

TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Hannes Toots 179906IVSB

**Security Analysis of the Web-based
Configuration Process at the Customer
Relationship Management
Team of a Bank**

Bachelor's thesis

Supervisor: Kaido Kikkas
Doctor of Philosophy
(PhD)
Mikko Maltsaar
Bachelor's Degree
(BSc.)

Tallinn 2022

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Hannes Toots 179906IVSB

Panga kliendisuhete haldustiimi veebipõhise häälestusprotsessi turvaanalüüs

Bakalaureusetöö

Juhendaja: Kaido Kikkas

Tehnikateaduste
doktor (PhD)

Mikko Maltsaar

Bakalaureusekraad
(BSc.)

Tallinn 2022

Author's declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Hannes Toots

15.05.2022

Abstract

The Customer Relationship Management team of the given bank has started developing access rights to other teams within the institution to their web-based configuration processes. During the expansion of the system's logic, the author has begun wondering whether the processes are completely secure and contain no security vulnerabilities. This thesis aims to assess the security of the web-based configuration processes.

The thesis first introduces the background of the topic – the CRM system. The topic begins by giving an overview of processes between CRM and Internet Bank. An example is given of how the customer view is generated within IB. It continues by giving a detailed explanation of CRM's web application design and its configuration processes by creating a test topic and linking multiple parameters to it. The background ends with an explanation of why this topic was chosen by the author.

Further, the thesis describes two different security frameworks which would enable easier assessment of the web application processes. The author decided on the best methodology by conducting a score voting based on three factors. Further, three different testing methods are discussed, which could be following during the security assessment of the web application. The author uses multiple factors to decide on which method to use. The topic ends by explaining how the framework requirements were validated on the web application, following the chosen testing method.

The last chapter starts with the results of the security analysis. The full list of satisfied and non-satisfied results cannot be shown in this thesis, as it is not good practice to publish detected vulnerabilities before they are mitigated or solved. Following the results, the author investigated one framework requirement and mitigated an identified vulnerability. Chapter ends with the author's opinion of how to use the findings to future improve the security of the various configuration processes.

This thesis is written in English and is 49 pages long, including 5 chapters, 22 figures and 3 tables.

Annotatsioon

Panga kliendisuhete haldustiimi veebipõhise häälestusprotsessi turvaanalüüs

Käesoleva panga kliendisuhete haldustiim on alustanud arendusi, mis võimaldavad anda ligipääsu veebipõhiste häälestusprotsessidele asutusesiseselt ka teistele tiimidele. Uue loogika rakendamise jooksul on tekkinud autoril arvamus, et mainitud protsessid pole täielikult turvalised ning kasutajatel on võimalik programmi turvakontrollidest mööda hiilida. Selle tööga soovib autor hinnata veebiprogrammi häälestusprotsessi turvalisust.

Esmalt tutvustatakse teema tausta – CRM süsteemi. Peatükis kirjeldatakse protsesside käiku tiimi süsteemides ning näidatakse konfigureerimise tulemust ka tarbija vaatepunktist. Edasi antakse detailsem ülevaade veebiprogrammist ning selle konfigureerimise võimalustest, valmistades ette ühe testeema ning lisades sellele juurde seonduvaid parameetreid. Sealt edasi selgitatakse, miks valiti antud teema lõputööks.

Järgmiseks kirjeldatakse kahte erinevat turvaraamistiku, mis hõlbustaksid käesoleva veebiprogrammi turvaanalüüsi. Parima raamistiku valimiseks hindas autor neid subjektiivselt kümnepallisüsteemis, kolmel erineval viisil. Peatükis kirjeldatakse ka kolme erinevat testimise meetodikat, mille vahel, võttes arvesse mitmeid tegureid, valis autor välja ühe. Peatükk lõpeb selgitusega, kuidas valitud standardit häälestusprotsesside testimiseks kasutati.

Viimases peatükis kirjeldatakse turvaanalüüsi tulemust. Terviklikku loendit analüüsi tulemustest lõputöös näidata ei saa, sest veebiprogrammis on vaja sisse viia parandused, et see turvanõuetele vastaks. Ühtlasi ei ole hea tava avaldada paikamata turvaauke. Pärast analüüsi uuris autor ühte nõuet detailsemalt. Selle tulemusena leiti rakenduses nõrkus, mida lõputöö jooksul parandati. Lõpetuseks kommenteerib autor, kuidas leitud tulemuste abil on võimalik rakenduse turvalisust täiendada.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 49 leheküljel, 5 peatükki, 22 joonist, 3 tabelit.

List of abbreviations and terms

ASVS	Application Security Verification Standard
CRM	Customer Relationship Management
CRUD	Create, read, update, and delete
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IB	Internet Bank
NIST	National Institute of Standards and Technology
OWASP	Open Web Application Security Project
REST	Representational state transfer
RSS	Really Simple Syndication
SPA	Single-page application
SSDF	Secure Software Development Framework
UI	User interface
URL	Uniform Resource Locator

Table of contents

1 Introduction	11
2 Background.....	12
2.1 CRM system	12
2.2 Web application.....	15
2.2.1 Application design.....	15
2.2.2 Configuration options	20
2.2.3 Example configuration	21
2.3 Security assessment cause	24
3 Methodology.....	25
3.1 Security framework	25
3.1.1 NIST SSDF.....	26
3.1.2 OWASP ASVS	27
3.1.3 Framework choice	28
3.1.4 ASVS blueprint creation.....	29
3.2 Testing technique.....	30
3.2.1 White box testing.....	31
3.2.2 Black box testing	33
3.2.3 Grey box testing	34
3.2.4 Testing method choice.....	35
3.3 Requirement validation.....	37
4 Analysis and results	38
4.1 Security assessment result	39
4.2 Proof-of-concept solution	40
4.2.1 Vulnerability introduction	40
4.2.2 Scope analysis	42
4.2.3 Vulnerability mitigation	43
4.3 Further steps for security improvement.....	46
5 Conclusion	47
References	48

Appendix 1 – Non-exclusive licence for reproduction and publication of a graduation
thesis 49

List of figures

Figure 1 Flowchart of communication between IB and CRM.	13
Figure 2 Two IB views of requested topics.	13
Figure 3 Web application's main menu screen form.	16
Figure 4 Action <i>omadus</i> configuration.	17
Figure 5 Rendered result of the <i>omadus</i> action.	19
Figure 6 Topic view in configuration process.	20
Figure 7 Parameter view in configuration process.	21
Figure 8 List of values for parameter <i>tahtae_kuud</i>	21
Figure 9 Test application view.	22
Figure 10 Topic configuration view.	22
Figure 11 Internet Bank's view of test topic.	23
Figure 12 Internet Bank's view with more parameters.	23
Figure 13 One OWASP ASVS point used within the assessment.	27
Figure 14 Classification of white box testing [11].	31
Figure 15 Parameter configuration screen.	40
Figure 16 Parameter form element in HTML.	41
Figure 17 Overwritten parameter <i>testoml</i>	41
Figure 18 Configuration of texts with current selected identifier in database.	43
Figure 19 New class for validating instantiated string with user session value.	44
Figure 20 Error created to be shown to user.	44
Figure 21 New procedure added to parameter saving module.	45
Figure 22 Validation error when attempting to inject identifier value.	45

List of tables

Table 1 Security framework evaluation.....	28
Table 2 OWASP ASVS requirements tested within the web application.	29
Table 3 Vulnerability scope result.....	42

1 Introduction

As of today, CRM team in the given financial institution is facing difficulties providing development and configuration services to all teams within the financial institution due to increasing workload and requirements set by other teams within the organization. As such, CRM team is aspiring towards outsourcing its configuration processes to other teams to lower development complexity and reduce dependencies between different team requirements.

The current web application can be classified as an in-house system that has been developed and maintained by the CRM team for over twenty years and has seen continuous developments during its life cycle. Due to ever changing requirements within the organization, the team has spent most of their efforts developing additional features in accordance with business requirements. Since the long evolution of the application, the author suggests analysing whether some security risks have been introduced during the many developments done by different personnel.

The purpose of this thesis is to assess one part of the given web application's security – its configuration processes. The analysis of given processes will be conducted using various grey box testing methods while following the version 4.0.3 security standard of Open Web Application Security Project's Application Security Verification Standard (OWASP ASVS). The analysis focuses on finding security vulnerabilities within the application's configuration processes in accordance with the selected testing framework. The author also aids in risk mitigation by investigating one topic within the standard in more detail, with the aim to discover a possible vulnerability. The discovered threat will then be mitigated with a development provided by the author.

The objective of the analysis is to document found vulnerabilities and increase CRM team's knowledge of potential security hazards that currently exist in their web application configuration processes.

Keywords: Web application security, Testing, Single-page application

2 Background

The organization, under which the CRM team resides, is an international bank and has employed over 4000 employees across the Baltic countries. This thesis covers one web application built by the customer Relationship Management team within the given financial institution.

This section of the thesis introduces the CRM system's objective within the bank and workflow within other systems of the organization. The thesis aims to assess the security of the procedures within CRM's web application; thus, the paper introduces the CRM software and shows results of changing the system's configuration through another application within the organization, Internet Bank. This program is used by customers to manage their funds and banking services online. The background will conclude with the full reasoning of what decided the security assessment to be started.

