UCL Université catholique de Louvain

École polytechnique de Louvain (EPL)

Design and Implementation of a Software APL Automation Framework

Analysis of Common Applications

Dissertation presented by
Alexandre D'HONDT , Hussein BAHMAD

for obtaining the Master's degree in
Computer Science and Engineering

Supervisor(s)
Ramin SADRE

Reader(s)
Peter VAN ROY, Muhammad BILAL

Academic year 2015-2016

Abstract

Nowadays, information security takes an increasing place in the world of Information Technologies. We work using software products provided by the enterprise, often assuming that these products are safe for use. More and more often, our work cannot be achieved without using online resources, requiring complex services and then expanding the attack surface on enterprise's systems. One may have noticed in the media these last years some effects of the exposure of enterprise or governmental systems, ranging from information leaks (e.g. top-secret information exfiltration from the NSA by Edward Snowden until 2013) to theft of large amounts of money (e.g. 81 million dollars stolen by hackers to Bangladesh in April 2016), including Denials of Service (e.g. a large distributed DoS on servers in US and Europe in 2014). A large part of these can be explained by an inappropriate or insufficient management of information security and a mistaken risk assessment, whose some are simply due to incomplete or non-existent knowledge about the security of used software products.

In this scope, a large effort was made this last decade in order to control the resources of an enterprise environment. Relying mostly on application white-listing, patch management and privileges restriction, which are the essential strategies to mitigate targeted cyber-intrusions, these products help enterprises mainly to detect (and, if possible, block) intrusions (e.g. aimed to cause data leakage or interrupting systems) but do not emphasize the process of maintaining Software Approved Products Lists in an automated way and as a common base to enforce the control on the various resources across their operational units. As such online systems are aimed to detect and/or block threats, offline systems better suit the need to prevent them. An offline framework, following well-defined guidelines, could fulfil the requirement to sufficiently know about a software before deploying it in a real environment.

In this master thesis, we propose to design and develop a modular and extensible framework in order to automate the greatest possible part of the Software APL establishment process by gathering data from various reference sources (i.e. security guidelines for hardening applications and security controls), by using an infrastructure with particular capabilities and means to gather application metadata and by structuring the collected data into a comprehensive dynamic knowledge base. We also propose to conduct some analysis of common applications based on a scenario elaborated to show the capabilities of our solution.

Foreword

"Security doesn't have to last forever; just longer than everything else that might notice it's gone."

OSSTMM3 [1]

As cunningly stated in this quotation, information security is a constantly evolving field that has to be recursively refined for covering new threats and vulnerabilities again and again. In this scope, we focus on software products (which is, of course, a limited subset of security matters when considering an enterprise network) and their approval in a business environment, that is, how to get the right knowledge on products that the enterprise uses, whether it is about acquired or self-made products.

"Start small, then grow your scope."

James Tarala, SANS Instructor [2]

During this thesis, we started from scratch reading and parsing a myriad of standards, guidelines and manuals. We quickly realized the broadness of the field of information security and we had to find our way through in order to appropriately reduce the scope to software security. We namely noticed that this particular branch of security was still lacking mature standards (e.g. ISO27034 whose two parts out of seven are ready for the time being) and we then parsed a bunch of them and tried to draw the pith and marrow from these varied knowledge sources.

Due to the broadness of the subject, another concern was to target some formations in order to quickly achieve a good level of competence. Unfortunately, our researches led us to the following observation ; there exist various formations for the fields of cybersecurity (i.e. for malware analysts, digital forensics investigators, penetration testers but also for security managers) but, in the field of software security assessment, taking into account what we would like to achieve with our framework, there does not exist a "quick-win" course gathering most of the required competences. That is then a marvellous opportunity to build our own approach, sticking the pieces together like making a jigsaw puzzle.

We must point out the fact that the system we build in support of our framework is only the starting point of an exciting project that (we hope) will live for a long time, continuously enhanced as we get more and more competences. The version we present in this thesis is then only a first release implementing some features interesting for showing relevant experiments. On the one hand, this system is thus still in the state of a proof-of-concept but, on the other hand, it opens a lot of doors to future developments in the form of internships or other collaborations, as already currently foreseen with a Belgian university in September 2016.

Acknowledgements

We would like to express our gratitude to our supervisor, Professor Ramin Sadre, whose comments, advices and engagement through the period of our master thesis helped us to direct our efforts on rewarding matters.

We would also like to thank our readers, Professor Peter Van Roy and Muhammad Bilal, who have kindly devoted their precious time for reading this thesis.

Furthermore, we would like to thank our entourage, for having supported us during this long and laborious process.

Table of Contents

List of Acronyms

ADM	Administrator
APL	Approved Products List
AS	Automation System
BB	BackBone
BE	Back-End
DI	Data Item
DL	Data List
DS	Data Sequence
FE	Front-End component
NU	Normal User
SAPLAFv1	Software Approved Products List Automation Framework
SC	Security Control
SCAPL	Security assessment for Computing APL
SDD	Software Design Descriptions
SE	Search Engine
SRS	Software Requirements Specifications
SU	Security User
VM	Virtual Machine

List of Figures

1 Introduction

"Securing a computer system has traditionally been a battle of wits: the penetrator tries to find the holes, and the designer tries to close them."

Morrie Gasser

Software security is all about finding ways to circumvent attempts to break into systems. Today, there exist many ways to increase the security level of deployed systems, requiring a large knowledge to deal with all its aspects.

From general to particular, this introduction reduces the scope to the field of software security assessment through the problem statement, presents the targeted objectives and learning outcomes and dissects the remainder of this document, that is, what we were able to investigate and perform during our master thesis, using a few conventions to be kept in mind while reading it.

Domain	Software Security Audit & Assessment
Audience	IT Security Managers and Auditors, Software Product Assessors and Experts
Purpose	Report on the security of software products in an exploitable way for the management, based on an automation framework

1.1 Problem Statement

These last years, more and more enterprises and governmental organizations are the target of hackers (e.g. from cyber-crime, other governments, hacktivists or also terrorists [3]). The world of Information Technologies is then evolving towards better security management in order to circumvent efforts of these hackers to reach their goals. Many solutions were developed, especially this last decade, to control IT environments and to be able to detect/block attempts to break into the systems. Simultaneously, many standard organizations try to develop standards and guidelines in order to bring uniformity to the security knowledge and to be able to interoperate at a multi-national level.

In the field of enterprise IT environment, many commercial solutions now exist providing features to keep a deep control on assets but are, in particular circumstances, inappropriate for rigorous security requirements such as these of an accreditation process (aimed to approve the operation of an environment designed to deal with sensitive data), especially for governmental organizations. Moreover, most of the currently developed standards essentially focus on applying security controls during the Software Development Life Cycle, mainly oriented on self-made products, which are mostly not applicable on bought products.

As far as we know, in the scope of information security, there still lacks a convenient open-source solution for coordinating managers, technicians and security assessors in order to approve software products for building a dedicated Approved Products List (APL)[1]. That is what we propose in this master thesis ; we try, first, to trim the material to reduce the scope to what we need, then we establish a framework that we materialize by a system allowing stakeholders to collaborate on choosing and approving the right products and maintaining an up-to-date list of these.

1.2 Objectives

Our objectives are three-fold :

1. State the **background**.
   - A – Review the current literature about IT security and especially software security.
   - B – Search for standards and select relevant ones for exploitation.
   - C – Browse some existing solutions and tools and select relevant ones for exploitation.
2. Build our own **framework**.
   - A – Define the process around the principle of Software APL.
   - B – Select relevant principles, controls, methods and techniques from standards.
   - C – Formulate principles, concepts and mechanisms of our framework with mappings between them.
3. Build the **system** from scratch in support of this framework.

[1] High-level term also used in various sectors of the industry such as goods transportation or construction materials.

A – Design and implement an extensible system for managing the Software APL process.

B – Develop extensions for new security testing automated tasks.

C – Report on fulfilled requirements and features.

4. Perform some **experiments**.

A – Determine the methodology for conducting experiments.

B – Define the controls used for conducting experiments.

C – Discuss experiments results on a few common software products.

1.3 Targeted Outcomes

The learning outcomes are provided by the university and are used as an analysis grid to mention academic disciplines and scientific methods we apply to lead our project. Succinctly, these outcomes are :

1. **Knowledge and skills** : by applying general theoretical knowledge acquired and refined during the studies and the thesis.
2. **Engineering approach** : by using the learned methods and knowledge in the scope of software engineering in order to build and validate an experimental system.
3. **Research work** : by providing background information about software security testing tools, by designing and implementing a new solution tailored to the thesis' objectives and by using the created system to perform experiments in order to assess the security of a sample of software products.
4. **Organization** : by organizing the work for a team of two students and by collaborating with a professional accreditation team from a governmental organization.
5. **Communication** : by applying best practices in the scope of reporting and presenting an innovative project.
6. **Impact** : by providing a public open-source project to the cyber-security community.

Note that more information can be found in our thesis plan, available on demand.

Regarding the skills acquired during our formation at the university, this project mainly applies, directly or indirectly, the following courses :

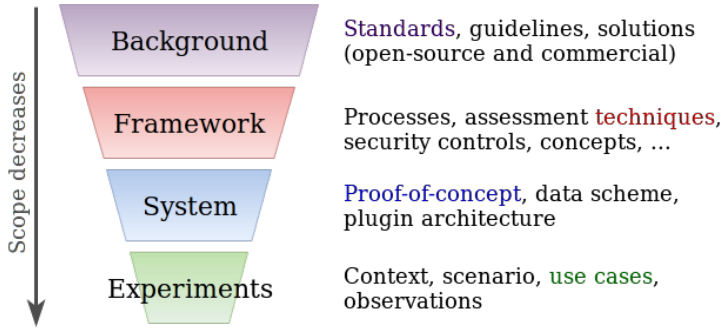
- [**INGI2141**] Computer Networks – Information Transfer : By using the acquired knowledge and tools related to protocol analysis for analyzing sofware products' interactions.
- [**INGI2142**] Computer Networks – Configuration and Management : By setting a dynamic network for automating security assessment tasks.
- [**INGI2144**] Secure Systems Engineering : By applying various assessment techniques such as, for example, password cracking.
- [**INGI2172**] Databases : By applying some advanced knowledge about relational as well as NoSQL databases.
- [**INGI2251**] Software Engineering Development Method : By using best practices from the learned development principles.
- [**INGI2255**] Software Engineering Project : By following the Agile method, to complement the already acquired knowledge about software engineering based on the Waterfall method.
- [**INGI2347**] Computer System Security : By using principles of information security, with the purpose of software products security assessment.

1.4 Content

The remainder of this document is structured as follows :

- **Chapter 2 – Background** provides background information in the field of IT security and especially software security assessment. It explains some relevant standards and guidelines and outlines a few existing solutions, either commercial or open-source.
- **Chapter 3 – Framework** provides the necessary material, picked from the parsed standards, guidelines and various documentation, to build our framework.
- **Chapter 4 – System** presents the system from the requirements to the implementation, including its design and architecture and providing explanations about the current built prototype.
- **Chapter 5 – Experiments** details the methodology used to achieve experiments on the system and relies on a full scenario to validate its working.
- **Conclusion** closes this introduction by presenting a general summary, by parsing the achieved objectives and learning outcomes and by providing ways ahead and ideas for future works.

The main thread is structured so that the scope reduces towards the solution we propose, as described in the figure aside.



1.5 Conventions & Reading Advices

This document is organized such that it can be read mostly using the method presented in the paper titled *How to Read a Paper* [4], presented in the excellent seminar course LINGI2349 [5]. Namely, the first pass can be easily achieved by using the summary and discussion boxes at the end of each chapter.

Why such a layout ?
In order to get this document as much attractive as possible, we designed it with our unusual style, starting each chapter with a cover page providing a quotation from an IT professional and introducing the chapter matter bouncing from the quotation. We hope you will enjoy reading it !

Want to spare time ?
Check the summaries and related discussions at the end of each chapter, they contain information enough so that you can quickly get the main thread !

More focused on Software Security ?
Check chapters **2** and **3** then look at the summaries and related discussions of chapters **4** and **5**.

More focused on Software Development ?
Check chapters **4** and **5** and their related appendices.

2 Background

"Security standards and solutions are a collective effort to deal with individual incompetency."

Hussein Bahmad & Alexandre D'Hondt

Information Technologies have nowadays become so complicated that one cannot get out of trouble without calling on people with miscellaneous specialities. Especially in IT security, the ways an attacker can get into a system have become so numerous that various disciplines and related skills have been developed to manage problems. These last decades, collective initiatives and businesses have grown to provide to the IT community means to overcome security problems that individuals cannot solve without support.

This chapter first presents an overview of the current IT security domains and disciplines, then some standards and guidelines that have become a reference and finally a few popular security solutions, for each section, with a relationship to the topic of this thesis.

Our Approach

We have chosen to present the background hereafter as we think it is necessary to figure out what are the domains and the required skills to address the topic of this master thesis. In particular, we try to select a subset of the IT security to rely on in order to define the framework (in Chapter 3) which is the foundation of the system we propose to build (in Chapter 4). With this in mind, that is why we first present the domains and disciplines of information security, then we scratch the surface of the existing standards, guidelines and security solutions in order to derive the necessary knowledge and material we need to define our framework.

2.1 Scope Definition

In order to frame the scope in a layered way, this subsection first provides some interesting facts and figures about the actual state of software security. It then depicts a view of IT security domains as defined in the Common Body of Knowledge (CBK) [6], a framework of security definitions, concepts and principles proposed by the International Information Systems Security Certification Consortium[1] (ISC)[2] (*"squared"* ; not a footnote) [7]. Afterwards, we give a non-exhaustive overview of the IT security disciplines that underlie these domains. We then focus on the audit and assessment field to finally reduce the scope to the software security.

2.1.1 Facts & Figures

The information presented in this subsection relies on two recent studies from the SANS Institute[2] :

- *2015 State of Application Security* [8] : Sponsored by Hewlett-Packard, Qualys, Veracode, Waratek and WhiteHat Security, involving 435 IT professionals
- *2016 State of Application Security* [9] : Sponsored by Checkmarx, Veracode and WhiteHat Security, involving 475 IT professionals

The surveyed professionals include security operators, managers, developers and system administrators.

