the main branch, an analysis of the repository is executed and I can later on see the results in the security tab of the repository.

CodeQL analysis results:

I will Start off with the analysis I did using CodeQL since it reported no signs of vulnerabilities when I executed the test. To perform the test, the main configuration was as follows:

```
9   on:
10     push:
11       branches: [ "main" ]
12     pull_request:
13       # The branches below must be a subset of the branches above
14       branches: [ "main" ]
15     workflow_dispatch:
```

Figure 15: Trigger configuration for CodeQL

In figure 15 we can see the piece of code that defines when these tests need to be performed, in this case, as I said, it is set so that the test runs when either the code is pushed or a merge request is made. On top of that I added one last feature that allows me to run the test manually whenever I want from the web application interface.

```
24     security-events: write
25
26     strategy:
27       fail-fast: false
28       matrix:
29         language: [ 'javascript', 'python' ]
30
```

Figure 16: Programming language configuration for CodeQL

In figure 16 we can see another part of the configuration that is essential. This list of programming languages determines for which code language the test is going to be performed, in this case the test will be performed for Python and JavaScript, as they are the ones used in the project. HTML is not there because it is not one of the languages accepted by the tool.

The final result of this analysis can be seen in the following image, which shows that the alerts reported by the analysis tools are 0, meaning that no vulnerabilities were found.

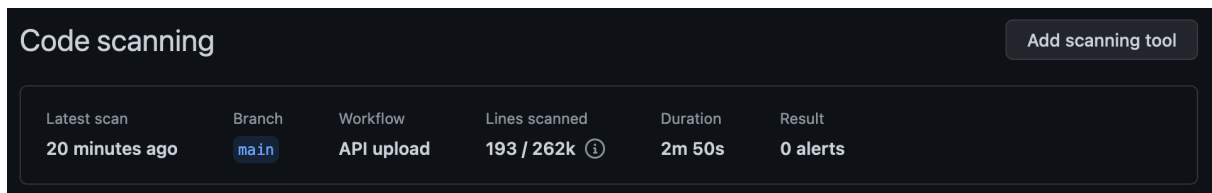


Figure 17: Results of the Static Application Security Testing (SAST) performed with codeQL

Following these results, I am now going to explain the results I obtained when performing the Dynamic Application Security Testing (DAST) using ZAP. In this case I did find some vulnerabilities that I will show now.

ZAP analysis results:

The usage of ZAP did not require much configuration since it is already set up for basic testing, and in our use case to reach level 1 maturity in most validation practices we only addressed the main security issues. To perform the test, I set up the tool to test against all the possible endpoints of the application as can be seen at the top section of the following image, and below there is another sector with the result I obtained. In figure 18 you can see that in this test 3 security issues were raised.

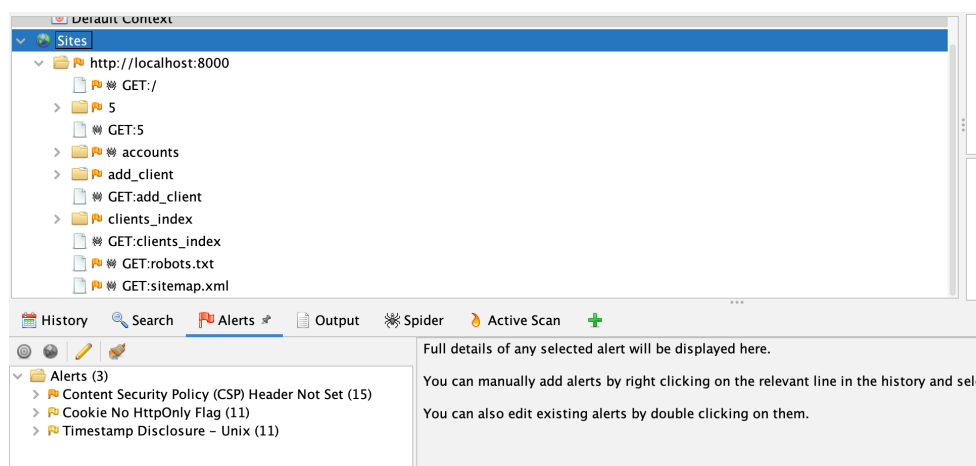


Figure 18: Results of the Dynamic Application Security Testing (DAST) performed with ZAP

Missing Content Security Policy (CSP) Header:

When coding the web application I did not properly set up the CSP Header. This header is a specification from the standardization organization World Wide Web Consortium (W3C), that offers the possibility to instruct the client browser from which location and/or which type of resources are allowed to be loaded when the different pages of the web application are accessed. To define a loading behavior, the CSP specification uses “directives” where a directive defines a loading behavior for a target resource type.