2.1 CRM system

The main objective of CRM's system is to handle every interaction between the customer and the bank. It is one of the most important systems of the bank, and is integrated with many others, including the core system. It has existed within the organization for over twenty years and has been developed by a dedicated team. It is vital that the system functions properly as it is the centre of customer information and its structure. As such, the processes are a necessary backbone for the functionalities of other teams.

For example, CRM's system has control over how different loans are shown in other user interfaces within the organization, with one of them being IB. The former has control over the configuration of such forms within its system's database. The latter requests this information from the former and uses the returned data to construct views on the customer point of view within the organization. Figure 1 explains the communication between IB and CRM's system when a customer starts any application through IB.

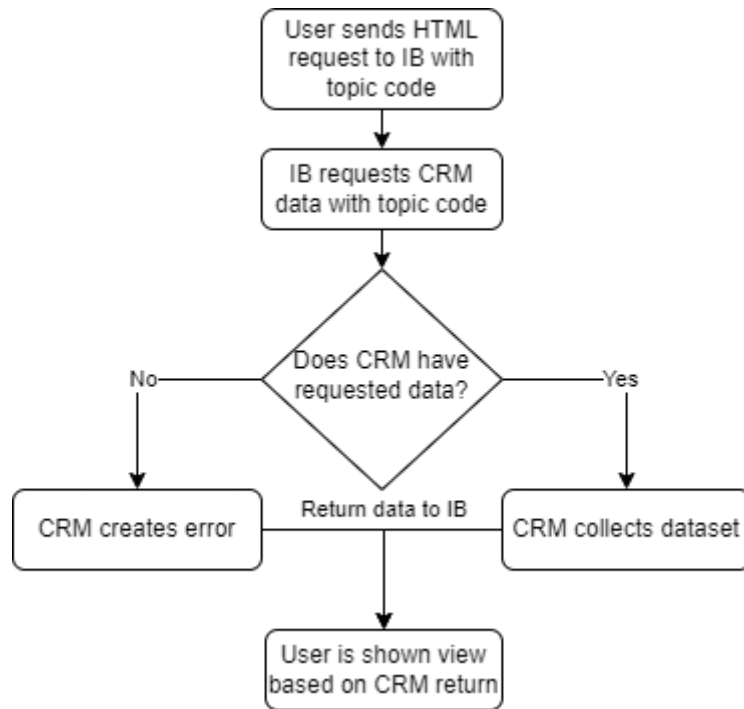


Figure 1 Flowchart of communication between IB and CRM.

Figure 2 showcases two different forms that are created when the given user applies for a loan within IB. In both cases, IB uses CRM services to gather the specific application's configuration data and uses it to create a form for the user. On the left, the IB request to the CRM system contains topic code *appl_homel*, which is the internal identifier for the home loan application. On the right, the topic code is *0*, which causes CRM's system to return an error.

Home loan

Please enter applicant's name and personal ID code

* First name

* Family name

* ID code

Home loan

* Special purpose of loan

Please enter applicant's name and personal ID code

* First name

* Family name

* ID code

❗ crmContactTopic

Figure 2 Two IB views of requested topics.

To better understand how the CRM system controls the data on the home loan form, a short introduction needs to be given to three CRM terms – *contact*, *topic*, and *parameter*:

- **Contact** – A contact can be described as a data set required for the organizational systems for its internal processes. It is used to describe interactions between the bank and the customer and contains metadata of the whole application that is necessary for following actions. [1] When an application is submitted to the organization or one is created by a teller, a contact is automatically created containing basic information – its creation time, the customer linked to it, where it was created (bank teller, Internet Bank, etc) and more. Different topics and parameters can be linked to contacts to provide more detailed information. A contact must have at least one topic.
- **Topic** – A contact is distributed between different topics. During a contact, it is possible to handle different subjects, such as loans, agreements, or applications. A topic defines the exact topic of discussion during the contact and is configurable within CRM system. [1] Much like a contact, a topic itself mostly contains metadata specific for the topic and contains more detailed information with the parameters linked to it. Topics can be added to contacts by the employees of the institution. For example, when personnel add a positive result to a loan application, the newly added result is an extra topic under the contact.
- **Parameter** - Contacts and topics contain elements that characterize the specific object. These elements are called parameters. The configuration describes and aids to link parameters to specific topics or contacts. [1] After the initial creation of a parameter, they are linked to the required topic or contact, where they can be catered to the specific needs of the given object (made mandatory, have a maximum length, etc). The left view visible in Figure 2 contains parameters linked to the home loan application, configured within the CRM's system.

For example, when any user submits the home loan application visible in Figure 2 through Internet Bank, among other steps, the CRM system does the following:

1. Contact is created automatically, which contains the whole dataset of the loan application. It defines the communication with the organization.

2. Required topics are created within the contact, containing the metadata of the loan application. Created topics are linked to the specific contact within the dataset.
3. Form variables that the user has given to the bank are stored within the various parameters of the whole contact – under the topic or the contact, depending on the specific parameter.

CRM's system sets the data structure of various customer data forms. It is possible to change different properties of various datasets, including topics and parameters. This allows for greater customization and easier catering to specific requirements. All data fields presented to the user are prepared through CRM's internal procedures and used by other systems to their specific needs, for example in IB it is used to show the user the given application form.

The system's database also contains the data that is stored from the above interaction; however, the data storage falls out of the scope of this thesis. The following topic will discuss the assessed web application made by the CRM team in more detail. It will also show an example of how changing the configuration can affect other areas within the institution.

2.2 Web application

This part of the thesis covers CRM team's web application (hereinafter as "application") in detail. The web application itself is built as a single page application, meaning that only one module is used to drive the whole logic of the product. The thesis will go into further details how different Hypertext Transfer Protocol requests are handled by the application, and how Hypertext Markup Language forms are presented to the user. Further, this section gives an overview of the possible configuration possibilities within the application. It will end with an example configuration process being done in the development environment and showcase the results visible through IB.

2.2.1 Application design

The application is built using the OpenEdge Advanced Business Language programming language. The language has its own environment called WebSpeed, which allows

building various applications that use mark-up languages as the user interface. It is also the environment where the assessed web application is deployed on.

The web application is designed as a Single-page application, using a single module that updates parts of the interface without sending or receiving a full-page request. All necessary content visible to the user is rendered using HTML templates. [2]

The views visible to the user and rendered by the web application are designed as *screen forms* and are configurable within the system. Figure 3 shows the screen form that is used to render the main page of the web application, a blank page containing the menu bar.

SCREENFORMS
Screenform code: main_menu Screenform group code: Screenform name: #Peamenüü Modul name: crm_mainmenu Menu: Yes
Opened by: ADMINISTRATOR 19.02.2001

Figure 3 Web application's main menu screen form.

Screen forms consists of different types of data. The *Menu* variable is out of use as it is an old data field that has been deemed obsolete. [3] The other variables have the following objectives:

- Screen form code – contains the identifier for this screen form.
- Screen form group code – Used by various processes to group user session parameters that are saved to the server-side database. It allows easier filtering of different variables based on the form group code that was in use while the parameter was stored. In this case it is not used and is empty.
- Screen form name – This name is rendered in the user view if it is selectable by the user. In the case of this form, it is a temporary name as the form itself is not selectable.
- Modul name – Name of the module that is rendered to the user when this screen form is activated. The module should be of type HTML and contain the view that is displayed to the user when the form is activated.

It can be noted that the view has an *Entered by* field, which contains the name behind the username who did the last change to this topic. However, in development environments, this name is scrambled and therefore does not contain real life information. Also, in this case, it was created by an admin account dating back to the starting years of the CRM system.

As described previously, the application consists of one main module that handles the requests sent by the user. There are many different configured screen forms that can be displayed to the user based on the submitted request. Requests are parameterized within the application as different *actions*, which are also configurable within the CRM system. When a user clicks on a button that sends a request to the application, an action is activated that commands the main module to run a set of code, specified within the action itself. [1] Figure 4 shows an image of an action that is activated when the user requests a table view of parameters from the application.

ACTION SETUP

Action: omadus

Name in english: Parameters

Description in native language:

Description in english:

Description in russian:

Name in native language: Omadused

Name in russian:

Right: H

Button: yes

Preserve: no

Empty: no

First icon of button:

Second icon of button:

Main action: konfid - Conf

Additional action:

Display screenform: omadus_valik - PARAMETER CHOICE

Display menu: konfid_menu - CONF

Belong to menu: konfid_menu - CONF

Queue number in menu: 1

Modul: crm_omadus-find

Parameters: moodul=valik,om_alates=1,form=V,btn_cancel=omadus_val_tagasi

Change **Change extra** **Close**

Figure 4 Action *omadus* configuration.