Developers and defenders [8]

According to the study of 2015, the top three challenges are for :

- Application Builders :
  1. Deliver software product on time
  2. Acquire sufficient skills to build secure software
  3. Deal with the lack of investment
- Defender Teams :
  1. Identify all applications in a portfolio
  2. Avoid breaking an application by inadvertently modifying production code
  3. Manage communication between security, development and the organization

[1] An international non-profit membership association acting for a safe and secure cyber world
[2] The most popular and largest source of information security training and certification in the world

These challenges reflect the constant trade-off to be found today between development and security professionals. It is not rare today to find on the market software products whose vendors quickly propose security patches as vulnerabilities were immediately discovered after their release because of a botched development even if a software security assessment process exits.

Top challenges in implementing AppSec for systems in production [9]

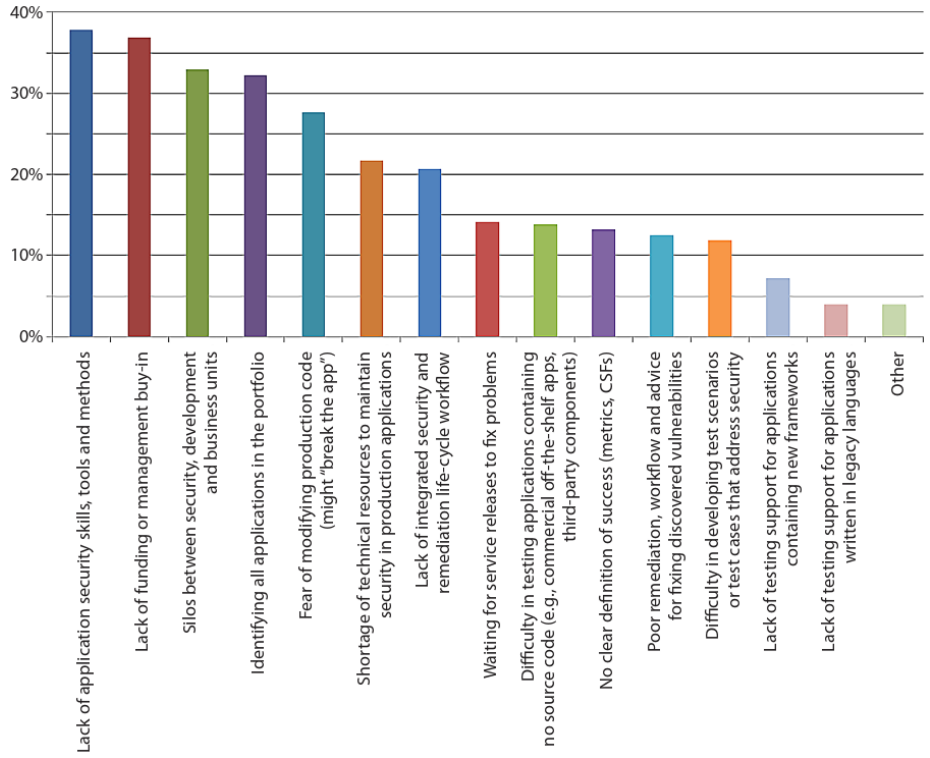


Figure 2.1: Top challenges in implementing AppSec [9]

According to the study of 2016, the surveyed IT professionals gave a top three of the challenges. This provided the trend depicted in Figure 2.1.

It can be mentioned that the lack of skills, tools, methods, budget, communication (between security, development and management personnel) and the application portfolio are the prevalent challenges.

Variety of security standards and models in use [8]

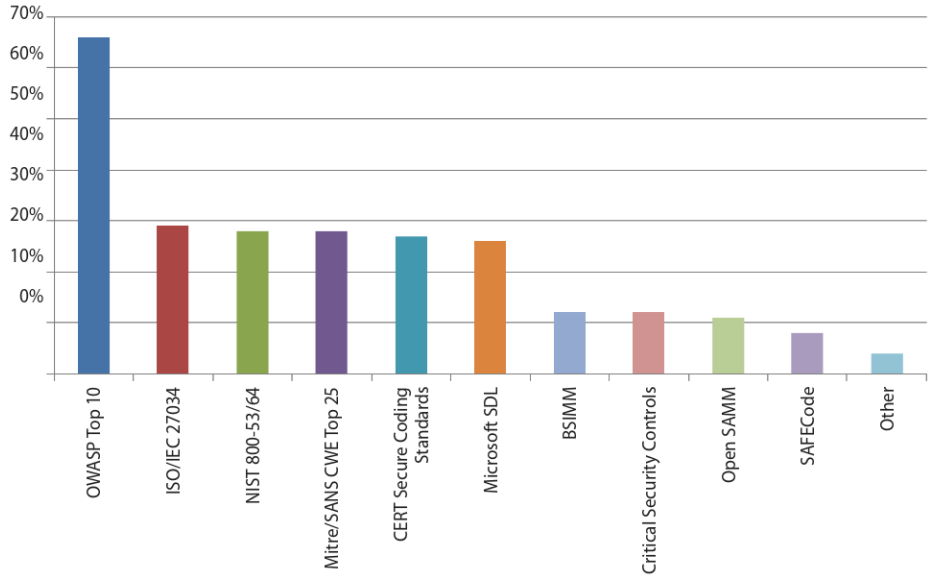


Figure 2.2: Application Security Standards in Use [8]

According to the study of 2015, the developers who took the survey answered about which standard they were following. This is depicted in Figure 2.2.

One can immediately realize the quantity of different existing standards, often related to the type of application. Especially, OWASP[3] Top 10 is focused on the web applications, which represents a significant subset of all the types of applications today but not the entire scope.

Maturity of Application Security programs [9]

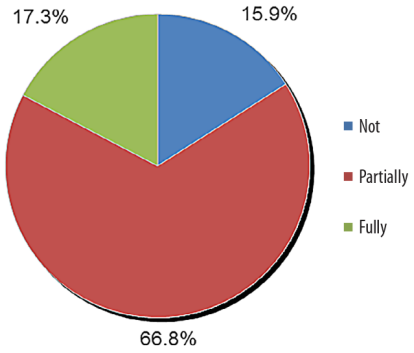


Figure 2.3: Integration of AppSec in Organizations [9]

According to the study of 2016, AppSec programs integration into the organization follows the trend shown in Figure 2.3.

AppSec program integration means how the processes of AppSec are introduced in the security operations. This is considered as a significant measure of AppSec maturity. One can point out in the figure that two thirds of the surveyed professionals are convinced that this integration is only partial up to now against one sixth either for full or non-existent integration.

Application Security resource allocation per type of application [9]

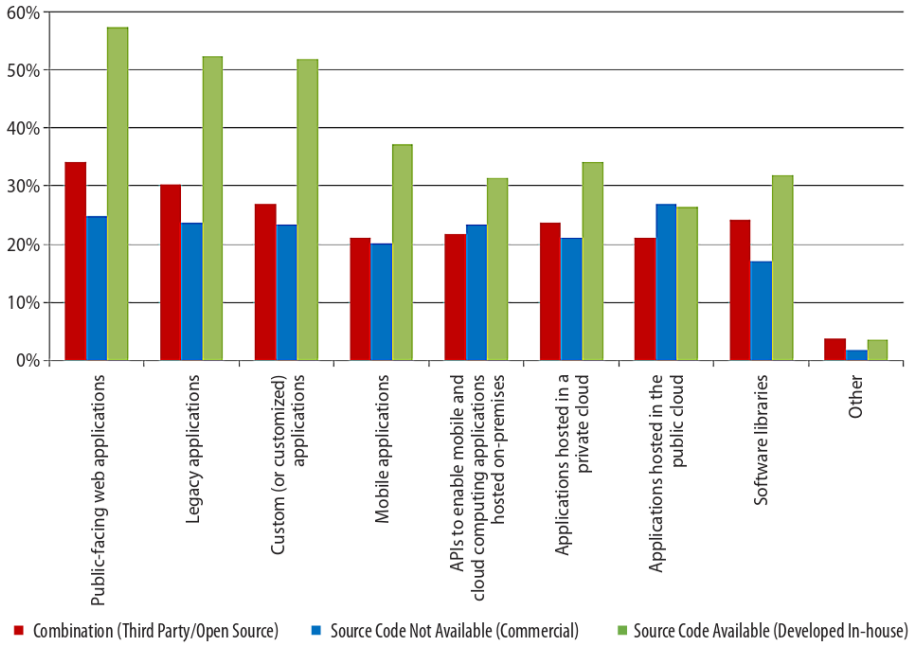


Figure 2.4: AppSec Resource Allocation by Type of Application [9]

According to the same study, AppSec resource allocation into the organization of the surveyed professionals seems to follow the trend shown in Figure 2.4.

This trend indicates how the resources are spent and is significant to reveal the current focus of AppSec. One can immediately notice that some types of application are prevalent and that, for most of these, two times more resources are spent on applications developed in-house than commercial ones.

In a more general way, these facts and figures reveal an overwhelming testimony : the Application Security is not so mature yet, whatever the aspect, namely about identifying and assessing commercial products that the organization uses or integrating related security controls, which are the most interesting points here regarding the objectives of this thesis.

2.1.2 Common Body of Knowledge

IT security domains can be defined in many ways. For this thesis, we consider the domains presented in the CBK of $(ISC)^2$, as it aims to contribute to the common understanding between information security professionals. These domains are the followings : [10]

1. **Access Control Systems and Methodology** : That is a collection of mechanisms allowing a user or administrator to restrict access to a system through Discretionary or Mandatory Access Control (DAC, where the access control decision is made by the user and MAC, where the access control is based on predefined criteria).
2. **Application and Systems Development Security** : This refers to steps and controls used in the development of systems and applications software. This addresses threats, risks and vulnerabilities inherent to the development as well for the developers as for the users.
3. **Business Continuity Planning and Disaster Recovery Planning** : This concerns the plans for a company to deal with disasters, either in case of natural cause or attack.
4. **Cryptography** : This encompasses the concepts, methods and means to treat the information in a way that ensures Confidentiality, Integrity and Availability (CIA).
5. **Information Security and Risk Management** : This addresses the development of policies, standards, procedures, guidelines and techniques to ensure CIA. This includes management tools such as risk assessment and analysis, threat modelling, assets identification and vulnerabilities or also personnel training and awareness.
6. **Legal, Regulations, Compliance and Investigations** : This refers to computer crime laws and regulations but also methods and techniques for investigations and evidence collection and storage.
7. **Operations Security** : This concerns the controls over the hardware, software, information and related operating personnel. Audit, assessment and penetration testing fall into this domain as well as monitoring and maintenance. One can categorize controls as follows : *preventive*, *detective*, *corrective* and *recovery*.
8. **Physical Security** : This addresses the physical protection of the organization's information and resources, the related countermeasures as well as the related administrative and technical controls.
9. **Security Architecture and Models** : This encompasses models that aim to build secure operating systems and networks through concepts, principles, standards and controls enforcing CIA.
10. **Telecommunications and Network security** : This addresses information format, transmission mechanisms, supporting infrastructure and the security measures that ensure CIA.

According to these domains, the topic of this thesis overlaps several domains, mostly :

- **Operations Security** : As we are interested in methods and techniques to control software and its related information.
- **Information Security and Risk Management** : As we are also interested in policies, standards and procedures that ensure CIA around a software product.
- **Application and Systems Development Security** : As the system we propose should include controls up to the source code level of the software products we would like to assess.

In the remainder of this chapter, the presented solutions will essentially fall into one or several of these domains.

2.1.3 IT Security Disciplines

Today, the job market in the field of IT security has diversified and specialized so that it is difficult to get a complete picture of its disciplines. Training institutes have also grown as the companies feel more and more the need to secure their systems. So, in order to get an accurate picture of the current IT security disciplines, one can for example mention the career roadmap of a leader in the IT security training, the SANS Institute : [11]

1. **Penetration Testing / Vulnerability Assessment** : Pentesting aims to gain as much access as possible to the tested systems. Vulnerability assessment is focused on identifying security holes that could allow an attacker to get into a system.
2. **Risk and Compliance / Auditing / Governance** : The objective is to assess and report risks to the organization by applying appropriate controls, procedures, policies and standards.
3. **Network Operations, Administration and Architecture** : This addresses supervision, monitoring and maintenance of the organization network.
4. **Security Operations and Intrusion Detection** : This concerns assets safeguarding, monitoring and surveillance but also situational awareness of the organization regarding the network and endpoints.
5. **Incident Response** : This is the first line for reaction and mitigation when a breach occurs.
6. **Digital Forensics** : This addresses evidence collection and system analysis dealing with the cybercrime (including phishing, espionage, insider threats or fraud).
7. **Secure Development** : This encompasses secure software design and coding in order to provide vulnerability-free products.
8. **IT Security Management** : This includes the leadership and the management of technologies, processes, policies and people, especially regarding the situational awareness.

Note that these IT security disciplines mainly cover nearly the whole domains presented in the previous subsection. The discipline of particular interest for this thesis, related to the three previously emphasized domains, are :

- **Penetration Testing / Vulnerability Assessment** : As the main objective is to provide a framework in which one can instantiate controls based on penetration testing or vulnerability assessment techniques.
- **Risk and Compliance / Auditing / Governance** : As it should be possible, with this framework, to help stakeholders to collaborate on the assessment of the compliance with standards and policies.
- **Secure Development** : As the security assessment of a product can sometimes be performed up to the source code level.

In the remainder of this chapter, the presented solutions will essentially concern one or more of these disciplines.

2.1.4 Audit & Assessment

The term *Audit* is very generic and often applied to the financial domain. Of course, we focus here on IT audits, especially those addressing the security. Nowadays, IT security audit has become a flourishing field, as the need is born from many companies that have already suffered the (often serious) damage caused by hackers (that is, secrets theft, denial of service or various other dramatical consequences), opening a huge market of potential clients for consultancy. But the audit remains a field that is associated to experimented professionals as it requires, to be efficient, a sufficient knowledge of what is being audited and a good capacity to manage a process that implies the application of many best practices, guidelines, standards and so forth.

In many standards or trainings, a distinction is made between *Audit* and *Assessment* ;

– **Audit** is described as the activity of examining and evaluating systems including compliance checking with established policies, standards, procedures and regulations [12]. This implies following a procedure ensuring that everything was checked and is always ponctuated by a report providing information about the vulnerabilities and advices to mitigate them. This can be simpler described by the two following questions :

  What ? This represents the scope and is the first question we have to ask ourselves.

  How ? This relates to the technical controls we will perform and is addressed later after the scope.

  An interesting notion to know about is the *Audit Process*. In the literature, this process can be described in several ways. As it is not absolutely required for the understanding of this thesis, the interested reader can find its explanation in Appendix A as the process presented in [12], slightly adapted with the one described in [2].