This vulnerability is identified in the medium risk level range, so it is fairly important to fix this and properly configure this header. In figure 19 you can see the detailed information that ZAP gives about this detected vulnerability.

Content Security Policy (CSP) Header Not Set	
URL:	http://localhost:8000
Risk:	Medium
Confidence:	High
Parameter:	
Attack:	
Evidence:	
CWE ID:	693
WASC ID:	15
Source:	Passive (10038 – Content Security Policy (CSP) Header Not Set)
Description: Content Security Policy (CSP) is an added layer of security that helps to detect and mitigate certain types of attacks, including Cross Site Scripting (XSS) and data injection attacks. These attacks are used for everything from data theft to site defacement or distribution of malware. CSP provides a set of standard HTTP headers that allow website owners to declare approved sources of content that browsers should be allowed to load on that page — covered types are	
Other Info:	
Solution: Ensure that your web server, application server, load balancer, etc. is configured to set the Content-Security-Policy header, to achieve optimal browser support: "Content-Security-Policy" for Chrome 25+, Firefox 23+ and Safari 7+, "X-Content-Security-Policy" for Firefox 4.0+ and Internet Explorer 10+, and "X-WebKit-CSP" for Chrome 14+ and Safari 6+.	
Reference: https://developer.mozilla.org/en-US/docs/Web/Security/CSP/Introducing_Content_Security_Policy https://cheatsheetseries.owasp.org/cheatsheets/Content_Security_Policy_Cheat_Sheet.html http://www.w3.org/TR/CSP/	

Figure 19: Detailed information on the Missing CSP header vulnerability

To integrate this into the application, following Django’s documentation [34] I added the CSP middleware to the list of components for the application, which allowed me to configure the directives from the main configuration file of the web application, as can be observed in the following images.

```

MIDDLEWARE = [
    'csp.middleware.CSPMiddleware',
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

```

Figure 20: List of middleware components located in the main configuration file for the web application

```
# Content Security Policy

CSP_IMG_SRC = ('self','https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css','https:
CSP_STYLE_SRC = ('self','https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css','http
CSP_SCRIPT_SRC = ('self','https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css','htt
```

Figure 21: Directives configuration for the CSP middleware component

In figure 21 you can see how I defined the directives. Apart from using the "self" directive which only allows the browser to load elements coming from the domain of the actual web application, I had to add several extra domains since I used some premade styles and scripts that helped with the visual part of the web application. If I did not add these domains to the directives, those style sheets and files coming from a different domain would not have been loaded, as can be observed in figure 22 where there are four elements that where stopped from loading.

Status	Method	Domain	File	Initiator	Type	Transferred	Size
200	GET	localhost:8000	/5/	document	html	2.72 KB (raced)	2...
404	GET	stackpath.bootstrapcdn.com	bootstrap.min.css	stylesheet	CSP		
404	GET	code.jquery.com	jquery-3.3.1.slim.min.js	script	CSP		
404	GET	cdnjs.cloudflare.com	popper.min.js	script	CSP		
404	GET	stackpath.bootstrapcdn.com	bootstrap.min.js	script	CSP		
200	GET	localhost:8000	company1.png	img	png	cached	2...
404	GET	localhost:8000	favicon.ico	FaviconLoader.jsm...	html	cached	3 ...

Figure 22: Blocked components by the CSP header

Due to configuring the directives with this domains as exceptions, it means that we are trusting them for the security of our application, and that is why in the next analysis I did (to see if the CSP vulnerability was actually fixed this time) reported a new CSP alert. In this case referring to these exceptions I configured (figure 23 and figure 24). Since I trust this sources, for my use case this was actually a false alarm and I did not have to perform any actions to fix it.