Actions can also be linked to each other, creating a hierarchical structure. Specific variables are understandable from the figure; however, some fall out of the scope of this thesis. The following gives an overview of more important variables to understand the logic behind actions:

- Main action – As described previously, actions can be linked to each other to form a structural order. Main action is used by the application as a pointer of the action that was run before this action became possible to be run, should the currently running action run into an error. In the case of a fault, the main module of the web application will start processing the action configured under the main action, the one that was ran previously. [3]
- Additional action – Specifies an action that is always run after the current action has finished. [3] Usually, actions that contain additional actions do not have any forms as they are used for back-end procedures, such as updating data within a database. After the action has completed, the additional action is used to redirect the user to a different view.
- Display screen form – This links the previously shown screen form that is rendered to the user to the given action. When the user activates this action, the HTML file behind this form is what is finally rendered to the user, should it run successfully.
- Display menu – If the action needs to render the menu of the web application in addition to the final screen form, it is configured here. [3] In this case, the user is rendered the web application menu alongside the following parameter table from the action screen form.
- Belong to menu – Tells the web application where within the menu this action should be visible. Here it is the same as the displayed menu which is created above the rendered list.
- Modul – Used to specify which module is activated by the main controller of the web application when this activation is run. Here the module is *crm_omadus-find*, which contains logic for collecting parameters from the database. Modules are run before HTML forms are rendered to the user.
- Parameters – Parameters contain a list of different variables that are stored within the current user session before the module is run. [3] It is probable that the following modules use the saved values.

Actions give an overview of the underlying pieces of modules that are activated by the SPA to perform tasks within the application. When the user activates the action, they are given the following view inside the application. Figure 5 shows the resulting view when the above action is activated. It can be noted that the main menu, which in this case consists of clickable underlined links, is fully rendered using the *Display menu* form.

The screenshot shows a web application interface with a main menu at the top containing links like 'Interactions', 'CEX', 'Marketing', 'Reports', 'Agreements', 'Links', 'Settings', 'Developer', and 'Exit'. Below the menu is a search bar for 'PARAMETER CHOICE' with fields for 'Code', 'Name', and 'Type' (set to 'ALL'), and a 'Search' button. The main content is a table with the following data:

Code	Name	Type	Related to
ddnext_review	Next review date	T	DD review
dd_review	DD review	T	DD review
telefon1_comm	telephone 1 comment	KYC,T	Private customer contact data
telefon2_comm	telephone 2 comment	KYC,T	Private customer contact data
telmobill_comm	Mobile comment	KYC,T	Private customer contact data
loans_number	Number of home loan liabilities	T,L	Loan administration order, Home loan, Home loan with EIS guarantee, Mortgage Loan
inforeq_params	Inforequest response parameters	M	
loan_liab	Number of home loan liabilities	T,L	
test_h	test param	UI	
inforeq_results_mapping	Inforeq application result mapping	M	
req_order_ID	Request ID	T	NBSX Inforequests
req_type2	Request type	T	NBSX Inforequests
req_email	Requester email	T	NBSX Inforequests
setup_manager_tf	TF allowed values	M2	
kyc_res-rights-investscheme	Did you obtain residence rights under citizenship by investment scheme?	KYC,T,I38	Private customer additional data (dyn. form)
kyc_res-rights-other-jur	Do you hold residence rights in any other jurisdiction(s)?	KYC,T,I38	Private customer additional data (dyn. form)
kyc_90days-other-jur	Have you spent more than 90 days in any other jurisdiction(s) during the previous year?	KYC	Private customer additional data (dyn. form)
kyc_incometaxreturn-other-jur	Have you filed personal income tax returns during the previous year in other jurisdictions?	KYC	Private customer additional data (dyn. form)
req_title	To title	T	NBSX Inforequests
req_court_dcsn	Court decision	T	NBSX Inforequests

At the bottom of the table, there are 'Next' and 'Add' buttons.

Figure 5 Rendered result of the *omadus* action.

Actions play a vital role in the design of the application. They are used on HTTP requests to instruct the main module of the program to run a piece of code and specify the parameters required, if needed. The resulting view on the UI is rendered through the screen forms linked to the requested action.

The configuring processes within the application are designed to be used by developers to create, read, update, and delete stored data within the CRM system. It gives access to the database through the web interface and can be used to interact with it. The next topic will showcase an example configuration of a topic using the application.

2.2.2 Configuration options

CRM's web application allows to edit many different variables within the system. The full list of configurable parameters is visible in the third row of the main menu in Figure 5, starting with *Parameters* and ending with *Actions*. Figure 6 shows an example of a topic that was created for the purpose of testing configuration processes during this thesis.

The screenshot displays the 'TOPIC DATA' configuration page. It contains several sections of metadata and requirements:

- Topic code:** test_a
- Main topic code:** Automatic
- Right:** W
- Initiator:** customer-bank
- Type:** 110
- Difficulty level:** ?
- Corporate customer:** ?
- Sort of:** ?
- Topic name in native language:** test topic
- Topic name in english:** test topic
- Topic name in russian:** test topic
- Additional topics:** ?
- Difficulty levels:** ?
- Report form:** ?
- Product:** ?
- List of product:** ?
- Product requirement:** ?
- Agreement requirement:** ?
- Title requirement:** ?
- Response requirement:** ?
- Difficult level requirement:** ?
- Message requirement:** ?
- Solution requirement:** ?
- Content requirement:** ?
- Decision requirement:** ?
- Repository requirement:** ?
- Title:** ?
- Default content:** ?
- Help notes:** ?
- Entered by:** TERTTU KLASEN 20.04.2022

At the bottom, there are buttons for 'Change', 'Delete', 'Relations', and 'Back', along with a 'Look history' link.

Order nr	Relation name	Relation code	Parent parameter	T	Default value	M	P	Min	Max	N	N	Function	L	H
0	Aasta	aasta			?	no	no	?	?	1	?	?	1	1
1	Loan amount	laenusumma			?	yes	no	7000	?	1	?	?	1	1
2	Month(s)	tahtae_kuud			?	no	no	?	?	1	?	?	1	1

Figure 6 Topic view in configuration process.

As seen in the image, topics have many metadata variables that are used in various forms. The thesis will not go into the full meaning of each parameter, as it falls out of the scope of the overall configuration processes.

Under the topic, a list of related parameters that have been connected to this topic is shown to the user. After parameters are linked, they can also be modified within the topic, catering to the specific needs of the current topic. In Figure 6, the value *laenusumma* has been made mandatory and given a minimum value of 7000 within this topic. This means that if another system within the bank sends the corresponding topic data to CRM's system, an automatic check determines whether the value corresponds to the given rules.

While the previously mentioned parameter is an input parameter that is filled by the user or other processes within the application, Figure 7 shows the configuration of one other linked parameter which contains multiple selectable values, *tahtae_kuud*.

PARAMETER DATA	
Parameter code: tahtae_kuud	
Name in native language: Kuu(d)	Value name: Väärtus
Name in english: Month(s)	Value name in english: Väärtus
Name in russian: Месяц	Value name in russian: Väärtus
Description:	
Rights: W	Preserve: no
Type: T,L	
Additional parameters:	
Function:	
Number of values: 1	Number of values by default: 1
Data type: CHAR	Format:
Conversion: N	Default value:
Length: 2	Minimum value:
Width: 1	Maximum value:
Display form (UPDATE): Vertical	Display form (READ): Vertical
Comment allowed: no	Comment width: 1
Comment name in native language: Kommentaar	Comment length: 10
Comment name in english: Kommentaar	Entity:
Comment name in russian: Комментарий	Currency default value:
Helptext code:	Currency restrictions:
	Currency display form:
Entered by: TADOLDER MARET 03.07.2003	
<input type="button" value="Values"/> <input type="button" value="Change"/> <input type="button" value="Delete"/> <input type="button" value="Back"/>	

Figure 7 Parameter view in configuration process.

When the user presses on the *Values* button, they are moved to a new screen, showcasing the values configured under the given parameter. When the front-end user is given a topic that is related with this parameter, a select box is created for the user, with the values taken from the parameter itself, visible in Figure 8.

PARAMETER - MONTH(S) - VALUES													
Value	Value2	Value3	Value4	Value5	Order	Value name	Value name in english	Value name in russian	Helptext code	Function	Entered by	Entry date	Delete
01					2	1 kuu	1 month	1 месяц	0		TADOLDER MARET	03.07.2003	Delete
02					4	2 kuud	2 months	2 месяца	0		TADOLDER MARET	03.07.2003	Delete
03					5	3 kuud	3 months	3 месяца	0		TADOLDER MARET	03.07.2003	Delete
04					6	4 kuud	4 months	4 месяца	0		TADOLDER MARET	03.07.2003	Delete
05					7	5 kuud	5 months	5 месяцев	0		TADOLDER MARET	03.07.2003	Delete
06					8	6 kuud	6 months	6 месяцев	0		TADOLDER MARET	03.07.2003	Delete
07					9	7 kuud	7 months	7 месяцев	0		TADOLDER MARET	03.07.2003	Delete
08					10	8 kuud	8 months	8 месяцев	0		TADOLDER MARET	03.07.2003	Delete
09					11	9 kuud	9 months	9 месяцев	0		TADOLDER MARET	03.07.2003	Delete
10					12	10 kuud	10 months	10 месяцев	0		TADOLDER MARET	03.07.2003	Delete
11					13	11 kuud	11 months	11 месяцев	0		TADOLDER MARET	03.07.2003	Delete
00					14	0 kuud	0 month	0 месяцев	0		TADOLDER MARET	27.10.2009	Delete

Figure 8 List of values for parameter *tahtae_kuud*.

2.2.3 Example configuration

To show what occurs when CRM system's configuration is updated, a blank topic was created within CRM, visible in Figure 9. Different parameters were added to this topic, and the resulting view was shown through IB.