– **Assessment** is about examining and evaluating systems in order to search for *gaps that need to be filled* [2]. This generally implies more technical skills to be conducted. This is more an item representing parts of a particular type in an audit than an activity in the same way.

The difference between both is thus a matter of compliance. Virtually anything is auditable from high-level documents (policies, procedures, ...) to the implemented architecture and systems, hence the myriad of audit types namely including IT general controls, application controls, IT governance, IT risk, system development and so forth. [12]

2.1.5 Software Security

Software security can be described as :

> *The idea of engineering software so that it continues to function correctly under malicious attack.* [13]

Regarding the previously presented domains, this essentially overlaps on :

- **Operations Security** : As this implies security controls and situational updates through assessments and penetration testing, either during each phase of a development or on an acquired product.
- **Information Security and Risk Management** : As this also addresses risk management and analysis, threat modelling and vulnerabilities identification and relies on policies, standards, procedures, guidelines and techniques as well.
- **Application and Systems Development Security** : The relationship to this domain is trivial and certainly the main concern of Software Security.

Regarding the previously presented disciplines, this encompasses :

- **Penetration Testing / Vulnerability Assessment** : As it requires various technical skills in order to assess that the developed or acquired software is secure.
- **Risk and Compliance / Auditing / Governance** : As it involves compliance with policies of the organization that uses the software and it implies monitoring of it through dedicated audit processes.
- **Secure Development** : Once again, the relationship to this discipline is trivial and is also certainly the main concern of Software Security.

What is essential to realize is that the security problem of software products is continuously growing as new products gain more and more in connectivity, extensibility and complexity. The interested reader can refer to the first chapter of [13] (specially dedicated to Software Security) to get a philosophical discussion illustrated by examples and references on how the security problem of software grows.

Moreover, in his book [13], Gary McGraw emphasizes the idea that Software Security is still in its early stage, by the time he wrote his book. Since then, standards, guidelines and knowledge have been developed but only begins now to provide complete frameworks to manage the security of software in a really exploitable way at the organizational level, that is, as well for the managers as for the technicians and security professionals.

Gary McGraw also points out that the term *Application Security* has a different meaning for different people. In particular, this term can relate to *the protection of software after it is already built*, thus referring to techniques that imply testing the software as a black box while Software Security should address design security flaws and bugs. In this thesis, we consider the Application Security as a subset of the Software Security, purely relating to a subset of the software products in general, that is, the applications by contrast to operating systems, drivers, libraries, ... no matter what the techniques are.

2.2 Standards and Guidelines

Amongst the standards that were parsed during this thesis, we have noticed some important families from ISO and NIST. In this section, we provide an explanation of each of the selected families, emphasizing the most important standards from these. We also mention some interesting best practices and guidelines that could help to the establishment of our framework.

2.2.1 ISO 27000 Series

The first collection of standards we can mention comes from the International Standards Organization (ISO), in particular the ISO 27000 series [14]. This collection addresses IT security in general, providing frameworks that can be instantiated by the organizations in order to help the management, although not necessarily sufficient for compliance with law and regulations.

ISO 27001 – Information security management This standard is *a specification for an Information Security Management System (ISMS). An ISMS is a framework of policies and procedures that includes all legal, physical and technical controls involved in an organisation's information risk management processes* [15]. Using such a framework relates to strategic, confidence, regulatory and effectiveness reasons [16]. This thus relates to information security in general and encompasses various underlying standards.

ISO 27005 – Information technology — Security techniques — Information security risk management This standard *provides guidelines for information security risk management and supports the general concepts specified in ISO/IEC 27001 and is designed to assist the satisfactory implementation of information security based on a risk management approach* [17]. This becomes more practical but still relates to a high-level analysis based on risk.

The ISO27000 series overview is depicted in Figure 2.5.

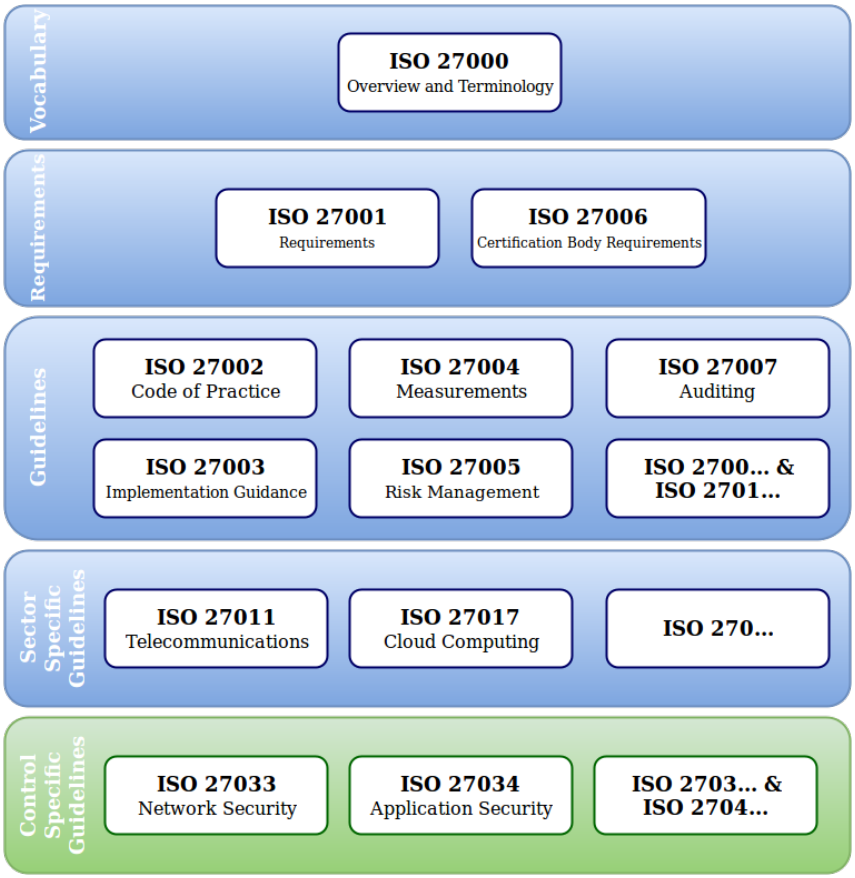


Figure 2.5: Overview of the ISO 27000 standards family

The first layer, *Vocabulary*, simply provides the necessary terminology for the whole family.

The second layer, *Requirements*, provides material to specify the requirements for the ISMS (ISO 27001) and also for audit and certification of organizations by accredited organizations.

The third layer, *Guidelines*, is a suite of best practices and guidance about the methods used in the ISMS.

The fourth layer, *Sector Specific Guidelines*, provides guidance specifically adapted to big sectors such as, for example, health, financial and cloud services.

The fifth layer, *Control Specific Guidelines*, provides guidance in specific control areas of the security such as incident management, application or network security or yet digital forensics.

From this overview and according to our scope, the most interesting standard is of course ISO 27034, dedicated to *Application Security* (which, according to our definitions of Software and Application Security in Subsection 2.1.5, is loosely named *Application Security* instead of *Software Security* as it also relates to the Software Development Lifecycle).

ISO 27034 – Application Security This standard is a specification that *provides guidance to assist organizations in integrating security into the processes used for managing their applications* and *is applicable to in-house developed applications, applications acquired from third parties, and where the development or the operation of the application is outsourced*. [18]

In essence, ISO 27034 provides a main framework with related processes and methods for organizations in order to deal with some challenges as presented in Subsection 2.1.1, especially with the lack of application security skills and methods or also application identification. The interested reader can get interesting information about this standard in Appendix B.

The disadvantage of currently using this standard is that it is not mature yet as only two parts out of seven (whose one was even cancelled) are published (the other parts are in draft). This is thus still under development but will provide, in a near future, a very good foundation for building efficient software security management and control.

2.2.2 NIST SP 800 Series

The second collection of standards we discuss comes from the National Institute of Standards and Technology (NIST), in particular the Special Publications (SP) of the 800 series about *Computer Security* [19]. This collection addresses cybersecurity in general, providing manuals, guidelines but also more informative publications such as studies or recommendations. Note that, for the time being, the new 1800 series is under development to complement the 800 series and will contain practice guides, more tailored to specific use cases or sectors such as financial services or also health records security on mobile devices. This two addressed series inscribe into the NIST documentation as depicted in Figure 2.6.

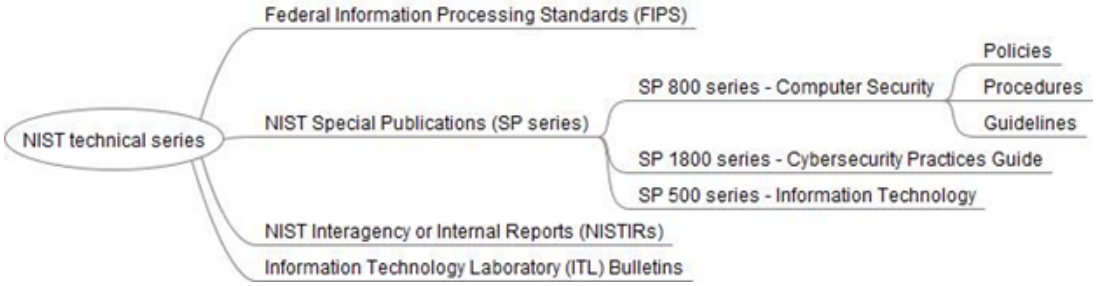


Figure 2.6: NIST documentation structure [20]

NIST SP 800 – Computer Security This family is *is a set of free-to-download documents from the United States federal government, describing computer security policies, procedures, and guidelines, published by the NIST (National Institute of Standards and Technology), containing more than 130 documents* [20].

Among the SP 800 series, the subset of particular interest encompasses a few publications allowing to categorize systems and to define security controls before planning assessments. From the most general to the most particular publication :

1. **SP 800-60 – Mapping of Information and Information Systems to Security Categories** allows to identify information types and systems and to analyze the impact levels on confidentiality, integrity and availability. Note that this publication is based on **FIPS**[4] **199 – Standards for Security Categorization of Federal Information and Information Systems**. This constitutes the first step before diving into security controls and assessment.
2. **SP 800-53 Rev4 – Assessing Security and Privacy Controls in Federal Information Systems and Organizations: Building Effective Assessment Plans** provides guidance in order to define baseline security controls and to assess that these are correctly implemented. Especially, this publication contains appendices with a large list of recommended security controls.
3. **SP 800-115 – Technical Guide to Information Security Testing and Assessment** is certainly the most useful publication for security practitioners as it provides a concrete guide with techniques and methods to conduct software products assessments. The publication contains appendices with useful information such as toolkit descriptions and pointers to other resources.

Hence, regarding the provided information, **NIST SP 800-115** seems to be the best candidate as a starting point for building a framework which can be instantiated by a system. Moreover, in addition to providing technical guidelines on planning and conducting assessments, it addresses the analysis of findings and the development of mitigation strategies. It also contains an overview of the key elements of assessments and testing processes by giving some practical instructions for implementing, designing and maintaining technical information relating to assessment processes and security testing. Another important aspect is that this standard emphasizes specific techniques in this field discussing their benefits, limitations and

[4] Federal Information Processing Standards

recommendations for their use. More information about the assessment techniques presented in NIST SP 800-115 can be found in Appendix C.

2.2.3 ISO 15408 – Common Criteria

ISO 15408 – Information technology – Security techniques – Evaluation criteria for IT security This standard contains *a common set of requirements for the security functions of IT products and systems and for assurance measures applied to them during a security evaluation* [21]. It is divided in three parts providing a general model, functional and assurance requirements.

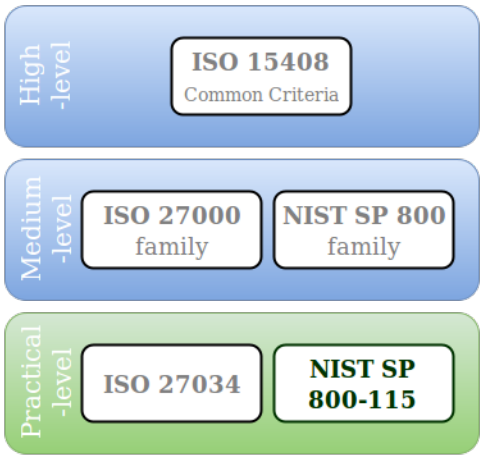


Figure 2.7: Stack of standards per abstraction level

Contrarily to other ISO standards that are purchasable, this particular standard is publicly available but requires heavy implementation. It is more aimed for the organizations that certify products and systems from vendors and the related assurance measures applied on them. As it defines a framework with appropriate terminology and requirements, it could be used in the scope of software assessment but at a high level in order to assess the vendor itself and not any more only software products.

Figure 2.7 depicts an overall view of the standards we already presented in the form of abstraction layers. This emphasizes that ISO 15408 remains an upper layer of interesting standards families and that the lower layer (the practical level), containing the NIST SP 800-115, is the layer of particular interest in the scope of this thesis.

2.2.4 Miscellaneous Guidelines

Australian Government – Information Security Manual (ISM) is a *standard which governs the security of government ICT systems* [22]. This consists of three parts : an *Executive Companion* (providing base knowledge and awareness with a case study about cyber threats), *Principles* (providing general knowledge about information security including e.g. risk assessment and software security) and *Controls* (listing security controls that are related to sections explained in *Principles*).

This suite of manuals provides an interesting complete list of security controls essentially related to the mitigation strategies [23] identified by the Australian Department of Defence (e.g. application whitelisting or use of antivirus). These controls represent an exploitable source in the field of auditing but also as a practical base for assessments in the scope of this thesis. It represents an invaluable source of inspiration.

Open Source Security Testing Methodology Manual (OSSTMM) *provides a methodology for a thorough security test, herein referred to as an OSSTMM audit. An OSSTMM audit is an accurate measurement of security at an operational level that is void of assumptions and anecdotal evidence.* [1]

This security research document provides a scientist methodology for accurate characterization of operational security through the examination of test results in a consistent and reliable way. This manual targets any type of audit including penetration tests, vulnerability assessments and so forth. It can act as a central reference in all security tests regardless of the size of the problem.

Open Web Application Security Project (OWASP) Testing Guide *OWASP is an open community dedicated to enabling organizations to conceive, develop, acquire, operate, and maintain applications that can be trusted* [24].

The test guide it proposes is a real reference in the world of Web application penetration testing and provides a very broad practical guidance.