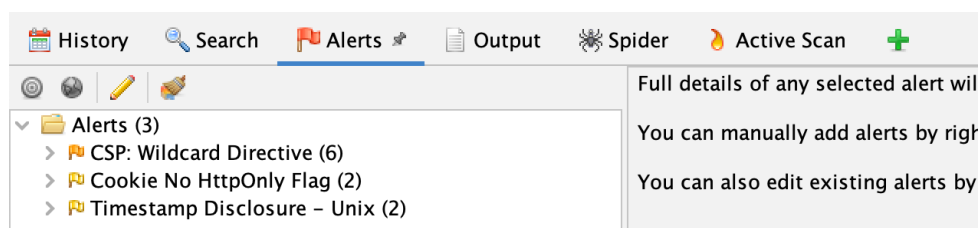



Figure 23: New alerts reported in the DAST analysis

CSP: Wildcard Directive

URL: http://localhost:8000/
 Risk:  Medium
 Confidence: High
 Parameter: Content-Security-Policy
 Attack:

Evidence: style-src 'self' https://stackpath.bootstrapcdn.com https://code.jquery.com https://cdnjs.cloudflare.com https://stackpath.bootstrapcdn.com; script-src 'self' https://stackpath.bootstrapcdn.com https://code.jquery.com https://cdnjs.cloudflare.com https://stackpath.bootstrapcdn.com; default-src 'self'; img-src 'self' https://stackpath.bootstrapcdn.com https://code.jquery.com https://cdnjs.cloudflare.com https://stackpath.bootstrapcdn.com

CWE ID: 693
 WASC ID: 15
 Source: Passive (10055 - CSP)

Description:
 Content Security Policy (CSP) is an added layer of security that helps to detect and mitigate certain types of attacks. Including (but not limited to) Cross Site Scripting (XSS), and data injection attacks. These attacks are used for everything from data theft to site defacement or distribution of malware. CSP provides a set of standard HTTP headers that allow website owners to declare approved sources of content that browsers should be allowed to load on that page —

Other Info:
 The following directives either allow wildcard sources (or ancestors), are not defined, or are overly broadly defined:
 frame-ancestors, form-action


Solution:
 Ensure that your web server, application server, load balancer, etc. is properly configured to set the Content-Security-Policy header.

Figure 24: Detailed information on the CSP wildcard directive vulnerability

HTTP Only Header:

This Vulnerability was detected by ZAP because it could find that the cookie where the CSRF token was stored was accessible using JavaScript code. This is very dangerous since data from the session could be obtained if malicious JavaScript code is introduced into the web application, which should not happen thanks to other security controls, but it is better to make sure that this cookie is protected.

Cookie No HttpOnly Flag

URL: http://localhost:8000/accounts/login/
 Risk:  Low
 Confidence: Medium
 Parameter: csrftoken
 Attack:

Evidence: Set-Cookie: csrftoken
 CWE ID: 1004
 WASC ID: 13
 Source: Passive (10010 - Cookie No HttpOnly Flag)

Description:
 A cookie has been set without the HttpOnly flag, which means that the cookie can be accessed by JavaScript. If a malicious script can be run on this page then the cookie will be accessible and can be transmitted to another site. If this is a session cookie then session hijacking may be possible.

Other Info:

Solution:
 Ensure that the HttpOnly flag is set for all cookies.

Reference:
<https://owasp.org/www-community/HttpOnly>

Figure 25: Detailed information on the no HttpOnly vulnerability

To fix this vulnerability, I simply added the line shown in figure 26 to the main configuration file of the web application, which meant that the HttpOnly header was activated for the CSRF Token cookie.

```
CSRF_COOKIE_HTTPONLY = True
```

Figure 26: HttpOnly Header configuration

Once I fixed the issue, we can observe now that the result of testing again with ZAP did not report any alerts regarding this vulnerability anymore.

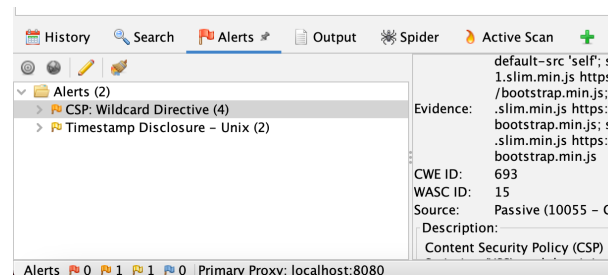


Figure 27: Results of the analysis with ZAP after fixing the HttpOnly header vulnerability

Timestamp Disclosure: This last alert was reported because the analysis uncovered that the server was disclosing a timestamp in the header section of its responses. This could be a problem if the data which the timestamp is referring to can be aggregated to disclose exploitable patterns. But since I could not find a solution for this problem and the vulnerability is classified as low risk, I decided to leave it as it is and carry on with the project.