TOPIC DATA											
Topic code: test_a						Type: 110					
Main topic code: Automatic						Difficulty level: ?					
Right: W						Corporate customer: ?					
Initiator: customer-bank						Sort of: ?					
Topic name in native language: test topic											
Topic name in english: test topic											
Topic name in russian: test topic											
Additional topics:											
Difficulty levels:											
Report form:											
Product:											
List of product:											
Product requirement: ?											
Agreement requirement: ?											
Solution requirement: ?											
Title requirement: ?											
Content requirement: ?											
Response requirement: ?											
Decision requirement: ?											
Difficult level requirement: ?											
Repository requirement: ?											
Message requirement: ?											
Title:											
Default content:											
Help notes:											
Entered by: TERTTU KLASSEN 20.04.2022											
Link history <input type="button" value="Change"/> <input type="button" value="Delete"/> <input type="button" value="Relations"/> <input type="button" value="Back"/>											

Order nr	Relation name	Relation code	Parent parameter	T	Default value	M	P	Min	Max	N	N	Function	L	H
<input type="button" value="Add"/>														

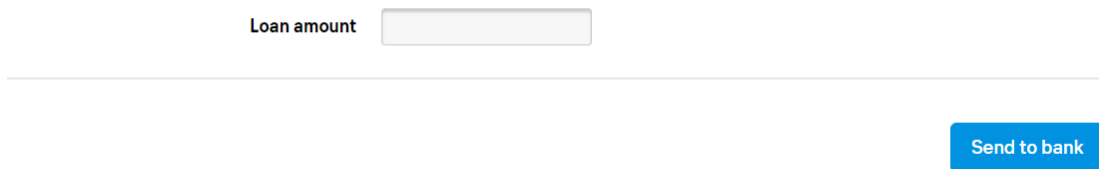
Figure 9 Test application view.

When adding parameters to a topic, the user must press the *Add* button at the bottom of the UI. This will open a new window where the user can add any existing parameter to the topic. The parameter must exist within the CRM system before it can be added to the topic. Figure 10 shows the view the user is shown when linking an existing parameter to the topic. As visible, the parameter select box is very large. This is because the system collects all possible parameters with their full values to this list. Some parameters have very long names.

ADD PARAMETER	
Setup configuration guide	
Topic code: test_a	
Parameter: <input type="text"/>	
Parent parameter: <input type="text"/>	
Type: <input type="text"/>	
Function: <input type="text"/>	
Preserve: <input checked="" type="radio"/> NO <input type="radio"/> YES	
Number of values: <input type="text"/>	
Order: <input type="text"/>	
Mandatory: <input checked="" type="radio"/> NO <input type="radio"/> YES	
Minimum value: <input type="text"/>	
Default value: <input type="text"/>	
Comment by default: <input type="text"/>	
Help text code: <input type="text"/>	
Number by default: <input type="text"/>	
Level: <input type="text"/>	
Comment allowed: <input checked="" type="radio"/> NO <input type="radio"/> YES	
Maximum value: <input type="text"/>	
Currency value by default: <input type="text"/>	
Currency restrictions: <input type="text"/>	
Currency display form: <input type="text"/>	
Entered by: KLASSEN TERTTU 20.04.2022	
<input type="button" value="Save"/> <input type="button" value="Cancel"/> <input type="button" value="Reopen closed parameter"/>	

Figure 10 Topic configuration view.

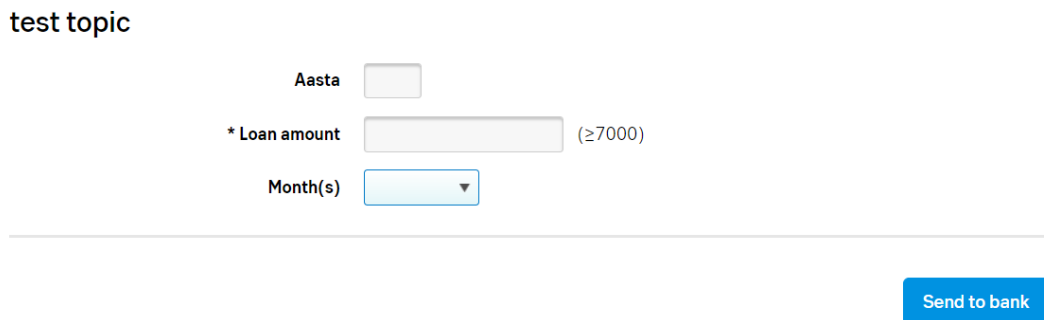
For the initial test, only one parameter with the code *laenusumma* was added to the topic. This topic can also be opened in IB, and the resulting view should showcase the topic with only one form input, titled *Loan amount*. Figure 11 is used to show that IB currently shows the user only one parameter within the topic, as expected.



The screenshot shows a form titled "test topic" with a single input field labeled "Loan amount". Below the input field is a blue button labeled "Send to bank".

Figure 11 Internet Bank's view of test topic.

Using the similar processes as previously described, other parameters can be added to the topic as well. Figure 12 shows the view from Internet Bank after adding more than one parameter to the test topic, with the configuration as visible in Figure 6.



The screenshot shows a form titled "test topic" with three input fields: "Aasta" (text), "* Loan amount" (text with a note "(≥7000)"), and "Month(s)" (dropdown menu). Below the input fields is a blue button labeled "Send to bank".

Figure 12 Internet Bank's view with more parameters.

CRM configuration is a powerful tool that can set up the base of how customer related products are shown in various systems within the organizations. Other systems within the institution request the setup data from CRM and use it in their own procedures. The web application enables the configuration options and allow its users to set up the objects as required.

2.3 Security assessment cause

As of today, CRM team is progressing towards enabling their configuration process for other teams within the bank, specifically the teams that rely on CRM data for further development of their own products. This means giving web application access to users from other teams that wish to make changes within the set-up parameters. Access to CRM system's configuration has previously been given only to select people and the number of users with configuration rights will increase substantially while moving towards outsourcing the processes to members of the other teams within the organization.

Giving other team developers access to configuration options will lower the reliance of CRM team's resources for configuring their product data and allow them to edit their own objects themselves, without CRM team's intervention. While CRM team is still required to validate the changes and allow them to move towards the production environment, less reliance is on them for enhancement within other team's products.

While the CRM team is constantly thinking about the security of their system and are currently preparing for giving other teams access with tasks such as limiting certain team users to only their respective configurations, the author of this thesis began to question whether the web application meets the security standards which would guarantee that users from other systems will be limited to their configuration processes and not have the rights to do something unintended.

Misconfiguration of CRM's objects can have many different results. It is possible that the forms that are generated and given to the bank customers are incorrect and contain data that does not match the legal compliance of the given product. Incorrect text configuration can also impact the applications that are printed out for contracts between the bank and its customers. This can impose possible legal risk for the organization. It is vital that the system is configured properly and with full compliance of other team's requirements.

Following the above, the author of this thesis decided to assess the security of CRM team's internal web application configuration processes to determine if they satisfy the requirements of up-to-date security verification standards.

3 Methodology

In this part of the thesis, the methodology of the security assessment is shown. First, to begin the security assessment, an existing framework of security requirements was chosen and used as a foundation to create a list of requirements that were evaluated within the web application.

Further, three “box” testing techniques – white, black, and grey box testing – were introduced that would later decide which method would be used for the security evaluation. The testing methods each contain different ideologies that will test the web application from different points of view.

Further, an assessment between the testing techniques were done to pick one that would be followed during the security evaluation. The choice was done on the backing of certain requirements which will be discussed more in the topic.

The methodology chapter will end with an explanation of how the security evaluations were done following the newly created blueprint and based on the chosen testing technique.

3.1 Security framework

Application security frameworks exist to help organizations determine the cybersecurity aspects for enhancing their application’s security. They offer a tried, tested, and comprehensive approach to cyber security developed by experts in the field. By adopting such an approach, you can take some of the guesswork out of developing or improving your own system, enhancing your defences in line with recognised industry best practice and accounting for aspects of cyber security that you might not otherwise have considered. [4]

The author of this thesis decided to follow a robust framework to aid in the testing of the web application. The skeleton can be used to create a logical flow to the vulnerability assessment and aid in the steps that should be done. To do this, two security frameworks were looked at in detail to decide on which framework should the author follow.

3.1.1 NIST SSDF

The National Institute of Standards and Technology Secure Software Development Framework is a core set of high-level secure software development practices that can be integrated into a software development life cycle implementation. Following the practices can help software producers reduce the number of vulnerabilities in released software, reduce the potential impact of the exploitation of undetected or unaddressed vulnerabilities, and address the root causes of vulnerabilities to prevent future recurrences. The framework provides a common vocabulary for secure software development which can be used in other management activities, including testing. [5]

The SSDF can help an organization to align and prioritize its secure software development activities with its business/mission requirements, risk tolerances, and resources. It describes a set of fundamental, sound practices for secure software development. Organizations should integrate the framework throughout their existing software development practice. [5]

The focus of the SSDF is on the outcomes of the practices rather than on the tools, techniques, and mechanisms to do so. This means that it can be used to by organizations in any sector or community, regardless of size or cybersecurity sophistication. It can also be used for any type of software development, regardless of technology, platform, programming language, or operating environment. [5]

NIST SSDF could be used within this security assessment through incorporation of the framework practices to the vulnerability testing. The practices contain a short introduction explaining why they are beneficial and have linked tasks to perform the practice successfully. The framework also gives notional implementation examples that can be used to implement a task within the software that is being developed. The SSDF also adds pointers to other established secure development practice documents and can be used by developers to aid in the task implementation to their product. One downside of the given framework within this thesis is that the detailed information of each practice will heavily impact required vulnerability assessment time due to the number of resources to review.