OWASP is an international organization and a large free and open community, active in the field of Web application security. It provides guidelines as well as articles, recommendations or even various tools available on its Wiki. It also actively supports various security projects developed by the community. If a software product owns a Web server, the OWASP Testing Guide is *the standard de-facto guide to perform Web Application Penetration Testing* [25].

Center for Internet Security – Secure Configuration Benchmarks *describe consensus best practices for the secure configuration of target systems and are developed via extensive collaboration with our volunteer consensus community. Configuring IT systems in compliance CIS Benchmarks has been shown to eliminate 80-95% of known security vulnerabilities. The CIS Benchmarks are globally used and accepted as the de facto user-originated standard for IT security technical controls and are freely available [...]* [26]

In other words, security benchmarks are consensus-based recommended security settings. For the time being, there exists benchmarks essentially for some big products, mostly operating systems (e.g. Windows or Linux). This source is the primary source that auditors use during their information research before conducting an audit. It allows them to perform a first compliance checking with commonly-admitted security settings. This is very useful for determining a baseline before starting assessments so that the findings can be checked against this.

2.2.5 Security Content Automation Protocol

This subsection is inspired from [27] that can be found in Appendix D. In this document, the interested reader can namely find a use case of SCAP. The main information from this document is addressed hereafter.

Since the early 2000s, the Security Content Automation Protocol (SCAP) is an evolving effort from the US department of Commerce through NIST to standardize how the security content is exchanged. The underlying idea is to provide *a suite of specifications that standardizes the format and nomenclature* to facilitate the exchange of security knowledge between security experts in order to automate some parts of the assessment process. [28]

SCAP components can be divided in three (disjoint and complementary) categories : **languages** (for checklists, tests suite and reporting), **enumerations** (of security information) and vulnerability **measurement and scoring** system.

Figure 2.8 depicts an overview of the specification evolution starting on the left vertical frame (SCAP 1.0 to 1.2) to the right. The three categories mentioned above encompass the components in the horizontal frames.

The most essential components, in the scope of this thesis, are :

- **Extensible Configuration Checklist Description Format (XCCDF)** : is a language for process security checklists/benchmarks and for reporting results of their evaluation.
- **Open Vulnerability and Assessment Language (OVAL)** : is a language for representing system configuration information, assessing machine state, and reporting assessment results.
- **Common Platform Enumeration (CPE)** : is a nomenclature and dictionary of hardware, operating systems, and applications.
- **Common Vulnerabilities and Exposures (CVE)** : is a nomenclature and dictionary of software security flaws.

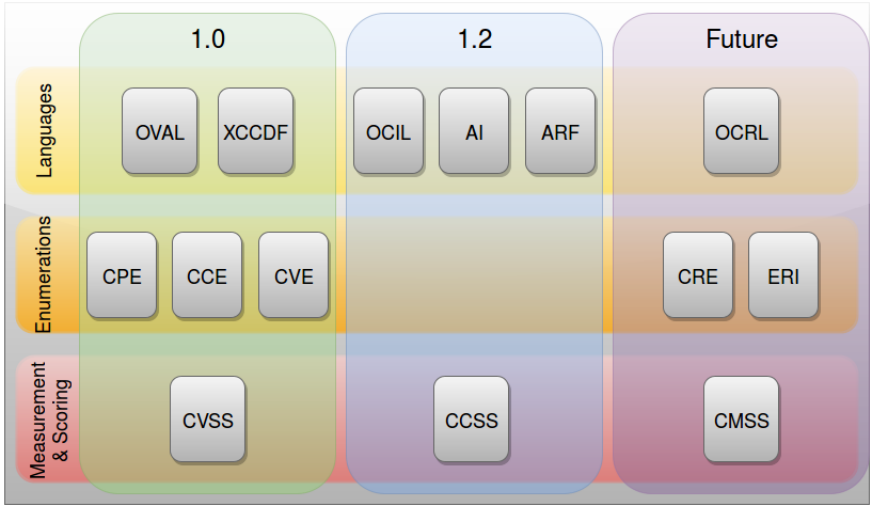


Figure 2.8: SCAP standard

The interested reader can find detailed explanation about most of the terms from Figure 2.8 and an example use case of compliance checking in Appendix D.

NIST SP 800-126 Rev2 – The Technical Specification for SCAP Version 1.2 This publication explains requirements for creating and processing SCAP content, that is, formats for defining CPE's, CVE's, XCCDF tests, OVAL definitions, ... [29]

SCAP was designed to address a large part of the information security management concerns in an automated way, including the following different use cases with their related security control questions :

- **Asset inventory** : *What are the assets in my network ?*
- **Vulnerability assessment**: *Is my system vulnerable to any exploit ?*
- **Patch management**: *Can my system cope with the latest flaw findings ?*
- **Configuration management**: *Is my system configured based on the best practices ?*
- **Policy compliance**: *Can my system comply with the given policies ?*

NIST IR 7275 Rev4 – Specification for XCCDF Version 1.2 XCCDF was originally created to define technical and non-technical security checklists through a standardized format. The main goal behind such a specification is to allow IT experts to define effective, interoperable automated checklists and to support the use of these with a wide variety of tools. XCCDF is an XML specification with two main purposes : *expressing security benchmarks* (the input) and *recording assessment results* (the output). [30]

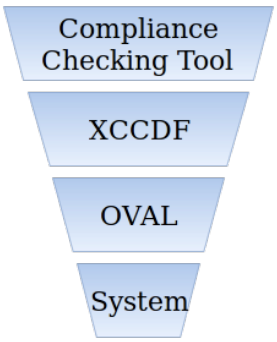


Figure 2.9: Compliance checking with SCAP – Layered model

The compliance checking with SCAP can be modelled as layers like in Figure 2.9. The security compliance tool consumes a benchmark (*XCCDF*) which defines the logical structure of tests by making references to existing OVAL definitions. *OVAL* layer is in charge of performing given tests and formatting a result back to the XCCDF component.

Using XCCDF and OVAL is sufficient to perform security compliance checking. Note that XCCDF is made to be platform-independent in contrast to OVAL.

From these explanations, we can point out that enumerations (CPE's and CVE's) can be easily used in a custom framework to quickly collect security information as several databases currently exist worldwide, exposing interfaces for querying (e.g. NIST NVD [31]). However, concerning CPE's, as these are provided by the security community, they mostly relate to well-known software products (e.g. Microsoft Office), do not cover the whole existing products and thus cannot be used for any case possible. Hence, as the CVE's are linked to CPE's, not every product is covered for vulnerabilities information.

Unless we implement our own tool (which is quite too much time-consuming for this thesis), dealing with XCCDF and OVAL is actually more difficult as, for the time being, there does not exist a tool for any platform that can perform a compliance checking with already defined XCCDF tests and OVAL definitions. NIST provides well a validation tool [29] to use tests and definitions but none of these is provided with it. One can still mention OpenSCAP from Red Hat [32] which is a first implementation running on some Linux distributions (e.g. Debian and Ubuntu) with XCCDF tests and their related OVAL definitions. Unfortunately, there does not exist such a tool for Windows yet, but only a workbench that allows to edit tests and definitions in a user-friendly way.

2.3 Security Solutions

In order to end framing the background, various existing security solutions are also parsed in order to get an overview of what could be usable for our the implementation of our framework. This section is split by category, gathering free dedicated distributions (the platforms), analysis systems, specific security tools and finally commercial-off-the-shelf products. Note that some of them are gathered in a same box as they own the same characteristics but this does not mean that they are bundled or linked.

2.3.1 PenTesting Platforms

Two Linux distributions seem to be the leaders regarding penetration testing ; Kali Linux and Pentoo.

Domain	Operations Security Information Security and Risk Management Application and Systems Development Security Telecommunications and Network security
Discipline	Penetration Testing / Vulnerability Assessment
Purpose	Reconnaissance, vulnerability analysis, wireless attacks, web application attacks, sniffing and spoofing, reverse engineering, digital forensics, exploit creation, ...
Pros	Open-source Community supported Various command-line tools

Kali Linux [33] is a Debian-based distribution created by Offensive Security [34] (another reference in security trainings and certifications) with advanced penetration testing features and tools. This is currently one of the most complete and powerful existing pentesting distributions. It also makes available various command-line tools that are appropriate for scripting and automation. A complete list of the available tools can be consulted at [35]. One can point out the Metasploit framework [36] as the most interesting and sophisticated tool to perform penetration tests and Volatility [37] for digital forensics (essentially the analysis of memory dumps).

Pentoo [38] is a Gentoo-based distribution also made for penetration testing and security assessment with a lot of security tools.

Both of these platforms are very beneficial to the project of this thesis as these provide a set of invaluable tools for assessment and testing. In the remainder of the project, we select **Kali Linux** for these purposes.

2.3.2 Analysis Systems

Among the various available systems designed with a specific purpose in mind, one can mention OpenVAS, Cuckoo and IRMA. These are already used by some organizations essentially in the field of security operations to assess vulnerabilities and analyze malware, two goals that are often addressed in the world of IT security professionals.

	Domain	Operations Security Information Security and Risk Management
	Discipline	Penetration Testing / Vulnerability Assessment
OpenVAS	**Purpose**	[Online] Vulnerability scanning and management
	Pros	Open-source Community supported Interfaceable through OMP

Open Vulnerability Assessment System (OpenVAS) [39] previously known as GNessUs is a framework initially forked from Nessus (another vulnerability scanner) offering several services and tools aiming to provide a vulnerability scanning and management solution. All OpenVAS products are freeware and are mostly licensed under General Public License (GNU GPL). It is important to note that vulnerability scans can be automated through the OpenVAS Management Protocol (OMP).

This can be used in the project of this thesis thanks to the fact that it can be interfaced to provide useful information for assessment and testing.

	Domain	Operations Security
	Discipline	Incident Response
cuckoo	**Purpose**	[Offline] Incident response and malware analysis through sandboxing
IRMA	**Pros**	Open-source Multi-platform Interfaceable through database query
	Cons	Installation and maintenance

Cuckoo [40] is a modular and an open source framework allowing malware analysis. It uses what is called sandboxing mechanism which allows to execute a software in an isolated environment with less risk to the operating system. It currently supports Windows, OS X, Linux, and Android. The following list enumerates some important features allowed by Cuckoo's solution :

- Various file types (besides executables, also Java applets and documents) are supported for analysis.

- The analysis provides a trace of the API calls and gives the general behavior of the given file.
- It dumps and analyses network traffic.
- It offers the ability to perform advanced memory analysis (support for Volatility).

Incident Response & Malware Analysis (IRMA) [40] is an open-source project made by Quarkslab, a cyber-security company. It is designed to help identifying, analyzing and managing malicious files. It allows to keep control over where goes and who gets the data in a controlled environment by faking a network. Each submitted file can be analyzed in various ways with multiple supported antivirus engines or even with own configured probes.

Both of these solutions could be used in the future but these require too much IT system administration skills for setting at this moment. Hence, these are not included in the design of the project of this thesis.

2.3.3 Specific Security Tools

Miscellaneous tools exist with very specific features. Amongst these, we present here two free security tools provided by Microsoft to help the security professionals in Windows environments.

Domain	Information Security and Risk Management
Discipline	Risk and Compliance / Auditing / Governance
Purpose	Attack surface and system state analysis
Pros	Free Command-line tool, thus interfaceable
Cons	Only Windows-oriented from Windows 7

Attack Surface Analyzer (ASA) [41] is a freely available third-party tool developed by the Trustworthy Computing Security group and provided on the website of Microsoft. It is aimed to take snapshots of the system state before and after the installation of a software product in order to analyze if some Windows key elements were changed (thus, increasing the attack surface or not) by making the difference between snapshots. Moreover, it produces convenient HTML reports that can be easily parsed for retrieving relevant security data.

Domain	Information Security and Risk Management
Discipline	Risk and Compliance / Auditing / Governance
Purpose	System state assessment
Pros	Free Command-line tool, thus interfaceable
Cons	Only Windows-oriented from Windows 2000

MicroSoft Baseline Analyzer (MSBA) [42] is a freely available third-party tool developed by Shavlik NetChk Limited and provided on the website of Microsoft. It is aimed to check for the compliance of a Windows system and some Microsoft derived products with commonly-admitted security settings.

These tools are both interesting for the project of this thesis for Windows environments as they address security assessment for common products in an easy and convenient way. Both of them provide command-line tools and, therefore, it can be automated through scripts.

CIS-CAT	**Domain**	Information Security and Risk Management
	Discipline	Risk and Compliance / Auditing / Governance
	Purpose	Compliance checking regarding CIS Security Benchmarks
	Pros	Relies on SCAP
	Cons	Available for CIS members only

CIS Configuration Assessment Tool (CIS-CAT) [43] is a Java-based tool that allows to assess or even audit the configuration of a target system regarding the technical controls described in the related CIS Security Configuration Benchmark (see Subsection 2.2.4) and that also allows to report conformance scores on a scale of 0 to 100.

This tool could be investigated for further use but provided the CIS member fee.

2.3.4 Commercial Off-The-Shelf Solutions

Some big commercial products scaling to the organizational level are mentioned in this subsection as these give an idea of what is already developed on the market and which problems these products solve. These are in general not suited for use in a project as for this thesis.

QUALYS	**Domain**	Operations Security Information Security and Risk Management
	Discipline	Network Operations, Administration and Architecture Penetration Testing / Vulnerability Assessment Risk and Compliance / Auditing / Governance Incident Response
	Purpose	[Online] Assets identification, network and system monitoring, Web application security and vulnerability management
	Pros	Cloud-based (no hardware to buy/maintain, scalable) Compliance with official regulations
	Cons	Cloud-based (dependency to external services) [44] Expensive

Qualys [45] is a cloud based solution (SaaS) for providing security intelligence on demand essentially for prevention. It offers the full spectrum of auditing, compliance and protection for IT systems and web applications. In 2012, Qualys introduces CyberScope [46] reporting capabilities for enhancing its compliance with federal and government agencies in the United States and attains the FedRAMP[5] compliance, which makes it rise in trustability to governments. This is an example of successful attempt to provide remote services for vulnerability assessment and compliance checking. Of course, this does not interest developers of offline solutions unless it provides interfaces for querying relevant information.