<p>Timestamp Disclosure - Unix</p> <p>URL: http://localhost:8000/accounts/login/</p> <p>Risk: Low</p> <p>Confidence: Low</p> <p>Parameter:</p> <p>Attack:</p> <p>Evidence: 31449600</p> <p>CWE ID: 200</p> <p>WASC ID: 13</p> <p>Source: Passive (10096 - Timestamp Disclosure)</p> <p>Description:</p> <p>A timestamp was disclosed by the application/web server - Unix</p> <p>Other Info:</p> <p>31449600, which evaluates to: 1970-12-31 01:00:00</p> <p>Solution:</p> <p>Manually confirm that the timestamp data is not sensitive, and that the data cannot be aggregated to disclose exploitable patterns.</p> <p>Reference:</p> <p>http://projects.webappsec.org/w/page/13246936/Information%20Leakage</p>

Figure 28: Detailed information on the timestamp disclosure vulnerability

5 Budget

In this section, the costs of developing this project are presented and explained. To finish it, I worked around 10 to 20h/week and I had a teacher advisor that worked around 1h/week, and the total duration of the project was around 6 months from March to September approximately. If this was a real project, the budget to pay the stakeholders involved would be as follows more or less, taking into account that I would be categorized as a Junior Engineer receiving around 11€/hour.

Table 5: Staff budget

	# people	salary/hour	Hours	Salary	Taxes(30%)	Subtotal
Student(Junior Engineer)	1	11	360	3 960	1 188	5 148 €
Teacher Advisor(UPC)	1	50	24	1 200	360	1 560 €
TOTAL						6 708 €

When it comes to tools used, regarding hardware, I had my own computer with a cost of around 1500 € and an expected life of around 4 to 5 years as presented in Table 6. When it comes to software, everything used to develop the application was free, and no costs were added on that end. Some of the tools include, Visual studio (IDE), GitHub, Django framework, OWASP SAMM framework and ZAP for testing purposes.

Table 6: Hardware budget

Computers initial purchase value	1 500 €
Residual value (25%)	375 €
Equipment life expectancy	5 years (60 months)
Equipment amortization (monthly)	6.25 €
TOTAL amortization (5 months)	31.25 €

6 Conclusions and future development:

In this section I aim to do a recap on some of the most important things I have explained and done in this masters thesis and I will try to give some personal conclusions on some of the topics.

6.1 Model Comparison:

regarding the model comparison between BSIMM and SAMM, In my opinion if I had to choose between SAMM or BSIMM, the clear winner would be SAMM. to me, it makes much more sense the way they distributed the activities and how maturity levels are calculated within each stream, to make sure that the further the level is, the more proficient the organization is in a more specific practice, contrary to BSIMM where the model is not as granulated and activities with different objectives that lie within the same practice are mixed for the measurement of the maturity level.

Nevertheless, BSIMM has good features too, like the database/pool of information, regarding the security initiatives of many companies that have been involved in the definition of the model, which is very helpful when trying to compare the results with other organizations of the same industry as yours. In fact, it is something that SAMM has started to come up with too, as they have made available an open project to gather assessment data from different organizations that use their model, in order to be able to establish a comparison platform too.

On top of this, by working and going in more detail through all the practices of the SAMM, in my opinion some improvements or changes could be made in them framework. There are two main changes I think could be applied, the first is related to the validation business function, and the second one is related to the governance business function.

6.1.1 Validation

In my opinion, the way they have separated the different practices within the business function is a bit confusing. When I was reading them and trying to understand each one, I felt like they were over-complicating the testing phase and repeating the same processes with slight changes of objective between them.