3.1.2 OWASP ASVS

OWASP ASVS is a community-driven effort to establish a framework of security requirements and controls that focus on defining the functional and non-functional security controls required when designing, developing, and testing modern web applications and web services. The web application is designed to be used as a blueprint to create a Secure Coding Checklist specific to any given application, platform, or organization. It allows its users to tailor the standard based on their specific needs to focus on the security requirements that are most important to who is using it. [6] The primary goal of the OWASP ASVS project is to normalize the range of coverage and level of rigor available in the marketplace for performing web application security verification. [7] Figure 13 shows one topic within the latest version of the framework during this thesis (v4.0.3), which requirements were also used during the security assessment.

V5.1 Input Validation

Properly implemented input validation controls, using positive allow lists and strong data typing, can eliminate more than 90% of all injection attacks. Length and range checks can reduce this further. Building in secure input validation is required during application architecture, design sprints, coding, and unit and integration testing. Although many of these items cannot be found in penetration tests, the results of not implementing them are usually found in V5.3 - Output encoding and Injection Prevention Requirements. Developers and secure code reviewers are recommended to treat this section as if L1 is required for all items to prevent injections.

#	Description	L1	L2	L3	CWE
5.1.1	Verify that the application has defenses against HTTP parameter pollution attacks, particularly if the application framework makes no distinction about the source of request parameters (GET, POST, cookies, headers, or environment variables).	✓	✓	✓	235
5.1.2	Verify that frameworks protect against mass parameter assignment attacks, or that the application has countermeasures to protect against unsafe parameter assignment, such as marking fields private or similar. (CS)	✓	✓	✓	915
5.1.3	Verify that all input (HTML form fields, REST requests, URL parameters, HTTP headers, cookies, batch files, RSS feeds, etc) is validated using positive validation (allow lists). (CS)	✓	✓	✓	20
5.1.4	Verify that structured data is strongly typed and validated against a defined schema including allowed characters, length and pattern (e.g. credit card numbers, e-mail addresses, telephone numbers, or validating that two related fields are reasonable, such as checking that suburb and zip/postcode match). (CS)	✓	✓	✓	20
5.1.5	Verify that URL redirects and forwards only allow destinations which appear on an allow list, or show a warning when redirecting to potentially untrusted content.	✓	✓	✓	601

Figure 13 One OWASP ASVS point used within the assessment.

OWASP ASVS can be incorporated to the thesis easily, allowing the security assessment to start quickly. The requirements give an overview of the necessities an application must have, and they can be hastily validated during the testing period of the security assessment. Compared to the NIST SSDF, OWASP ASVS is also less detailed, which can impact the depth of the security assessment. The tester needs to think more about the design of the application and confirm that the requirements are fully tested within the application, as the relevant information is not provided as largely as in the NIST SSDF.

3.1.3 Framework choice

In this thesis, the security framework is used to create a skeleton of steps that are followed during the web application's assessment. To choose an existing base, the author used the following metrics to decide on one:

1. Relevancy
2. Simplicity
3. Familiarity

The framework needs to be easy to use and relevant to the task – assessing the security of CRM's web application configuration processes. Understanding all the possible frameworks is accomplishable, however familiarity with the chosen standard will help lower the overall time required for the security assessment.

The choice between the frameworks was done subjectively by the author. The following metrics were considered – familiarity, simplicity, and relevancy. All topics were assigned a value based on a 10-point scale. The author decided to choose the framework using the sum of all the metrics. Table 1 shows the author's evaluation of the two different frameworks.

Table 1 Security framework evaluation.

Framework	Relevancy	Simplicity	Familiarity
NIST SSDF	7	4	7
OWASP ASVS	10	8	8

The total scores were as follows - NIST SSDF scored 18 points, whereas OWASP ASVS scored 26 points. Therefore, the author chose to use OWASP ASVS as a framework to test the CRM web application. It can be noted that the chosen standard is also frequently used during other security assessments within the organization.

3.1.4 ASVS blueprint creation

OWASP ASVS is a very thorough framework, containing fourteen different security topics that a web application should follow to be classified with the standard's level 3 protection. However, many of these points are not relevant when assessing the security of the given web application's security processes. For example, the V1 of the standard, "Architecture, Design and Threat Modeling", contains points for the web application's architecture, which falls out of the scope of the thesis.

After the security framework was chosen, it was evaluated that many of the topics in the skeleton are not needed in this thesis. As also written in the OWASP ASVS itself, the framework should be used to create a Secure Coding Checklist that caters towards the special necessities of the application being tested. Therefore, a blueprint needed to be created following the OWASP ASVS that should contain the checklist necessary to verify the configuration process' security. This was accomplished by reviewing ASVS itself and deducing the necessary points from it into a new blueprint. Table 2 shows the list of the framework requirements that were subjectively chosen by the other to be used to assess the security of the web application configuration processes.

Table 2 OWASP ASVS requirements tested within the web application.

V5	V7	V8
V5.1.1	V7.1.3	V8.1.3
V5.1.3	V7.1.4	V8.1.4
V5.1.4	V7.2.1	V8.2.2
V5.2.2	V7.4.1	V8.2.3
V5.2.4	V7.4.2	
V5.3.1	V7.4.3	
V5.3.2		
V5.3.3		
V5.3.4		
V5.3.5		
V5.3.6		
V5.3.8		
V5.4.2		
V5.4.3		

All the requirements incorporated in this security assessment are, in some form, linked to the configuration processes of the web application. V5 of the standard requires the software under verification to ensure that the input and output data is secured and validated. V7's primary objective is to confirm that error handling and logging procedures provide high quality results which are protected as per local data privacy laws. While V8's key element is data protection, and private data is not available through the application's configuration interfaces, some requirements within the topic are still relevant to the software's processes. [6] It was decided that other requirements have either too large of a scope or handle information not relevant to the configuration process (i.e., application architecture or private data handling).

3.2 Testing technique

In security testing, we consider the entire set of unacceptable inputs – infinity – and focus on the subset of those inputs that are likely to create a significant failure with respect to our software's security requirements – still infinity. We need to establish what those security requirements are and decide what kind of tests will provide evidence that those requirements are met. With logic and diligence, we can provide useful evidence to the product's owner. [8]

Software testing has been widely used in the industry as a quality assurance technique for the various artifacts in a software project, including the specification, the design, and the source code. As software becomes more important and complex, defects in software can have a significant impact to users and vendors. The importance of planning cannot be underestimated. It is carried out to demonstrate the presence of errors that exist during the program's execution and can provide a higher chance of discovering unidentified design flaws. [9]

There are many testing techniques available to assess the security of a web application. The author decided to test the vulnerability checklist using one of three existing testing systems – Black, white, and grey box testing. During this topic, the three methods are discussed, and author will choose one of them to be the main testing method for the security analysis of the web application.

3.2.1 White box testing

Most software starts with the initially developed code. White box testing is a testing strategy based on internal paths, code structures, and implementation of the software being tested. [10] White box testing generally requires detailed programming skills, as the tester using this method needs to understand and access the applications' source code. Figure 14 explains the different types of white box testing techniques.

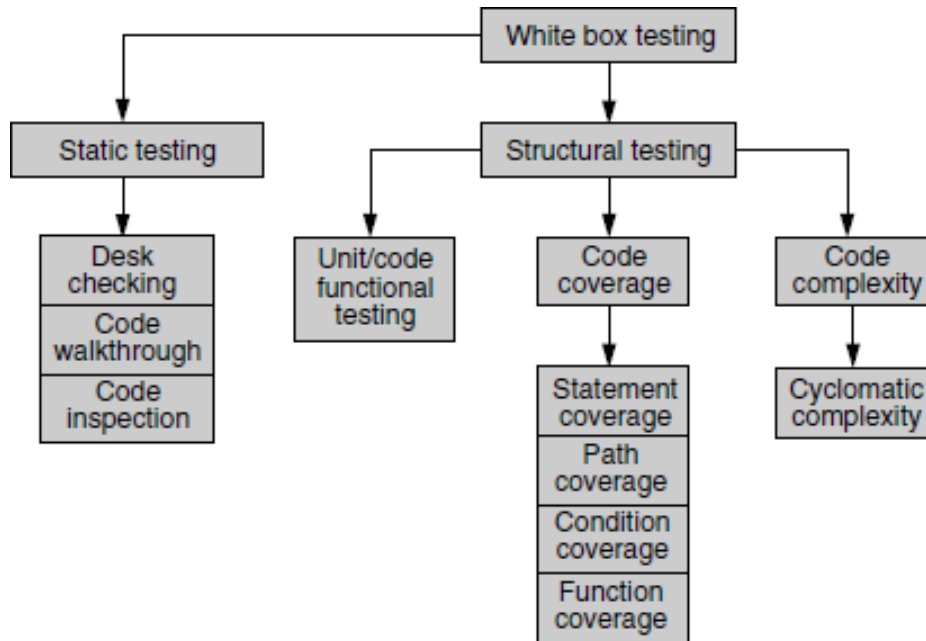


Figure 14 Classification of white box testing [11].