[5] A seal of approval from the US government concerning cloud service providers

Domain	Operations Security Information Security and Risk Management
Discipline	Incident Response Security Operations and Intrusion Detection Network Operations, Administration and Architecture Risk and Compliance / Auditing / Governance
Purpose	[Online] Assets identification, network and system monitoring, incident response management
Pros	Cloud-based (no hardware to buy/maintain, scalable)
Cons	Cloud-based (dependency to external services) [44] Expensive Trustability

Carbon Black [47], previously known as **Bit9** is a company that sells a product which provides an endpoint protection and incident response management namely relying on hashing to identify and allow trusted files on systems and workstations. It manages the incident response process following a classical three-steps approach of detection / response / prevention. The firm proposes an interface on demand to its huge database of hashes, allowing developers to use it as an interesting data feed for trustable files but this interface is very expensive.

In 2013, the security firm Bit9 was hacked allowing to a malicious third party to illegally gain temporary access to one of their digital code-signing certificates. This stolen certificate may have been used to illegitimately sign malwares and thus silent the entire system by considering these as trusted files.

Domain	Operations Security Information Security and Risk Management
Discipline	Incident Response Security Operations and Intrusion Detection Network Operations, Administration and Architecture Risk and Compliance / Auditing / Governance
Purpose	[Online] Operational intelligence, network and system monitoring, incident response management, log management, vulnerability management, patch management, compliance checking
Pros	Cloud-based (no hardware to buy/maintain, scalable) Powerful data processing SCAP compliance with some primitives such as CVE and CVSS
Cons	Cloud-based (dependency to external services) [44] Expensive

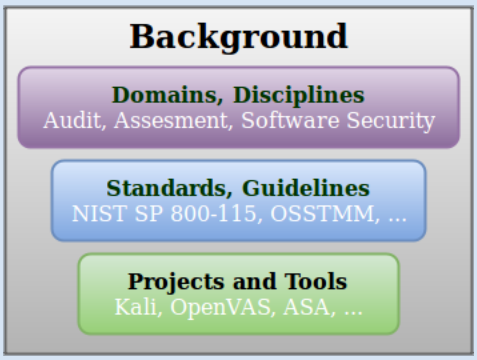
Splunk [48] is an American multinational corporation which provides a comprehensive solution based on big data aggregation and processing using a self-made language called Search Processing Language (SPL), that is, collecting organization's systems data to parse and sort it in a way that allows the organization to efficiently task its operational services for surveillance and monitoring.

SUMMARY

According to reliable security surveys related to the software security [8] [9], some **facts and figures** show that :

- There exists a complicated **trade-off** between challenges of **developers and defenders**. Indeed, it is hard for developers to fulfil their objectives without influencing these of the defenders and vice versa.
- The top **challenges** essentially represent the lack of skills, investments or methods in order to build efficient security.
- **Various standards** exist and are employed by many IT professionals, sometimes in very specific fields.
- The **maturity** or allocated resource related to application security programs **is considered poor**.

The figure aside depicts the matter parsed in this chapter.



On a general point of view, Information Security is divided into **multiple domains**, whose these of interest for this thesis are mostly *Operations Security*, *Information Security and Risk Management* and *Application and Systems Development Security*. In terms of skills, all these domains relate to a few **disciplines** among which *Penetration Testing / Vulnerability Assessment*, *Risk and Compliance / Auditing / Governance* and *Secure Development* are relevant for determining the scope of this thesis. This thus encompasses a **large amount of skills**.

Audit is what one can practice when it comes to compliance checking of systems against laws, regulations, policies, standards or guidelines. This is mainly used in financial companies but also a lot in information security. It can be considered as a business object aimed to report information and advices to the management for finding and remediating security gaps. **Assessment** differs from the audit as it relates to the activity of checking a particular control, with no matter of whether it is related to a law, regulation, policy, standard or guideline. In fact, it can be seen as an activity being part of an audit.

Now that these notions are in mind, it is pointed out that **Software Security** is the right field that encompass the scope definition in terms of domains and disciplines and that the **related standards**, guidelines and procedures are still in their **early stage**. It can also be noted that there could exist an amalgam between Software and **Application Security**, whose the second **refers to already-built software** products in some literature. For this thesis, Application Security represents a subset of the other, concerning only applications that run upon a system, and the rest of the set encompasses operating systems, drivers, libraries and so forth.

Regarding the **existing standards** and guidelines, the ISO 27000 and NIST SP 800 series address the management of risk and vulnerabilities of software products and even provide techniques and methods to perform assessments. Unfortunately, these are very high-level and difficult to implement. However, **NIST** has published its **SP 800-115** [49] about testing and assessment techniques which seems to be exploitable. Moreover, some test methodologies exist such as the **OSSTMM** [1] and pre-made security controls are provided in manuals such as the Australian **ISM** [22].

Amongst the **existing security solutions**, many open-source projects are potential candidates for use, e.g. **Kali Linux** [33] or for integration in a larger system, e.g. **OpenVAS** [39], and some third-party tools are even freely available for some particular environments such as **ASA** [41] and **MSBA** [42] for Windows.

Discussion

One can point out that some **facts and figures** are still **overwhelming regarding the implementation of software security strategies** when we realize the quantity and diversity of applications that are running on the systems of any organization. Despite the myriad of existing standards and guidelines, how can it be that this matter is not overcome yet ? Surely that is a question of skills, investments and various other challenges that every enterprise is facing. In all of this, we want to build something that could take advantage of what already exists and we then start by delimiting our scope.

Among the existing domains and disciplines, we want to point out that **Information Security** is now **very diversified** and that **multiple specialities** exist, among which Software Security overlaps multiple ones. In our thesis, we essentially require skills in vulnerability assessment and maybe a bit of audit in order to start our project. Note that, even though the framework we want to build should be related to already built as well as self-made products in every sense in the future, we choose to **focus**, for this thesis, **on already built applications** (that is, we do not care about operating systems, drivers, libraries, ...).

So, according to the definition of an **audit**, we can now use it to simply answer the **two basic questions** that define the scope of this thesis :

> **What ?** We focus on **Software Security**, that is, we want to figure out how to analyze the security of an application, e.g. Adobe Reader.
>
> **How ?** We thus need to **follow methods and techniques to assess** the security of a software **product while not necessarily understanding** its **whole working**. How can we do that ? In order to understand our approach, please carefully read the next paragraphs then rendez-vous at Chapter 3.

Several powerful families of standards exist but the **most reasonable approach** remains to **select** these of the **NIST** Special Publications, especially the **SP 800-115** which provide us **practical guidance** regarding security assessment techniques so that we can study which of these could be automated or at least systematized through a collaborative platform allowing stakeholders to cooperate in an efficient way. Among other standards and guidelines, we can start working with the **ISM** for selecting **security controls**. We do not select any other one at this moment for a question of time even though it is already foreseen in a near future after this thesis.

The major drawback of standards and guidelines presented in Section 2.2 is that, for most of them, these are often high-level and very tedious and costly to implement as they are not understandable for uninitiated people and require high skills and experience. Moreover, most of these documents are aimed for a management audience and do not necessarily fit to practitioners as we are in the scope of this thesis.

While other very interesting standards or guidelines exist, we choose to keep these in mind for future developments and choose to **currently focus only on the NIST SP 800-115**. Note that, regarding SCAP, we are still enhancing our skills on how to use the suite of specifications and, furthermore, open tools for compliance checking according to these are still under development. Therefore we hope that we will be soon ready to integrate this matter into our solution.

Among the **existing solutions**, except the commercial-off-the-shelf products which are of course unusable for us for an obvious question of cost, we can still **select a few ones for attempts to automate or systemize** their working in the scope of assessments, e.g. Kali or OpenVAS. The next chapter will try to bring to light the concept of what we want to build.

3 Framework

"The mantra of any good security engineer is: 'Security is a not a product, but a process.' It's more than designing strong cryptography into a system; it's designing the entire system such that all security measures, including cryptography, work together."

Bruce Schneier

Information Security is a process, that is, a set of measures that provides more assurance that bad things would not happen, yet letting the good things work as expected. The same idea also applies for Software Security. In this thesis, we attempt to gather such a set into a framework.

This chapters starts by defining the framework, whose central principle is the Software APL process. We thus deduce sub-processes and provide the concepts, principles and techniques of the one of interest, the *Security Assessment*. We then explain some essential mechanisms and technologies that will allow us to provide to the framework its automation characteristic.

Our Approach

In this chapter, we define our framework using pieces from the parsed standards and guidelines that we stick together. We then explain techniques, mechanisms and technologies that will provide it its automation characteristic. This will be the theoretical foundation, as a first version, from which we will deduce the requirements, design and architecture of the system we propose in support of this framework.

The framework, called **Software Approved Products List Automation Framework (SAPLAFv1)**, is mostly a process accompanied by some principles, techniques, methods and mechanisms, just like a toolbox with its instructions of use. This is aimed to produce Software APL reports providing evidences that a software is safe-to-use with a certain degree of assurance. The goal is to gather the necessary material in a single set to allow the APL process to be implemented.

In order to keep the framework as simple as possible on a management point of view, it only relies in the first instance on the standard NIST SP 800-115 explained in Subsection 2.2.2 and the ISM explained in Subsection 2.2. Future versions of the framework should incrementally include elements from OSSTMM and the CIS Benchmarks (see Subsection 2.2.4) then, once mature, a useful part of the ISO 27034 explained in Subsection 2.2.1.

3.1 Software APL Framework

This section presents the essential foundations of the framework we propose. We first depicts the process we propose to manage the Software Approved Products List with the notions of actors and status.

3.1.1 Main Process Definition

The management process is defined as a four-step process and is represented in Figure 3.1. The output of this process is a list of product reports, that is, the famous Software APL.

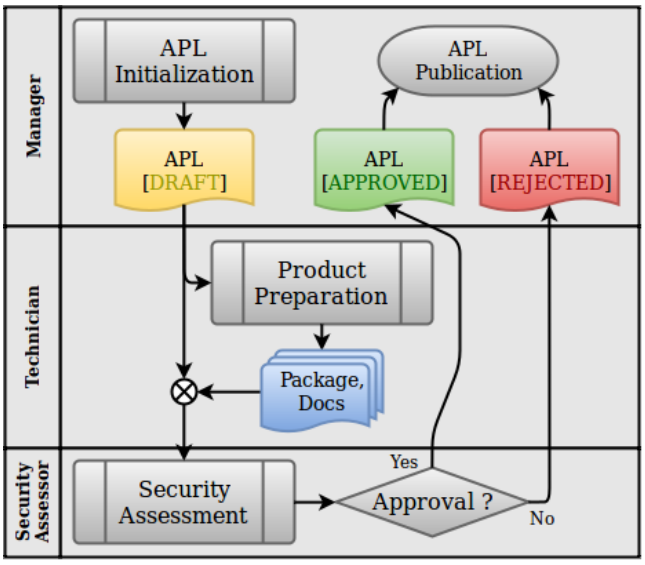


Figure 3.1: Software APL Management Process

Each product report is called an *APL file* and holds the necessary information about its related software product, filled in through the process as explained hereafter.

The main elements in this representation are the **actors**, **sub-processes** and **files** with their **status**. The actors with their responsibilities are described in the next subsection. The other elements are explained just after the figure.

Note that this process encompasses a bit more than simply the security assessment. It also addresses Software Asset Management (SAM) information in order to define and document the software product. Various standards and guidelines exist for this purpose but this is not addressed in details in this thesis as the aim is to focus on the security aspects.

1. An *APL file* is initialized (sub-process *APL Initialization*) by a *Manager* who prepares the basic information of the software product, typically related to a SAM method. At this stage, its status is DRAFT.
2. The *APL file [DRAFT]* is sent to the *Technician* as well as the *Assessor*. At this point, the *Technician* performs the *Product Preparation* in order to provide an installable package to the *Assessor* and to fill in useful information such as installation notes, applied settings... While the *Technician* is preparing the product, the *Assessor* can already perform first researches of security information.
3. The packages and documentation are then sent by the *Technician* to the *Assessor* and the real assessment job can start. The sub-process *Security Assessment* is then executed. Once done, this leads to the decision (only up to the *Assessor*) to approve/reject the product providing motivations, limitations of use and/or recommendations in the *APL file*. At this point, the file becomes in status APPROVED or REJECTED.
4. Finally, either the *APL file [APPROVED]* or the *APL file [REJECTED]* is published by the *Manager* so that it joins the Software Approved Products List.

Note that, once the *APL file [DRAFT]* and the package and documentation are sent to the *Assessor*, he/she can still decide to send it back for revision if it does not satisfy the required data (which is not represented in Figure 3.1 not to overload it).

3.1.2 Actors

The **actors** (or also called roles) of the APL process consist of **three categories**, as depicted in Figure 3.1 : the *Manager*, the *Technician* and the *Security Assessor* (or simply called *Assessor*). Note that, in the real world, any of these actors could refer to a team (e.g. *Technician* could represent a technical service in charge). The following table shows the responsibilities for each role associated to an actor.

Role	Responsibilities
Manager	• Management of the entire APL process, *APL Initialization* sub-process • Constant awareness of the status of APL files • Provision of software product identification data into the APL files • Provision of organizational policies to the assessor (e.g. regarding updates) • If required by the assessor, revision of (badly filled in) APL files • Submission of the APL files to the Technician and the Security Assessor
Technician	• *Package Preparation* sub-process (e.g. downloading the installer, writing a script to automate installation and configuration) • Configuration according to management-defined policies (e.g. implementation of updates, particular security settings) • Provision of software product installation and configuration data (including technical documentation and manuals) into the APL files • Submission of the updated APL files and the packages and documentation to the Security Assessor
Security Assessor	• *Security Assessment* sub-process • Assessment report management (archiving, ...) • Provision of assessment conclusions data into the APL files • Submission of the updated APL files back to the Manager

3.1.3 Status

So, according to Figure 3.1, we can distinguish the **three types of status** described in the following table :

Status	Description
DRAFT	Information is filled in up to the technician level providing software product identification, installation and configuration information.
APPROVED	The APL file is complete, contains the security assessment conclusions and eventually limitations of use and/or recommendations for hardening the settings or mitigation measures to avoid risk.
REJECTED	The APL file is complete and contains the motivations for explaining why this product should not be used. It can also contain recommendations for an alternative product.

In the remainder of this chapter, we only focus on the security aspects and the *Security Assessment* sub-process, not on the other sub-processes that could rely on many other non-security standards. In a future version of the framework, these will also be detailed with pointers to their respective underlying standards and guidelines.

3.2 Security Assessment

This section presents the *Security Assessment* sub-process included in the main process with the notions of perimeters, security controls and testing levels.

But first, we mention two definitions from the NIST SP 800-115 [49], which are a foundation of what follows in the next subsections :

> **Security assessment** : The process of finding an effective way in order to ensure that an entity being assessed meets predefined security objectives.
>
> **Assessment object** : An entity being assessed, e.g. a system, a network but also a person, a procedure and so forth.