In general there are 4 main areas of testing: Code testing, Behaviour Testing, Component Testing and Environment/Configuration Testing. From my point of view, a much better approach would be to divide the validation business function depending on the types of test I just mentioned, in such way that Code Testing is referring to Static Application Security Testing (SAST) (or testing from the inside, since we are reviewing the code and internals of the application), in Behaviour Testing we would be addressing Dynamic Application Security Testing (DAST) (or testing from the outside, since we will test how the software behaves against a set of actions) and other manual testing such as penetration tests, in Component testing the focus would be on Software Component Analysis (SCA) and finally in environment testing we would talk about all the reviews regarding the environment components and tools that are going to maintain the application operative (i.e. Configuration file review for docker containers or image testing)

The problem in my opinion comes from the fact that several practices and activities within the validation domain can be performed with the same tool. One example of this was the ZAP tool I used for DAST, because at first, it detected that I did not have the CSP header in place, therefore the security control was missing, but later when I added the control, it also detected that a possible faulty configuration was present. This essentially covered various of our practices, since it was used to evaluate if a security control was in place, but also to observe the quality of it and to verify if there were more vulnerabilities that were not accounted for. This is the reason why in my opinion it is more suitable to use the types of testing as the different practices, since we will be able to focus more on each test and the level of complexity that it is performed with, in order to evaluate the maturity of the initiatives in a more meaningful way.

6.1.2 Governance

In this regard, It is not as big of a change as the previous, but I think that having security training as a practice within governance does not fit the overall objective of the business function. Instead, I think it should be a separate business function that has its own practices such as evaluation of security knowledge and planning for security awareness. The reason for this is that governance is more focused on the planning of the SDLC and the processes involved in the development of the software and training is one of those processes/domains in my opinion that should be treated with the same intent as any other business function, since having a development team that knows how to do things properly is essential, because if they are not able to follow the guidelines that are set up in terms of security, the security controls will not be properly integrated and the security of the web application will be always compromised.

6.2 Practical work:

When it comes to the development of the web application, I accomplished the objective set in the beginning of the thesis, where I aimed at developing a web application that did not report any critical vulnerabilities when tested by security review tools. Initially we had some vulnerabilities, specially one of medium risk level that was successfully fixed. Even though I successfully built the web application, there are lots of improvements that could be made to both the security functionalities and the methodologies applied to develop it.

When addressing the initial objectives for the use case, I opted to only apply the first level of maturity across the processes of the whole development, which tends to be the one that is most ad-hoc in most practices, therefore the one most adapted for this practical example. But, the model can be much more complex than what has been presented.

In real environments, specially in organizations that develop software as one of their main activities, more mature initiatives would be expected, therefore a level 1 would not be enough as we want to start integrating more automatic features that standardize the development process and provide more consistent results, using tools such as Jenkins [24] (for pipeline definition and execution) or more complex testing tools such as Fortify [33] for DAST and SAST. Another tool that would be more interesting in less ad-hoc situations

is JIRA [35], which works as a ticketing service that can be used as defect management tool, to report the defects and send them to the responsible person in charge of fixing it.

Even though the Software Assurance Maturity Model (SAMM) allows this next level of maturity, for the use case of this project it did not make much sense to try and perform some of those next level activities, since I was only developing one web application, and automating many of the processes would have required much more work that would not have been useful, because I will not be making more web applications following this one or continue its development in any way.

This goes to show that models like these are not really made for situations where we just want to develop one application, instead they are very useful in the use cases I just mentioned, when organizations need a way of producing better, faster and more secure software in a continuous manner.