As shown, white box testing is classified into two different types of testing, *static* and *structural* testing.

Static testing is a testing method which requires only the source code of the application without its binaries or executables. It involves select people reviewing the code and determining whether it works according to the original requirements, is fully functional and handles exceptions correctly. This type of testing can be done by humans or with the help of tools, however it is important that the tools are specialized for the given application. [11] The latter can also introduce new developments and more time investment – a large resource that some teams might not have.

Structural testing is like Static testing in the sense that it continues the trend of using developed code and design of the application to test the application. However, Structural testing also incorporates use of the runtime environment. This means that alongside the developed code the system is also tested with its executables or binaries. Structural testing usually means that the product is run against predesigned test cases to exercise as much of the code as possible, meaning that the amount of code executed during the testing period is maximised. [11]

The major advantage of strategies used during white box testing is that the objective criteria can be defined to quantify the coverage of a given test suite. Because the testing technique starts from the source code of the application, testing all parts within the product is in reach. It is no longer possible to overlook obscure parts of the source code as they are an integral part of the internals of the software component that is being tested. [12]

However, white box testing also has disadvantages. For one, test suites can only be developed late in the life cycle of a software component, because the implementation of such parts within the product must have been worked out before any test suites can be developed for them. If test cases are created before the components are fully developed, changes within the software will require the test cases to be rebuilt as well. Furthermore, white box testing strategies require testers to have an in-depth knowledge of the given implementation techniques. As such, deeper understanding of the program and its language is required. In such context, the testers are often also the developers of the validated product. [12]

3.2.2 Black box testing

Unlike White box testing, Black box testing involves looking at the specifications of the application and does not require examination of the underlying code of the program. It is also done from the customer's viewpoint, not the back-end developer. The test engineers engaged within black box testing only know the set of required inputs and expected outputs and are unaware of how those variables are moving within the software. [11]

Black box tests are more convenient to administrate than white box testing as they use the finished application and do not require knowledge of its design. Independent testers outside the development team can administer black box tests to ensure functionality, compatibility, and security compliance. [11]

Unlike White box testing – Black box testing always uses the final product, and therefore does not have sub-categories based on if the application is running or not. Once the code is ready and delivered for testing, this method of tests can be executed. Since the testing method uses external functionalities, the test cases need to be thought of in such a way that as much of the internal functionality is tested as possible. This allows the testing to uncover as many defects as possible in a shorter amount of time.

For example, in a web application with a login system, a black box test task would attempt to login successfully into its system. Other testing steps in this process might include attempts of getting user feedback in failed login attempts, such as not filling the password in the given form or a similar required value. The test case should always have a result, which in this case would be a notification supplied to the front-end user requiring them to input the required values.

Black box testing is a testing strategy based solely on requirements and specifications. It requires no knowledge of internal paths, structures, or implementation of the software being tested. [10] As such, without access to the logical flow of the program, it is harder to confirm that tests cover all scenarios within the code. Furthermore, assigning clear test cases is more difficult as it is not clear whether some tests will be redundant (run the same logic twice) or not necessary at all.

3.2.3 Grey box testing

While white box and black box testing have separate bounds of use, grey box testing, also known as *translucent* testing, allows the tester to focus on all the layers within the application and uses a combination of the previous two testing methods. It means that the tester needs to have knowledge of the internal data structures within the program (white box testing), while also running actual tests on the interfaces that the program exposes (black box testing). [13]

Grey box testing technique increases the testing coverage by allowing the tester to focus on all the layers of the product through the combination of existing white and black box testing methods. In grey box testing, the tester must have knowledge of internal data structures and algorithms, for the purpose of designing test cases. In grey box testing, the test cases are created based on the modules (white box testing), while actual tests are performed on the exposed interfaces (black box testing). [14]

Grey box testing technique combines the benefits of both white and black box testing methods. It allows the tester to design excellent test scenarios due to having knowledge of both functional specification and the interface definition, instead of deep knowledge of one side of the product. It also allows for unbiased testing, as the developer and the tester can be different people, while still having access to the same resources.

Grey box testing also has disadvantages compared to other testing techniques. Even though the tester has knowledge of the design of the product, the test coverage can still be limited since the tester might not have full understanding of the source code. Therefore, it is difficult to test the full flow of the product, akin to black box testing. [14] Test cases are also more difficult to design, as they need to be comprehensive enough to cater to the design of the product and the user interface. Creating test cases using two different sides of the product is more difficult than concentrating on one.

Grey box testing contains parts of the previously two mentioned testing methods. Initially, the tester investigates the application to understand how it has been implemented. Then, with the knowledge acquired from the short examination, chooses more effective black box tests to incorporate for thorough testing results. [10]

3.2.4 Testing method choice

Choosing a testing method for this thesis was substantially difficult. The author of the thesis does have access to the internal design of the web application, however, is not very knowledgeable in it. The author also has access to the test environment, where the application is run. Therefore, the following factors were used to decide on a testing technique

- Testing objective – The objective of the tests is to assess the security of a web applications' configuration processes. This is done by testing the product interface for possible vulnerabilities following the created blueprint of OWASP ASVS requirements. In this instance, black box technique is ideal as it bases on the testing of the front-end view of the application.
- Current web application phase – As the web application and its configuration processes are fully in use and functional, the ideal testing method to test the different actions is to test the interface of the product. As stated in the previous point, black box testing covers testing the app from this side.
- Testers' knowledge and skills – Due to the author's capability of accessing and understanding the source code, it is possible to review the internal design of the code and understand the flow of the program. Therefore, white box testing could allow for a deeper investigation of the application and increase the amount of code tested.
- Time – It is important to decide on a testing method based on the amount of time the tester has. It can be assumed that creating test methods using white box techniques takes more time than black box. This is because white box testing requires deep knowledge of the internal systems of the program, whereas black box testing requires the tester to think outside of the box while assessing the product's user interface. Grey box testing stands in the middle.

With the mentioned factors considered, the author decided to use grey box testing technique in this security assessment. As stated previously, grey box testing involves testing methodologies from both black and white box testing. The choice was done due to the following points:

1. The testing objective is to secure the configuration processes of the web application. Those processes exist on the application user interface. While this can also be done with black box testing, grey box testing is preferred in the upcoming points
2. The tester has access and knowledge of the web application's design. This is a marginable reason why grey box testing was chosen. With access to the program's codebase, it is possible to create better test cases and fully test the product's back end.
3. The author does not have enough time to fully indulge in the full design process of the application. Therefore, it is less time consuming to understand the design partially and assess the security of the product using its user interface. This factor is also incorporated into grey box testing.

3.3 Requirement validation

The purpose of web application security testing is to find any security weaknesses or vulnerabilities within an application and its environment, to document the vulnerabilities, and to explain how to fix or remediate them. There are several types of testing methodologies. These include web application security audits, vulnerability assessments, and penetration tests. [15]

For this thesis, the author decided to analyse the security of the web application by assessing the vulnerability of the program. A vulnerability assessment is a subset of an audit and is focused on finding weaknesses or vulnerabilities within the application. It involves real-time testing and exercises the application components such as all input fields. [15] There are different vulnerability testing tools for enabling the assessment, however, within this thesis, it was done manually.

After the blueprint was made following the chosen security framework and the testing method was successfully selected, it was time to validate if the web application configuration processes comply with the security assessments of the blueprint. To do this, the following steps were taken:

1. Understanding the meaning behind the security requirement – In order to understand how to test, it is important to know what to test. This step is to understand the meaning behind the requirements in the framework. For example, requirement V5.1.3 states “Verify that all input (HTML form fields, REST requests, URL parameters, HTTP headers, cookies, batch files, RSS feeds, etc) is validated using positive validation (allow lists).” The requirement needs to be understood by the tester before it can be reviewed in the web application.
2. Understanding the back-end codebase – Next, it was time to briefly focus on the codebase of the web application’s configuration processes with the aim to gather overall understanding of the web application procedures. This was done to simplify front-end testing as initial review can show faults which are then easily tested in the product’s user interface.

3. Test the application behaviour following the requirement – Following the investigation of the codebase, the next step was to activate the back-end code using the user interface of the web application and test whether it works following the requirement or not. Following point 1 example, the web application would pass the requirement if input passed by the web is validated using positive validation. Sources [8], [16], and [17] were used to help validate the security of the web application based on the requirement list selected earlier.
4. Mark up the testing results – The application was tested until the first fail of a requirement. This means that if the test failed on the first configuration process out of 12, it would not be tested further, and the requirement would be noted as not satisfied. This is to save time and resources as future analysis can be done to focus on one requirement and determine which processes need to be fixed as a result.

4 Analysis and results

The analysis of this thesis resulted in a security assessment regarding the web applications' configuration processes. It is important to understand the outcome of the tests to take further actions regarding the configuration processes of the application.

This topic will first briefly introduce the results. The full list of satisfied and non-satisfied requirements cannot be revealed within this thesis as they are not fully solved when this thesis is publicized. However, initial plans were made to move towards satisfying all the requirements.

Following the results, one vulnerability is taken by the author to fix within the thesis. The vulnerability will be introduced alongside the full analysis of the configuration processes and will end with the solution that was completed.

4.1 Security assessment result

The result of the security assessment shows that 13 requirements had been satisfied by the web application, out of 24. This totals a satisfaction percentage of roughly 54%. It was found that the program uses outdated technologies, such as incorporating hidden HTML forms as data input to the back-end procedures. The results indicate that the configuration processes within the web application can contain some vulnerabilities, as they do not fully satisfy the requirements.