3.2.1 Perimeters

The *Security Assessment* sub-process can be divided in multiple areas, each owning particular controls according to their scopes. These areas can be defined as **three perimeters** and are represented in Figure 3.2, each encompassing the lower ones.

Note that the representation in Figure 3.2 does not necessarily fit to any situation as, for example, a software product could be a set of multiple systems spread across the network environment, like for example if the product relies on a client-server infrastructure. However, in such a case, multiple APL files can be made like they were independent interacting products, then providing a dedicated comment in the files to point out the relationship. Fortunately, in general, many vendors often make a suite of products instead of providing a single product gathering multiple systems, thus explicitly determining how the APL file(s) should then be organized.

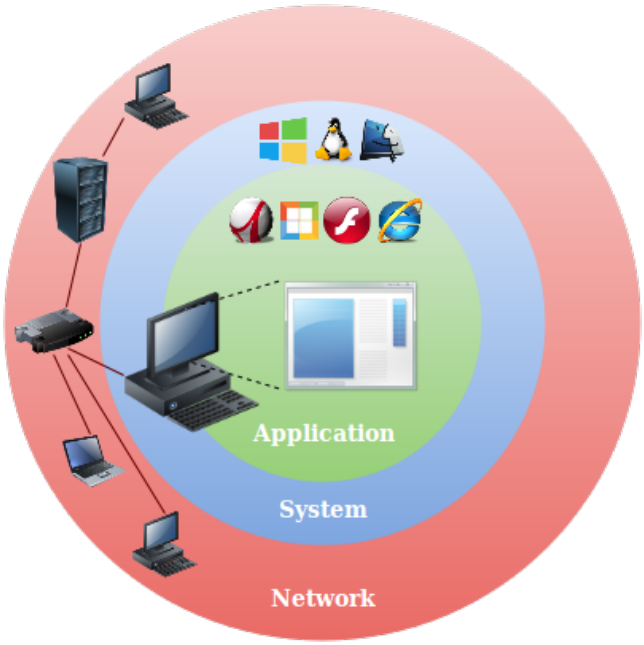


Figure 3.2: Software Security Assessment Perimeters

1. **Application** (on a particular system ; e.g. Adobe Reader, MS Office, Flash Player, MS Internet Explorer) : This encompasses all materials supporting the working of the tested software product, that is, its executable, libraries and particular settings in the installation environment (e.g. registry settings in Windows that are dedicated to the product).
2. **System** (on the same particular system ; e.g. Windows, Linux, Mac OS) : This includes all materials of the installation environment affected by the software product (e.g. called system libraries and drivers).
3. **Network** (interactions with other systems) : This gathers all possible interactions of the software product with other systems (e.g. if the product is a web server, interactions will occur with browsers from other systems).

3.2.2 Security Controls

In order to be consistent, the *Security Assessment* sub-process must rely on various Security Controls (SC). But what is a SC ?

Security controls are technical or administrative safeguards or counter measures to avoid, counteract or minimize loss or unavailability due to threats acting on their matching vulnerability, i.e., security risk. [50]

The security controls are measures that are applied in order to grow the assurance that the controlled asset reaches an appropriate level of security. Regarding the aspect of *level of security*, note that ISO 27034 [18] already addresses it through the concept of *Level of Trust* which is closely related to the control depth. Furthermore, in this standard, two levels of SC are defined : Organisation (OSC) and Application (ASC). More information can be found in Appendix B. So, in our framework, this notion of SC could be enhanced in the future with the concepts of ISO 27034 once this will be more mature.

Amongst the various standard appendices and guidelines providing controls (that can be possibly defined in any form or according to many different categorizations), we choose to use a selection from the ISM [22] (see explanation in Subsection 2.2.4) as it uses a clean and easy-to-use structure. From this “toolbox", we focus on the Chapter *Information Technology Security* and define our scope of SC according to the relevant categories and subcategories. The following tables list these and shows which perimeters[1] they apply on by category.

Note that, in reality, we do not present any formal SC here, as it could confuse the reader due to the broadness of the possibly definable controls. The most important idea is to provide a categorization so that the main matter of each (sub)category can give a view as complete as possible of the potential controls.

[1] Abbreviations : App = Application, Sys = System and Net = Network

Category	Subcategory	App	Sys	Net
Product Security	Selection and Acquisition	✓		
	Classification and Labelling	✓		
	Installation and Configuration	✓	✓	✓
	Maintenance and Repairs	✓	✓	✓
	Sanitization and Disposal	✓	✓	✓

This category relates to the base information about the product. It relies on information provided by the *Manager* and the *Technician* (for example, based on Software Asset Management and Configuration Management procedures). The three last subcategories cover all the perimeters as they relate to the full operation life cycle of a product, that is, the **installation**, **usage and maintenance** and the **uninstall**. Indeed, for example, a product can affect the system at the installation or even let artefacts once uninstalled.

The *Assessor* applies controls such that he/she can determine if the product was made according to best practices and standards and that nothing unexpected happens (e.g. malware download and installation during product's installation) during the phases of software product's operation life cycle.

Category	Subcategory	App	Sys	Net
Software Security	Operating Environment		✓	
	Patching	✓	✓	✓
	Software Development	✓	✓	✓
	Database Systems	✓	✓	✓

This category relates to the technical information of the product, mainly provided by the *Technician*. This provides the relationships between the product and its operating system[2]. This also includes information on how the product is to be updated (either for new features or interface changes as well as for security updates mitigating known vulnerabilities). This last information concerns any perimeter as a product could use its own update mechanism, rely on a service of the operating system or also be updated through the network. The development subcategories namely address the threat modelling (which is a matter out of the scope of this thesis) and security measures already included with the product, thus relating to any perimeter as a threat could lead to the exploitation of a vulnerability through the product itself as well as through a vulnerability of the system incurred by the installation of the product or also through the network if the product has network-related features. This also encompasses possible database systems in support of the product, concerning any perimeter as it could be managed inside the application, by the system or through the network.

The *Assessor* applies controls such that he/she can determine that the product is safe for operations in the given environment regarding its design, update and interactions with supporting systems.

Category	Subcategory	App	Sys	Net
Access Control	Identification, Authentication and Authorization	✓		✓
	Privileged Access	✓	✓	✓
	Event Logging and Auditing	✓	✓	✓

[2] Note that the assessed product can be an operating system itself, but this is not addressed in details here

This essentially encompasses infor- mation collected by the *Assessor* during his/her evaluation. This information (or a part) can also be submitted by the *Technician* but it requires testing to ensure that access control measures are well implemented. It relates, for the authentication, to *Application* and *Network* perimeters as an end-user could connect to the product through a built-in authentication scheme as well as through the network if the product exposes an interface (e.g. a Web application). *Privileged Access* relates to any perimeter as the product could require rights and permissions (e.g. on installed files) managed from the application, the operating system or even also another system in the network (e.g. Active Directory). Logging and auditing also concerns any perimeter as different log exploitation mechanisms exist, either managed by the product itself, by the operating system (e.g. Windows Event Logging) but also in a centralized way on a dedicated system in the network environment.

The *Assessor* applies controls such that he/she can ensure that no end-user can get unexpected access or privileges on the tested product and supporting systems. Moreover, he/she verifies that accesses are monitored through logging and that the resulting logs are collected and used.

Category	Subcategory	App	Sys	Net
Cryptography	Cryptographic Fundamentals	✓		
	Cryptographic Algorithms	✓		
	Cryptographic Protocols	✓		✓
	Transport Layer Security			✓
	Secure Shell			✓
	Internet Protocol Security			✓
	Key Management	✓	✓	✓

This category encompasses all cryptographic measures that ensure the CIA[3] of the data. From *Cryptographic fundamentals* to *Cryptographic Protocols*, the controls relate to questions like *is encryption used to protect the data ?*, *is the used algorithm strong enough ?* or *what are the protocols in use ?*. The next three sub-categories address secure protocols that should be used in the network environment. The last sub-category relates to PKI[4] concerns. The required information can be provided by the *Technician* but should also leverage some testing by the *Assessor*.

The *Assessor* applies controls such that he/she can ensure that the data is appropriately protected and that available cryptographic materials are sufficient and correctly implemented. If sensitive data like passwords or private keys have to be stored, it should be ensured that it cannot be retrieved because of too weak cryptographic measures (e.g. passwords should not be stored in clear).

Category	Subcategory	App	Sys	Net
Network Security	Management			✓
	Design and Configuration			✓
	Service Continuity for Online Services			✓

This category mostly relates to products requiring network communications and therefore only applies on the *Network* perimeter. *Management* and *Design and Configuration* consist of measures applied to ensure that the product operates in a secure way in the network environment. *Service Continuity for Online Services*

[3] Confidentiality, Integrity and Availability
[4] Public Key Infrastructure

relates to measures that can overcome or at least reduce the impact of some particular risks such as a denial of service.

The *Assessor* applies controls such that he/she can determine how the product is managed, how it interacts in the network environment and to what extent it could resist to a denial of service.

Category	Subcategory	App	Sys	Net
Cross Domain Security	Gateways			✓
	Firewalls			✓
	Web Content and Connections			✓

This category clearly also addresses network concerns related to information transfers and is, in particular, applicable to network environments that require a high level of assurance regarding the security of their data. The first sub-category encompasses measures that ensure appropriate internal routes for the connections and that no data could transit through an unexpected path. The second sub-category gathers controls ensuring that only required connections are possible between the systems and that no other path could be used to get into these. The third sub-category addresses the transit of Web content and the related controls such as if a proxy server is used. All these sub-categories relate to information that can be provided by the *Technician* (as, generally speaking, this role also encompasses system and network administration) but should be carefully verified by the *Assessor*.

The *Assessor* applies controls such that he/she can identify how the data transits and that it effectively transits in a secure way to the expected destinations. In the scope of the assessment of a product, this could for example lead to recommendations on Windows firewall rules or new settings to be applied on a proxy server being part of the network environment.

Note that the presented controls categories are a subset of the ISM [22] (see explanation in Subsection 2.2.4) as this manual addresses more security concerns than what we require to define our framework (which does not mean that we consider that this is complete but it should be incrementally refined, hence the name *SAPLAFv1* as a first version). The point here is not to get a complete framework at a first glance but well to define a substantial base for selecting assessment techniques and try to automate the related tools.

3.2.3 Methodology & Methods

At this stage, we introduce some notions of NIST SP 800-115 [49] that will help us to formalize the testing levels presented in the next subsection. The interested reader can find more information about this standard in Appendix C.

Methodology The methodology for any assessment consists of the following phases :

1. **Planning** : The assessment should be planned as any other project by defining the scope, objectives, team roles and responsibilities and so forth. This phase is critical to the success of security assessment.
2. **Execution** : The goal of this phase is to identify vulnerabilities and validate them when appropriate. This includes activities related to chosen techniques.
3. **Post-Execution** : This phase is aimed to determine the root causes of identified vulnerabilities and eventually addresses mitigations and recommendations.

Assessment Methods We define three different methods :

- **Testing** : The method that compares actual and expected behaviors of one or more assessment objects when exercising under specified conditions.
- **Examination** : The method of inspecting, studying or analyzing one or more assessment objects in order to facilitate understanding in order to obtain evidence or clarification.
- **Interviewing** : The method of conducting discussions with individuals or groups within the same organization in order to facilitate the understanding in order to obtain some clues or clarifications.

3.2.4 Testing Levels

As far as we know, the notion presented in this subsection does not appear in any of the standards we have parsed even though it could be considered as a level of trust within the meaning of ISO27034-1 (see Appendix B). In order to regroup assessments to be performed in comprehensive sets, we define five testing levels, summarized in the following table. Note that each level encompasses controls from the lower levels and that the descriptions hereafter take assessments methods presented in the previous subsection into account.

Level	Description
0 Waiver	This happens when the organization decides to approve a software product **without** any **assessment**, especially when the vendor is considered as trusted or presents some security-related certifications regarding its development process. Hence, this level has a rank of 0. Of course, this should be avoided in practice but can be used, for example, if other products have higher priorities. However, the absence of assessment does not mean that the approval decision is not up to the *Assessor* ; it is always him/her who decides.
1 Basic Search	This level is the most succinct one in term of assessment, requiring **no test environment**. The *Assessor* only performs **examinations** and eventually **interviewing** with the *Technician* in order to control if the **documentation** regarding the product is sufficient (e.g. applied update policy, data sheet or technical manual) in order to decide its approval.
2 Blackbox Testing – Offline	This addresses **testing and assessment** of the product as a blackbox (so, when the source code is not available) installed in a **simulated environment** managed by the *Assessor*, hence the term *Offline*. This environment should be set up as similar as possible to the real one in order to make the controls consistent. From this level, the *Assessor* mostly performs **testing**, the examinations and interviewing being addressed in the lower level.
3 Blackbox Testing – Online	This level relies on a test environment or even the production environment managed by the *Technician*, hence the term *Online*. This environment is thus set up so that it is very **close by or actually the production environment**. It often requires more coordination for the *Assessor* to operate as the *Technician* could not be able to provide the necessary hardware deployed and ready-to-use like in the real environment even though, in some cases, it could be possible.

4 Whitebox Testing	With the test or production environment, this level also includes the analysis of the product as a whitebox, that is, also **reviewing the source code**. This is definitely the best level one can achieve in term of software security assessment but obviously requires lots of efforts to be conducted with accuracy and completeness.

3.2.5 Techniques

This subsection relies on NIST SP800-115 [49], presented in more details in Appendix C. The reason why we have decided to follow this standard regarding techniques is that it provides, in the form of a technical guide, some interesting guidelines on planning and conducting technical information security assessment and testing.

On can mention, as an example in a totally different scope than ours, a successful implementation of NIST's assessment and testing techniques in [51], a methodology document made and applied in the field of nuclear power plants. The reason why it is so interesting to mention is that it provides a series of techniques with, every time, their associated tools that the IT security professionals use in this field. This is clearly a quick-win in terms of tool allocation to assessments as it is already approved and used in a sensitive field.

Categorization According to NIST SP800-115, the assessment techniques are divided into **three categories**, described in a logical order (progressively covering all assessment methods as defined in Subsection 3.2.3) :

1. **Review Techniques** : This encompasses the review of multiple assessment objects such as documentation, logs, firewall rulesets and system configurations. This aims to search for potential weaknesses in organization's policies, network infrastructure design and foreseen security measures in place. This mainly concerns examinations, interviewing and a bit of testing.