References

- [1] Atlassian, “What is devops?” [Online]. Available: <https://www.atlassian.com/devops/what-is-devops>
- [2] —, “Atlassian’s no-nonsense guide to agile development.” [Online]. Available: <https://www.atlassian.com/agile>
- [3] Microsoft, “What is devops?” [Online]. Available: <https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-devops/#devops-overview>
- [4] A. C. C. [IBM], “A complete guide to devops.” [Online]. Available: <https://www.ibm.com/cloud/learn/devops-a-complete-guide>
- [5] GitGuardian, “Devsecops, protecting the modern software factory.” [Online]. Available: https://res.cloudinary.com/da8kiytlc/image/upload/GitGuardian.WhitePaper.DevSecOps_2022.pdf?utm_source=Social&utm_campaign=OWASP&utm_medium=OWASP_Post
- [6] Atlassian, “Devsecops tools.” [Online]. Available: <https://www.atlassian.com/devops/devops-tools/devsecops-tools>
- [7] R. Hat, “What is devsecops?” [Online]. Available: <https://www.redhat.com/en/topics/devops/what-is-devsecops>
- [8] T. A. N. [Acunetix], “Devsecops vs. secdevops.” [Online]. Available: <https://www.acunetix.com/blog/web-security-zone/devsecops-vs-secdevops/>
- [9] A. C. C. D. Defense], “What is secdevops and why is it so important?” [Online]. Available: <https://www.altexsoft.com/blog/what-is-secdevops/>
- [10] OWASP, “Samm model information and characteristics.” [Online]. Available: <https://owasp samm.org/about/>
- [11] B. G. [NIST], “Leveraging the owasp software assurance maturity model (samm).” [Online]. Available: <https://www.nist.gov/system/files/documents/2021/09/03/OWASP%20SAMM%20-%20BrianGlad-and%20NIST%20Consumer%20Software%20Labeling.pdf>
- [12] BSIMM, “Bsimmm framework.” [Online]. Available: <https://www.bsimm.com/framework.html>
- [13] S. K. R. Security], “Developer-first security, shifting security left with a plan.” [Online]. Available: <https://www.reshiftsecurity.com/developer-first-security-shifting-security-left-with-a-plan/>
- [14] Synopsys, “What is the bsimm and how does it work?” [Online]. Available: <https://www.synopsys.com/glossary/what-is-bsimm.html#B>
- [15] B. G. [OWASP], “Comparing bsimm & samm.” [Online]. Available: <https://owasp samm.org/blog/2020/10/29/comparing-bsimm-and-samm/>

-
- [16] Synopsys, “Building security in maturity model (bsimm).” [Online]. Available: <https://www.synopsys.com/software-integrity/software-security-services/bsimm-maturity-model.html>
 - [17] OWASP, “Owasp top 10: 2021.” [Online]. Available: <https://owasp.org/Top10/>
 - [18] —, “Application security verification standard [git repository].” [Online]. Available: <https://github.com/OWASP/ASVS/tree/v4.0.3#latest-stable-version---403>
 - [19] C. Community, “About cwe.” [Online]. Available: <https://cwe.mitre.org/about/index.html>
 - [20] —, “2022 cwe top 25 most dangerous software weaknesses.” [Online]. Available: https://cwe.mitre.org/top25/archive/2022/2022_cwe_top25.html
 - [21] N. S. [CMU], “Thread modeling: 12 available methods.” [Online]. Available: <https://insights.sei.cmu.edu/blog/threat-modeling-12-available-methods/>
 - [22] Cynance, “Pasta threat modelling – the complete cyber security meal.” [Online]. Available: <https://www.cynance.co/pasta-threat-modelling/>
 - [23] A. H. S. Secured], “Stride threat modeling: What you need to know.” [Online]. Available: <https://www.softwaresecured.com/stride-threat-modeling/>
 - [24] Jenkins, “Jenkins developer reference.” [Online]. Available: <https://www.jenkins.io/doc/developer/book/>
 - [25] HashiCorp, “Manage secrets & protect sensitive data with vault.” [Online]. Available: <https://www.vaultproject.io/>
 - [26] OWASP, “Owasp zap.” [Online]. Available: <https://www.zaproxy.org/>
 - [27] GitHub, “Codeql repository.” [Online]. Available: <https://github.com/github/codeql>
 - [28] E. Parliament and C. of the European Union, “General data protection regulation.” [Online]. Available: <https://gdpr-info.eu/art-32-gdpr/>
 - [29] OWASP, “Cross-site request forgery prevention cheat sheet.” [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html
 - [30] —, “Cross site scripting prevention cheat sheet.” [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html
 - [31] —, “Injection prevention cheat sheet.” [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Injection_Prevention_Cheat_Sheet.html
 - [32] Django, “Django form and field validation.” [Online]. Available: <https://docs.djangoproject.com/en/4.0/ref/forms/validation/>
 - [33] M. Focus, “About fortify.” [Online]. Available: <https://cloudsecurity.cyberres.com/application-security-trend-report-hub/about-fortify>

-
- [34] Django, “Configuring django-csp.” [Online]. Available: <https://django-csp.readthedocs.io/en/latest/configuration.html>
- [35] Atlassian, “Jira software features.” [Online]. Available: <https://www.atlassian.com/software/jira/features>