The exact results are unable to be shown in this thesis due to the vulnerabilities not being solved at the time of the publication. However, the CRM team were made aware of the full list of requirements and their results. The weaknesses were noticed, and future development plans were made to increase the security of the various configuration processes.

After the vulnerability assessment results had been finalised, the author decided to investigate one requirement reviewed during the testing in more detail, with the aim to discover a vulnerability within the configuration process that could be mitigated by the author. It was decided that OWASP ASVS point 8.1.3, “Verify the application minimizes the number of parameters in a request, such as hidden fields, Ajax variables, cookies and header values,” will be taken for further examination. The choice of the requirement was done subjectively by the author, and this point was chosen due to the following reasons:

1. The application configuration processes use hidden fields during the CRUD operations that are activated. This means that, without proper precautions, the user might be able to inject custom data into the fields to activate unexpected events, with the possibility of tampering stored data within the CRM system’s database.
2. Since first seeing the design of the application, and after further assessment of the processes within, the author speculated that the hidden variables most likely have a vulnerability associated to them that could be exploited in different ways, bypassing the already done developments that limited user access rights.

The next topic covers the security vulnerability that was identified and the steps that were taken for its mitigation in detail.

4.2 Proof-of-concept solution

The possibility of a vulnerability was first discovered when reviewing the web application configuration processes during the OWASP ASVS requirement 8.1.3 validation. It was found that many parameters are hidden within the various user interface HTML forms that are used to communicate the data to the application. The framework point exercises the thought that such variables should be minimised, however, they are existing on almost every configuration process within the software.

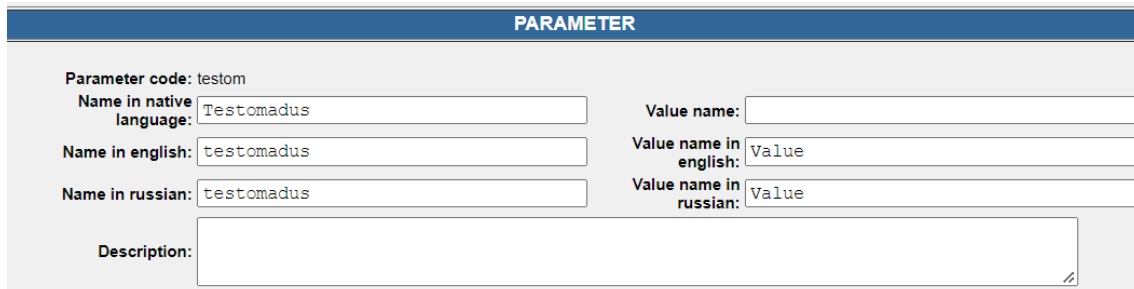
Some developers and testers misunderstand the nature of *hidden* form fields. These are fields invisible on a rendered page but provide additional data when the page is submitted. Relying on the user's ignorance that they will not spot the hidden values is dangerous. [8]

A deeper analysis regarding hidden variables was done and one vulnerability was found by the author. For the vulnerability, a proof-of-concept solution was done as an example of how to use the security assessment results to identify security vulnerabilities within the system. This topic introduces the vulnerability and actions done to mitigate it.

4.2.1 Vulnerability introduction

A vulnerability was spotted by the author indicating that a simple HTML injection on hidden values within a form can be used to manipulate which parameter is being configured at a given time. The back-end procedures ignore whether the parameter that was initially chosen by the user is the same one that is being sent back to the system.

Figure 15 is used to show part of the configuration window, which is shown to the user when any parameter is chosen to be edited. Only the initial part of the table is shown, as the page is only used to access the actions and HTML forms within the response.



The screenshot shows a configuration window titled "PARAMETER". It contains the following fields:

- Parameter code: testom
- Name in native language: Testomadus
- Name in english: testomadus
- Name in russian: testomadus
- Description: (empty text area)
- Value name: (empty text field)
- Value name in english: Value
- Value name in russian: Value

Figure 15 Parameter configuration screen.

The current screen form sends the user multiple hidden HTML forms within the response that are prefilled and later used by the back-end modules during parameter saving. Figure 16 shows the form within the previously described view.

```
<form method="POST" action="/crm_main.r" name="omadus_lisa">
  <input type="hidden" name="g_kasutaja_id" value="" >
  <input type="hidden" name="g_kasutajanimi" value="" >
  <input type="hidden" name="g_seansi_voti" value="" >
  <input type="hidden" name="g_tegevus" value="omadus_muuda0">
  <input type="hidden" name="g_peategevus" value="" >
  <input type="hidden" name="g_isiku_id" value="" >
  <input type="hidden" name="s_omadus_kd" value="testom">
  <input type="hidden" name="valitud_omadus" value="testom">
  <input type="hidden" name="s_sailitada" value="1">
  <centered>...</centered>
</form>
```

Figure 16 Parameter form element in HTML.

From this form, the value used for exploiting the given vulnerability is *s_omadus_kd*. This tag holds the value of the parameter code that the user is currently configuring. When changing this value to a different existing code and attempting to save, the action overwrites the data of the newly inserted code. After the procedures have completed, the user is shown the view where the saved data can be seen, visible in Figure 17.

PARAMETER DATA	
Parameter code: testom1	Value name:
Name in native language: Testomadus	Value name in english: Value
Name in english: testomadus	Value name in russian: Value
Name in russian: testomadus	
Description:	

Figure 17 Overwritten parameter *testom1*.

As shown, the parameter data of *testom* overwrites the database record of *testom1*. This is because the user injected the latter parameter code into the form while editing the former parameter. This vulnerability can be maliciously used to bypass any parameter-based access rights and allows the user to update any parameter in the CRM system, given that they know the parameter code.

4.2.2 Scope analysis

To fix this vulnerability, an analysis was done to see how big of an area this vulnerability covers within the system. As stated previously, CRM web application allows for multiple different configuration options, exactly twelve of them, and it needs to be noted which of these options can be manipulated in the way as described above.

This was done by creating a similar situation in all the configuration capabilities – attempting to save a parameter to a different identifier value within the database than what was provided to the user initially. Table 4 shows the results of the investigation with the following format:

- x – vulnerability spotted
- o – no vulnerability spotted

Table 3 Vulnerability scope result.

Parameters	x	Topics	x
Products	x	Documents	x
Error messages	x	Types	x
Result types	x	Texts	x
Problem types	x	PL conditions	x
Condition types	x	Actions	o

The result of the scope analysis shows that out of 12 configuration options, it is possible to tell the back-end systems where the form data needs to be stored in 11 of them, no matter which parameter was initially given to the user. The action configuration option does not keep track of the currently selected configuration with a hidden HTML variable, and thus is safe from this vulnerability.

4.2.3 Vulnerability mitigation

As described previously, the CRM web application stores user session data in the system's database. Figure 18 shows the user viewing a test text within the application with the database row that contains the user session stored within it when this action is activated.

TEXT DATA	
Text ID: 9070	
Text code: test_a	
Type: VT,KYC	
Name: test_name	
<hr/>	
Title in native language: test_title	
Title in english: test_title1	
Title in russian: test_title2	
<hr/>	
Text in native language: test	
Text in english: test1	
Text in russian: test2	
<hr/>	
vali	String
s_tekst_id	9070

Figure 18 Configuration of texts with current selected identifier in database.

Through discussions with the CRM team, it was decided that the vulnerability can be mitigated by validating whether the data stored on the user session within the database matches the value the user is sending with the hidden values inside the HTML form. If they differ, the user should be given an error. It was proposed that a new variable, named *chosen_parameter*, would be created to keep track of the latest selected parameter within the web application. A new parameter was created as other parameters could be in use or overwritten during other actions, meaning the new solution could intertwine with previous logic. A new session parameter allows the validation to always occur and be correct, no matter the path the user took to the configuration window.

To validate the database session value with the identifier from the user form, a new class was added to CRM's codebase named *CrmValidate*, that can be instantiated with an input value. A method was added to this class that can be used to retrieve values from the current user session within the web application. The last retrieved value is stored within the object and another method can be used to store a logical True or False value whether the two parameters are equal. This class, shown in Figure 19, will be used to validate whether the value that was given to the user is the one as the application is expecting.

```

1  BLOCK-LEVEL ON ERROR UNDO, THROW.
2
3  CLASS CrmValidate:
4      DEFINE PUBLIC PROPERTY userValue AS CHARACTER NO-UNDO INITIAL "" GET. SET.
5      DEFINE PUBLIC PROPERTY expectedValue AS CHARACTER NO-UNDO INITIAL "" GET. SET.
6      DEFINE PUBLIC PROPERTY equalValues AS LOGICAL NO-UNDO INITIAL ? GET. SET.
7
8      CONSTRUCTOR CrmValidate(userValue AS CHARACTER):
9          THIS-OBJECT:userValue = userValue.
10     END.
11
12     METHOD PUBLIC VOID fillExpectedValue(
13         sessionId AS INTEGER, sessionFormCode AS CHARACTER, sessionParamCode AS CHARACTER
14     ):
15         DEFINE BUFFER bSeans FOR crm.seans.
16         DEFINE BUFFER bSeansiInfo FOR crm.seansi_info.
17
18         FIND bSeans WHERE bSeans.seansi_id EQ sessionId NO-LOCK NO-ERROR.
19         IF NOT AVAILABLE bSeans THEN RETURN.
20         FIND bSeansiInfo OF bSeans WHERE bSeansiInfo.vormi_grupi_kd EQ sessionFormCode AND bSeansiInfo.vali EQ sessionParamCode NO-LOCK NO-ERROR.
21
22         ASSIGN
23             THIS-OBJECT:expectedValue = IF AVAILABLE bSeansiInfo THEN bSeansiInfo.vaartus ELSE THIS-OBJECT:expectedValue.
24     END.
25
26     METHOD PUBLIC VOID compareValues():
27         ASSIGN THIS-OBJECT:equalValues = IF THIS-OBJECT:userValue EQ THIS-OBJECT:expectedValue THEN TRUE ELSE FALSE.
28     END.
29
30
31 END CLASS.
32

```

Figure 19 New class for validating instantiated string with user session value.