   List of techniques : *Documentation Review*, *Log Review*, *Ruleset Review*, *System Configuration Review*, *Network Sniffing*, *File Integrity Checking*

2. **Target Identification and Analysis** : This addresses devices, ports and services identification using automated tools. This aims, through the analysis of the collected data, to identify potential weaknesses in organization's network infrastructure and implemented security measures in place. This mainly concerns testing and a bit of examination, i.e. network scanning.

   List of techniques : *Network Discovery*, *Network Port and Service Identification*, *Vulnerability Scanning*, *Wireless Scanning*

3. **Target Vulnerability Validation** : This aims to validate the existence of vulnerabilities by exploiting them in order to measure and understand the exposure of the attacked devices. This is essentially manual work but some tools can be automated. This mainly concerns testing, i.e. password cracking and penetration testing.

   List of techniques : *Password Cracking*, *Social Engineering*, *Penetration Testing*

Note that, for the *System Configuration Review* technique in general, assessors manually verify that system settings are correctly configured relying on checklists or security configuration guides. Several repositories of checklists related to IT product configurations exist, namely NIST Checklists [52] and USGCB [53]. Assessors just have to compare found settings with the recommended ones from the related checklists. At the end of this process, assessors can then report if the settings meet an expected security level or not. SCAP, explained in Subsection 2.2.5, can also be used in order to automate the compliance checking based on settings baselines.

Relationship with the Testing Levels According to the *Testing Levels* defined in Subsection 3.2.4, the use of these techniques obviously starts from level 1, that is, from a *Basic Search*. This level encompasses Review Techniques and overlaps on a part of Target Identification and Analysis. The next levels address every technique, but with a progressively more complete usage and inside a more and more accurate testing environment according to the level (with even also the source code available for the Level 4).

Mapping with Security Controls Regarding the SC subcategories presented in Subsection 3.2.2, a mapping can now be performed between a selection of SC's referenced in the ISM [22] and some assessment techniques as presented in the following table. Note that, as it can be seen in the bottom of this table, some techniques are more general and can be mapped to a lot of controls.

Technique	Subcategory	Sample Control ID's	Control Description
Network Sniffing	Network Design & Configuration	1006	*Security measures should be implemented to minimize the risk of unauthorized access to network management traffic on a network*
Documentation Review	Product Selection & Acquisition	0463	*Agencies must check product evaluation documentation, where available, to determine any product specific requirements*
Ruleset Review	Network Design & Configuration	0576	*Agencies must develop, implement and maintain an intrusion detection and prevention strategy*
Log Review	Event Logging & Auditing	0859, 0991	*Agencies must retain event logs for a minimum of 7 years after action is completed in accordance with the NAA's Administrative Functions Disposal Authority* and *should retain DNS and proxy logs for at least 18 months*
Password Cracking	Identification, Authentication & Authorisation	0417, 0421, 0422	*Agencies must not use a numerical password (or personal identification number) as the sole method of authenticating a user* and *agencies using passphrases as the sole method of authentication must enforce* organization's passphrase policy
Social Engineering	Identification, Authentication & Authorisation	0976	*Agencies must ensure users provide sufficient evidence to verify their identity when requesting a passphrase reset for their system account*
Network Discovery	(Any)	(Any)	[No specific description] Can be applied on any network-related control
Penetration Testing	(Any)	(Any)	[No specific description] Can be applied on almost any control

Mapping with Tools NIST SP800-115 also proposes a list of tools for applying some techniques. However, one should note that most of these tools are only supported in the Linux operating system and thus, for example, not on Windows. However, various tools are aimed to analyze captures or dumps of information offline. Hence, these tools are well good candidates for being scripted so that they can be automated.

Furthermore, the proposed list is mostly outdated and relates to the former version of Kali Linux (discussed in Subsection 2.3.1) which was called BackTrack. The interested reader can refer to Appendix E for our overview of some usable tools in the scope of our framework, namely some ones that are described in Section 2.3.

3.2.6 Test Environments

As mentioned in the previous subsection, available tools like presented in Appendix E can be a problem regarding the operating system. Indeed, if a Windows software product is to be tested, ASA and MSBA (discussed in Subsection 2.3.3) could be used to retrieve exploitable assessment information but this would not be valuable for Linux, of course. This then imposes that we build different test platforms according to which operating system the tested software product runs on.

However, in order to figure out how the test environment should be designed, we must distinct the various tools according to their purpose and regarding the perimeter they act on. We can then define :

- **Online Internal Tools** : Acting at the *System* perimeter, they have the problem that they are platform-dependent. These tools should be installed on a test platform with the software product to be assessed. We mean, for example, tools that dump memory, system settings or perform file integrity checking or also log events. Note that a tool that listens to a network interface for recording traffic will be considered as an *Online Internal Tool*, while acting at the *Network* perimeter. This means that, in reality, there is a slight overlap between perimeters regarding the concept of online internal and external tool.
- **Online External Tools** : Acting at the *Network* perimeter, these encompass the external systems that are aimed to perform scanning or testing platforms like Kali Linux (discussed in Subsection 2.3.1) that provides operational tools for exploiting found vulnerabilities. For example, a vulnerability scanner belongs to this category as it will scan the system owning the tested software product from the outside.
- **Offline Tools** : This class of tools does not depend on the perimeters as they are used to analyze either system or network data collected by *Online Internal* or *External Tools*. However, these tools can be hosted on a testing platform like Kali, with the *Online External Tools*. For example, a log analyzer will parse log files collected by an event logger or a traffic analyzer will rely on a packet capture recorded by a sniffer.

Studying each possibly automatizable tool requires a lot of time for testing and scripting. That is why we do not opt for performing such studies and concentrate on a generic way in which we could call a scripted tool from a central system, that is, a plugin architecture, as we will present in Chapter 4. Concerning the study of various tools, this will be addressed during an already foreseen internship for a student of the University of Mons (see future works in Subsection 6.4) during the first semester of the academical year 2016–2017.

The best way to design a good test environment is to make the platform resemble as far as possible the one used in a production environment. Of course, it is not always possible and some tools must be installed, possibly affecting some settings, services or other items that could be related perhaps critical to the tested product. This is why it should be made use of built-in operating system features as much as possible.

For example, Windows provides PowerShell, a scripting language that allows to check multiple Windows objects, especially useful in the scope of security assessments. A PowerShell script can then be pushed on the test environment and run its code to retrieve data that can be analyzed offline, without affecting the normal working of the operating system. In the scope of this thesis, this language requires too much skills for processing at this stage. But this is part of our future development.

3.2.7 Sub-Process

Now that we have defined perimeters, security controls, testing levels, tools, test environments, assessment methods and techniques, we can now define the *Security Assessment* sub-process. This is depicted in Figure 3.3.

Figure 3.3: Security Assessment Sub-process

The steps of this sub-process are the followings :

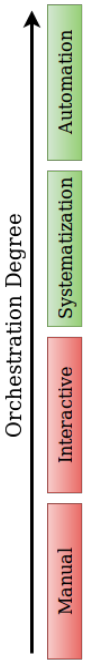
1. The **Product** is first identified. At this stage, the base information is received from the *Manager* and the *Technician* and the package (if relevant) and documentation are also provided.
2. From this information, the **Perimeters** can normally be determined according to what is currently known about the working of the software product. Note that the *Assessor* may consider not necessary to conduct an assessment up to the *Network* perimeter, even if the product has network interactions (i.e. for a question of time, simplicity or priority on other products).
3. Once the perimeters are determined, the **Security Controls** can now be selected. At this stage, these controls will not all necessarily be used. The selection is up to the *Assessor*.
4. According to what is already known on the product, the *Assessor* can determine the **Testing Level** that is required. It then filters the determined set of security controls.
5. According to the testing level, the **Test Environment** is set up or not, or a coordination is taken between the *Assessor* and the *Technician* to come and perform the assessments on a test or production environment (if relevant).
6. Each foreseen **Assessment** is then executed according to the phases mentioned in Subsection 3.2.3 and a report is made with the findings. Motivations for rejection, limitations of use and/or recommendations are deduced and provided in the APL file.
7. The decision of **Approval** is then taken and communicated to the management, with the APL file in support.

3.3 Orchestration Technologies

As the base concepts and principles of our framework are defined, we can finally describe mechanisms and technologies that could be used to orchestrate automated tasks in the system to be implemented upon the framework. We first describe some possible sources of data feeds that will allow us to retrieve relevant information for an APL file. We then explain how we can automate these feeds in an asynchronous way. We close this chapter with the virtualization that allows to easily build test environments.

3.3.1 Modes of Operation

Before going further, it is necessary to make the difference between the possibilities of information retrieval tasks ; we distinguish four modes of operation :



1. **Automation** : This means that the task is repeatedly executed, each time providing a final result, handled, formatted and exploitable as is. That is what we are trying to implement as much as possible. For example, if we use an external system to identify vulnerabilities of a Web server being part of the assessed product, we consider it is about *Automation* if the returned result is exactly the data we include in the APL file.

2. **Systematization** : But sometimes, the result provided by the task requires handling before being exploitable (e.g. if the task may not return the desired result while still providing an expected one) but no processing mechanism exists to perform the right handling so that we must still refactor the result by hand. For example, if we use a search engine based on some keywords, we get a list of possibly desired results while all are expected (because of the way the engine works). This introduces the notion of **suggestion**, that we will use soon. This is a possibly desired result, with no certitude that it is actually the desired one.

3. **Interactive** : This concerns the tasks that provide no result but offer an interface to the user, for example, to a test environment. This mode of operation implies tasks to install and configure the interface.

4. **Manual** : This mode is trivial, we only consider the tasks that simply cannot be orchestrated and require a human input.

Note that the presented scale uses two colors ; green for the orchestrated tasks providing a result and red for those which require a human interaction.

We consider the principle of orchestration as the scale of all modes of operation. So, from here, the reader should not be confused between orchestration and automation or any other mode.

3.3.2 Data Sources

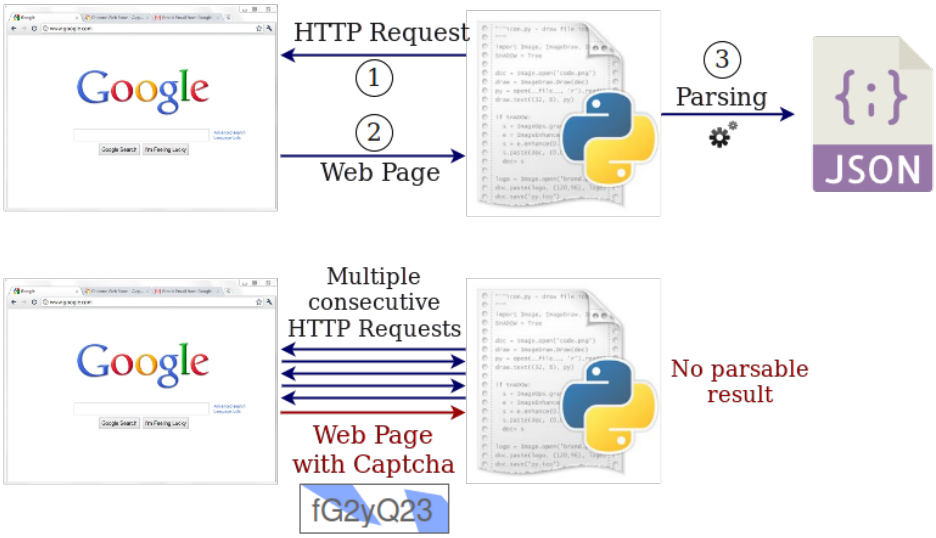
Many information can be found using various sources. The most obvious ones are surely search engines. Once retrieved, the information is often raw and needs to be handled (filtered, aggregated, reduced, ...). This can then sometimes lead to data mining through advanced techniques for extracting an exploitable result from a data set. External systems can also be used and allow to leverage pre-formatted results, specifically made to be ready-to-use.

Search Engines There exist many available Web search engines that help to retrieve information from the Internet. They return results, containing references to pages as well as various file types, referred to as Search Engine Results Pages (SERP). Among the most famous engines, one can surely mention Google, undisputed leader in this field. For the results to be easily exploitable, it is better if the used engine exposes an API[5] so that we can script a request and get a response formatted in accordance with what the producer defines in the API. It can be noted that, like any self-respecting engine, Google provides such an interface.

Anyway, as even a simple Web request can be sent by a script to the engine exactly as if it was a browser (as depicted on the topmost picture in Figure 3.4), it is easy to parse the SERP's in HTML format for converting it into another one (e.g. JSON[6]) more commonly handled with the scripting language. However, using this approach can have a drawback as many engines nowadays tend to prevent robot-like scripts from working

[5] Application Programming Interface
[6] JavaScript Object Notation

more than a few consecutive requests, then responding with a Web page containing a captcha[7] for checking that the interacting entity is a human. This behavior is depicted on the downmost picture in Figure 3.4.



In the normal case, the request/response provides a Web page that must be parsed for conversion into another format, more easily exploitable.

Basically, for a search engine, the result item we want is a triplet *(hyperlink, title, description)*.

Figure 3.4: Web Requests with Google

Search engines can provide various results, depending on their algorithms. In the case of Google, the PageRank ensures that the list of results presents the most relevant lines first. It is difficult to evaluate (and out of the scope of this thesis) to what extent Google returns a relevant list for any software product provided a defined set of keywords (including a characteristic such as the system requirements) but we can mention that, after a bit of manual testing, the following keywords scheme seems to be a good trade-off, provided that we consider at least the top-5 of the resulting list.

Keywords scheme : *vendor product_name product_version [characteristic]*

However, a problem arises regarding particular characteristics ; these are often not provided by the vendor or these are filled in but each vendor calls it differently. For example, if one types *"CTI Text Encryption" 6.0.1 "version history"* (this is an encryption tool which is not so common), the result is very poor, but not because of the search engine, just meaning that the desired result does not exist as the characteristic is not provided by the author. In such a case, the information can just be reported as unavailable.

Regarding what precedes, the result can be refined such that, if the desired information does not exist, the engine will return an empty set. This can be achieved first by using quotes to separate each important set of keywords (e.g. if the product name has multiple words) but there exist other particular techniques such as, in the case of Google, dorks [54] which are advanced search operators allowing to target more efficiently the desired information. One can namely mention :

- *site* : reduces the search scope to the given website or domain
- *filetype* : reduces the search scope to the given file format
- *intext* : allows to search for keywords inside Web pages

The previous keywords scheme could then become, for example :

Keywords scheme : *product_name product_version [characteristic]* site:*vendor_website*

Status In order to implement a *systematized* research, we can either implement a module ourself or use an existing one. *GoogleScraper* [55] is the perfect candidate for this purpose. It supports requests to many engines, namely Google, Bing, DuckDuckGo or also Yandex.