Figure 20 shows the new error created within the CRM system for this validation. The error is returned whenever the parameter identifier sent by the user does not match the one previously stored within the database. Errors can support multilanguage messages and can be shown to the user based on their language settings. However, the current error is a system error, and it was agreed that it will be initially developed in English.

ERROR DATA	
Error ID:	2421
Base code:	CRM
Base ID:	crm#validation_invalid_parameter
Text in native language:	Unable to validate parameter, please go back and try again
Text in english:	Unable to validate parameter, please go back and try again
Text in russian:	Unable to validate parameter, please go back and try again
Comment:	
Entered by: TERTTU KLASEN 19.04.2022	
<input type="button" value="Add"/> <input type="button" value="Change"/> <input type="button" value="Delete"/> <input type="button" value="Back"/>	

Figure 20 Error created to be shown to user.

Next, the class needed to be added to a configuration screen to be functionally tested. The author chose to first add this solution to the module that is run when parameters are saved, as this was the one where the vulnerability was first spotted. Figure 21 shows the new procedure created for mitigating the vulnerability, *validateSessionValue*. This instantiates the previously created class and compares two values, the one within the user response and the one previously stored in the database. When this validation fails, the procedure returns an error that is shown to the user, following the previously discussed action logic.

```

PROCEDURE validateSessionValue:
    DEFINE VARIABLE oCrmValidate AS CLASS CrmValidate NO-UNDO.
    oCrmValidate = NEW CrmValidate(Get-Variable('s_omadus_kd', b_vorm.vormi_grupi_kd)).
    oCrmValidate:fillExpectedValue(g-seansi_id, 'omadus', 'chosen_parameter').
    oCrmValidate:compareValues().
    IF NOT oCrmValidate:equalValues THEN DO:
        RUN crm_error-find("CRM", "crm#validation_invalid_parameter", g-lang, "", "", "", OUTPUT oVeakood, OUTPUT oVeateade).
        RETURN.
    END.
END PROCEDURE.

```

Figure 21 New procedure added to parameter saving module.

The author then validated the changes by attempting to override the parameter *test_cb* form input with value *test_h*. Figure 22 shows the result of the testing. The user is now given a validation error instead of saving the data to the database row identified by the hidden value.

Figure 22 Validation error when attempting to inject identifier value.

All the saving processes within the configuration use different actions. That means that after the text testing had been successful, the author needed to add the same developments for all the other actions and their incorporated modules. The testing was done on all configuration processes, the code was reviewed by CRM team developers and the mitigation has been successfully added to the web applications development environment. It can be noted that setup configurations can only be done in the development environment, as other processes are used to move the changes to the test and production environments.

4.3 Further steps for security improvement

Following the security assessment results, the author suggests CRM team to do a deeper investigation of the application's configuration processes and analyse the vulnerability scope of the introduced requirements. As stated previously, misconfiguration can have unforeseen consequences when it is not spotted by CRM team developers. It is necessary to ensure a safe environment to introduce capabilities to members from other teams.

The security assessment can also be used as a base for a further investigation of the web application. An example for a requirement analysis was given with the proof-of-concept solution which can also be used for other requirements. The results indicate that many other security necessities could be enforced to the configuration process that would improve the overall security of the system. While it is thought that the set-up processes do not impose a large threat on the system, the base requirements should still be in place.

The author also suggests that the CRM team should analyse the possibility of a rework of their application or moving the various procedures to new software. The current web application managed by the team has been through multiple revisions, with earlier solutions being incorporated as early as the beginning of the 2000's. It is important to evaluate whether the required development time for increasing the security of the application could be better spent creating a new solution that, in addition to the current abilities, would allow the team to incorporate new functions alongside more sophisticated security measures.

In case the current solution is kept, and a deeper security assessment of the whole product is done, it is preferred the application is periodically assessed to ensure that the security requirements are in place and working as intended. It is vital to review the whole solution, not only the configuration processes, to validate the security of the product.

5 Conclusion

The author of this thesis has proved through this thesis that the configuration processes within CRM web application can be manipulated and produce unforeseen consequences as a result. While configuration processes do not impose a big threat on the whole system, CRM team is suggested to further use the requirement results to improve the security level of their web application to avoid any situations that could be caused by exploiting the various processes.

This thesis explored how to validate the security of a web application by following a created checklist using an existing verification standard framework and incorporating different testing techniques. Various sources were used to test and confirm whether the product was up to standards from the basis of source code review and interface testing. A proof-of-concept solution was created to show how the results of the security assessment can be used for a deeper investigation within the program to find vulnerabilities to be resolved.

This thesis challenged the idea that CRM web application configuration processes cannot be manipulated to produce unforeseen consequences following malicious or inaccurate actions. In doing so, this thesis analyses possible vulnerabilities within the product following the aforementioned methodology and reviews the result of the assessment.

The assessment allows the CRM team to further investigate whether their web application is up to security standards. With the configuration processes being reviewed, it was found that multiple security requirements following the OWASP ASVS were not satisfied. The author presented the assessment to the CRM team and final decision must be made if other mitigations will be implemented in the existing configuration processes or a new solution should be created in accordance with the security requirements.

It is important that the security assessment was only done for the configuration processes of the given web application, only one part of its functionalities. The author proposes that CRM team can use the newly acquired information to further review the security of their product, incorporating a larger part of the system. Indeed, while the system is internal and requires special institution rights for access, periodical validation of security requirements within the application should still be one of the main focuses of the CRM team.

References

- [1] P. Truusa, Interviewee, *CRM team lead - CRM terminology*. [Interview]. 2 May 2022.
- [2] F. Monteiro, *Learning Single-page Web Application Development*, Packt Publishing, 2014.
- [3] K. Nõmmik, Interviewee, *CRM analyst - CRM web application design*. [Interview]. 9 May 2022.
- [4] A. Calder, *The Cyber Security Handbook - Prepare for, respond to and recover from cyber attacks*, IT Governance Publishing, 2020.
- [5] M. Souppaya, K. Scarfone and D. Dodson, "Secure Software Development Framework (SSDF) Version 1.1," National Institute of Standards and Technology, 2022.
- [6] OWASP, "Application Security Verification Standard 4.0.3," 2021.
- [7] M. S. Merkow and L. Raghavan, *Secure and Resilient Software*, Auerbach Publications, 2011.
- [8] P. Hope and B. Walther, *Web Security Testing Cookbook*, O'Reilly Media, Inc., 2008.
- [9] Y.-F. Li, P. K. Das and D. L. Dowe, "Two decades of Web application testing - A survey of recent advances," *Information Systems*, vol. 43, pp. 20-54, 2014.
- [10] S. Koirala and S. Sheikh, *Software Testing*, Jones & Bartlett Learning, 2009.
- [11] S. Desikan and G. Ramesh, *Software Testing: Principles and Practices*, Pearson, 2007.
- [12] E. Steegmans, P. Bekaert, F. Devos, G. Delanote, N. Smeets, M. v. Dooren and J. Boydens, "Black & White Testing: Bridging Black Box Testing and White Box Testing," in *Department of Computer Science, K.U.Leuven*, Leuven, 2004.
- [13] S. Shah and B. M. Mehtre, "A Modern Approach to Cyber Security Analysis Using Vulnerability Assessment and Penetration Testing," *International Journal of Electronics Communication and Computer Engineering*, vol. 4, no. 6, p. 48, 2013.
- [14] M. E. Khan and F. Khan, "A Comparative Study of White Box, Black Box and Grey Box Testing Techniques," *International Journal of Advanced Computer Science and Applications*, vol. 3, no. 6, pp. 12-15, 2012.
- [15] R. Lepofsky, *The Manager's Guide to Web Application Security: A Concise Guide to the Weaker Side of the Web*, Apress, 2014.
- [16] J. A. Whittaker and M. Andrews, *How to Break Web Software: Functional and Security Testing of Web Applications and Web Services*, Addison-Wesley Professional, 2006.
- [17] J. Andress, *Foundations of Information Security*, No Starch Press, 2019.

Appendix 1 – Non-exclusive licence for reproduction and publication of a graduation thesis¹

I Hannes Toots

1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis “Security Analysis of the Web-Based Configuration Process at the Customer Relationship Management Team of a Financial Institution”, supervised by Kaido Kikkas and Mikko Maltsaar
 - 1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;
 - 1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.
2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.
3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

15.05.2022

¹ The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.