[7] Completely Automated Public Turing test to tell Computers and Humans Apart

EXAMPLE

Search engine : Google

Keywords	Popularity	Top-5 *
Microsoft Office 2010 "system requirements"	Very common	12345
Microsoft Office 2010 "version history"		12345
PDFForge PDFCreator 1.7.0 "system requirements"	Common	12345
PDFForge PDFCreator 1.7.0 "version history"		12345
"CTI Text Encryption" 6.0.1 "system requirements"	Uncommon	12345
"CTI Text Encryption" 6.0.1 "version history"		12345

* Color code :

Green: relevant

Orange: related to the product but the results do not relate to the characteristics

Red: weakly relevant or irrelevant

Data Mining This is *the analysis of (often large) observational data sets to find unsuspected relationships and to summarize the data in novel ways that are both understandable and useful to the data owner*. [56]

The main goal of data mining is to extract knowledge and patterns through information processing from a data set, such that this can be given a non-obvious meaning that can support a decision process. This is generally achieved in several steps through what is called the Knowledge Discovery in Databases (KDD) process, including a selection of data, a pre-processing, eventually some transformations, the mining itself and the validation of the results. [57]

> **Status** This implies more advanced techniques of data processing and is clearly out of the scope of this thesis. However, it should be planned in a near future to study such techniques in order to refine information researches. This could allow to improve systematizated researches to *automated* ones.

External Systems These are a particular source of interest in our work. We want to provide a system that allows to plug external systems such that their outputs are usable together. This does not mean that all the collected information is aggregated but well that this is gathered in a single location, allowing considerable gains of time. By external system, we mean solutions like discussed in Subsection 2.3.2. Note that managing external systems in the infrastructure of our own system can be tedious and existing solutions should be carefully chosen for integration such that these can be maintained without too much efforts. This can possibly reduce the set of integrable solutions.

> **Status** In order to implement *automated* or *systematized* tasks, we must rely on a plugin architecture that allows to handle the external system's output such that our system can process it. *CVE-Search* [58] is an example of a perfect candidate for this purpose. It retrieves a list of CVE's (see Subsection 2.2.5), that is, vulnerability enumerations for a given software product, and supports several output formats (e.g. JSON or HTML). Moreover, its installation is straightforward and does not imply heavy maintenance.

Public Data Repositories and Databases These are also an interesting source of information but often rely on pre-formatted data such that their use requires particular skills. This is, for example, the case for

the NIST NVD [31] discussed in Subsection 2.2.5, which requires a base knowledge of how to handle the collected information according to SCAP's components.

> **Status** This source is currently handled in our framework through *CVE-Search*, the external system discussed in the previous paragraph, which namely relies on NIST NVD to feed its database (other repositories such as *vFeed* [59], *Exploit-DB* [60] are used by this project). This should be implemented as an *automated* task.

Now that we have defined a few data sources, we still lack an underlying mechanism so that we can orchestrate tasks. The next subsection addresses this problem by explaining asynchronous tasking.

3.3.3 Asynchronous Tasking

As mentioned previously, the strength of the system we want to develop in support of our framework is in its ability to interface existing projects that perform certain security tasks in our place, then growing the security scope coverage. So we need to collect and arrange the various output provided by various open-source projects that our solution is based on.

Requirement Some security tasks are time and resource consuming. That is why we need an asynchronous mechanism in order to perform long computations without waiting operations end such that the system can launch several security tasks at the same time relying on a callback mechanism to push the output back to the user.

But, in order to achieve that, the first great challenge is : *What can be orchestrated or not ?*

In order to prove the necessity of automation, one can mention a Research Report from IBM of 2014 [61] stating that, according to a study, 95% of all security incidents involve human error. So, reducing manual operations could help to improve this overwhelming observation.
In terms of automation, a study has shown that only 30% of the security controls from the ISO27001 [62] could be automated [63] but it should be noted that this standard encompasses many high-level controls, not only including the controls we want to orchestrate in the scope of this thesis, that is, these of the ISM [22].

Orchestrable Techniques Regarding what was defined in Subsection 3.2.5 and the forementioned study [63], it is obvious that not every technique can be orchestrated. In the particular scope of our thesis, several criteria must be considered to evaluate if a technique is orchestrable or not :

- *Does an existing project already addresses this technique ?* We are searching for quick-wins and are conscious that we cannot implement everything ourselves. Therefore, we first rely on existing solutions that can do the job for us.
- *If so, what does the selected project require to work ?* We mean, in terms of resources regarding the hardware as well as the man power for maintaining the related system.
- *Does the project provide a command-line tool ?* That is, does the project expose an interface such that we can trivially script it. This is more difficult if the project only exposes a GUI[8].

[8] Graphical User Interface

- *What are the input/output formats ?* We want projects that use formats that can easily be handled and preferably do not require too much pre- or post-processing.

If all of these criteria match our expectations (note that the last criteria about formats can be relatively easily overcome), we can start working with the selected project by just using it as a plugin in our system provided that we build an interface allowing our implementation to communicate with the plugged external system. This can be done by using a dedicated mechanism that provides this communication, explained hereafter, and subsidiary a wrapper (that is, actually the plugin) that can map generic arguments of our system to the input arguments of the related external system.

Orchestration Components In order to manage asynchronous tasking, it is necessary to use particular mechanisms so that our system can communicate with the external ones (e.g. CVE-Search). This implies using a dedicated protocol, a coordinating system called a message broker (designed according to an architectural pattern called publish-subscribe) and a producer/consumer implementation so that the communication is possible with the broker. Note that, while the choice of the protocol falls within standardization, the choices of the broker and producer/consumer already fall within the implementation of the system we want to build.

- **Protocol** : Advanced Message Queuing Protocol (AMQP), defined in [64], is *an open Internet protocol that defines a binary wire-level protocol for reliable exchange of business messages between two parties*. In essence, it relies on message queuing to make the components communicate.
- **Broker** : Multiple implementations exist for supporting AMQP but a very popular one is RabbitMQ. It is written in Erlang, open-source and multi-platform, making it a perfect candidate for our system.
- **Producer/Consumer** : Various implementations exist but a very popular one is Celery (another common one is Pika). This is a task queue which manages distributed message passing in an asynchronous way. It provides the meaning of tasks to the messages managed by AMQP and its broker. Celery provides what is called workers to implement consumers. For producers, these are just simple functions inside an application that publish messages to the broker through AMQP.

Figure 3.5 depicts the message queuing architecture. Note that a task information is formatted as a message before being sent through the architecture in order to asynchronously trigger a specific computation that gives back a usable result.

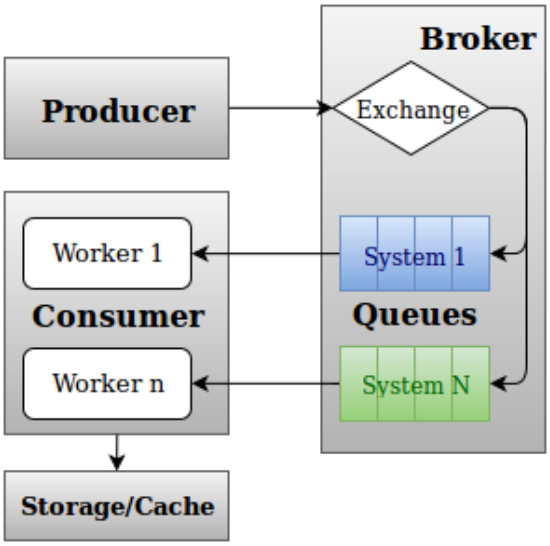


Figure 3.5: Message Queuing Architecture

Producer : This is a simple publish function inside a user application that sends specifically formatted messages to the broker.

Consumer : This is a component consisting of N workers in charge of consuming messages from appropriate queues.

Broker : This is the entity in charge of routing a given message from a producer to a consumer. The broker consists of an exchange and one or more queues.

Exchange : This is the entity in charge of applying the routing algorithm in order to forward the messages to the appropriate queues.

Queue : This is a buffer that stores messages, by default, in a FIFO manner, but which can be customized with priorities.

Note that the queues are labelled *System X* as it is logical to design these such that each queue routes the messages to a specific external system running potentially multiple workers. So, the reader should not be confused ; the *Consumer* as represented in Figure 3.5 could represent a cluster of systems.

Such an architecture makes it a scalable and modular system. We simply need to add a new worker in order to improve the throughput or simply plug a new sub-system by allocating a new queue in order to extend the system capabilities.

3.3.4 Virtualization

Virtualization is in fact the ability to run an operating system into another one, that is, the software implementation of a hardware, therefore avoiding to make a test operating system run on a dedicated hardware (which could be very expensive). That is why we choose to rely on it in the implementation of the system in support of our framework. However, it is to be noted that this point is still at the stage of concept as, for a question of means and experience in system administration, the system we describe in Chapter 4 does not rely on virtualization in its prototype version yet, even though it is already addressed in its design.

Appliance Deployment So, using virtualization greatly facilitates the deployment of a test environment but, in the scope of multiple security assessments on different software products, it is clear that not only one environment can be used to test everything, as some products could conflict with each other. Ideally, what we want is to test products independently such that we can avoid noises caused by other products. Note that, in practice, an appliance could be managed through snapshots (if the virtualization software supports it), after each assessment going back to the last snapshot in order to restore system's state, that is, without any trace of the previously installed products.

But, at the scale of ten's of software product evaluations at the same period, the time required to deploy one appliance for each product or even to make use of snapshots on a single appliance might run the risk of driving assessors insane. That is why a virtualized environments management should be considered.

Virtualized Environments Management Some appropriate tools exist in order to achieve a realistic and efficient management of virtual appliances. A few ingredients are required for this purpose :

- **Virtualization Platform/Software** : This is the system and/or software that supports the virtualized environments, that is, providing a virtual abstraction of physical resources (e.g. CPU, memory, interfaces) shareable across multiple appliances. One can mention, as a leader on the market, VMware and its products like Workstation or Fusion (virtualization software, respectively for Linux and Mac OS X) and ESXi Server (hypervisor) [65]. However, these solutions are commercial and may be very expansive. A good (but not always very stable) open-source alternative of virtualization software is VirtualBox [66].
- **Virtualization Management Tool** : This is a tool implementing a mechanism such that, by templating systems' hardware settings and operating system, new appliances can be quickly deployed based on these templates. An example of this is Vagrant [67].
- **Configuration Management Tool** : In conjunction with virtualization management, such a tool can push configurations (packages to be installed, user and application settings, ...) so that, once deployed, the new appliance can be customized based on its purpose. Several open-source solutions exist for this purpose, namely Ansible [68] and Puppet [69]. In our system, we will use Ansible.

Using such a chain of tools is really powerful and efficient for managing test environments. That is why we still already address using such a technology in our design (see Chapter 4).

SUMMARY

Software Approved Products List Automation Framework (SAPLAFv1) is a toolbox with its instructions of use, that is, a set of concepts, principles, techniques, methods and mechanisms around a central process aimed to build a Software APL. In other words, we provide all the required materials for an organization to build this famous list of approved products. Each item of the list is report about an assessed software product and is called an APL file.

This framework consists of the following items :

- **APL Process** : This allows to manage and control in a few steps the establishment of APL files and thus the APL.
- **Actors** : These are the *Manager* (managing the entire process), the *Technician* (for purely technical information about the product) and the *Assessor* (for the security assessment).
- **Status** : This defines the state of an APL file ; a file remains in *DRAFT* along the process and finally becomes *APPROVED* or *REJECTED* according to the decision of the *Assessor*.
- **Sub-processes** : There is one sub-process per actor from which the *Security Assessment* is the one of interest for this thesis.

The sub-processes of the *Manager* and the *Technician* mostly relate to other fields of Information Technologies such as, for example, *Software Asset Management (SAM)* or *Configuration Management* and are therefore not addressed in this thesis.

Inside the *Security Assessment* sub-process, one can find the following items :

- **Perimeters** : These are areas around the assessed product that section the scope into *Application*, *System* and *Network*, each with particular related controls and techniques.
- **Security Controls** : Mainly based on the ISM [22], we define categories and subcategories in order to determine the scope of possible controls. Note that defining specific controls remains a responsibility of the organization that wishes to implement the framework.
- **Methodology & Methods** : These notions are based on NIST SP800-115 [49]. The essential notions are the methods with the difference between *Testing*, *Examination* and *Interviewing*.
- **Testing Levels** : They define the depth of assessment and determine the required resources. They look a bit like the *Level of Trust* as defined in ISO27034 [18]. We define 5 levels from 0 (no assessment) to 4 (with a full test environment and even the source code available).
- **Techniques** : They are defined according to NIST SP800-115 [49] and consist of three classes : *Review Techniques*, *Target Identification and Analysis* and *Target Vulnerability Validation*.
- **Tools & Test Environments** : They address considerations about tools and test environment limitations related to the given techniques.

Moreover, some possible ways to provide its automation characteristic to the framework are provided, starting with the concept of **Modes of Operation**, together representing the notion of orchestration. Four modes are defined : *Manual*, *Interactive*, *Systematization* and *Automation*, defined according to the degree of interaction required with the user and the completeness of the output.

This notion influences how we can actually orchestrate tasks regarding the available sources, mechanisms and technologies that are proposed to build a system upon the framework :

- **Data sources** : This encompasses search engines, external systems and public repositories.
- **Asynchronous Tasking** : This provides a mechanism that allows to trigger tasks without waiting for them to finish, thus not blocking the overall working.
- **Virtualization** : This addresses the management of virtualized test environments for improving the efficiency of assessments.

Discussion

A lot of information is required to build a framework. As we could realize by parsing multiple standards, these always provide a myriad of concepts, principles, considerations, often guidelines and sometimes practical techniques but each time require an instantiation, that is, a long learning and thinking process to understand their substance and to be able to implement them in an organizational environment.

The little special thing we attempt to provide **with SAPLAFv1 is** to prepare it for **building a system upon** it. Even though we especially address the *Security Assessment* sub-process in the framework description as it is our point of interest for this thesis, the other sub-processes are still already handled in the system. These are described in a high-level way but are just placeholders to make the main process complete.

By defining the *Security Assessment* sub-process, we have defined **multiple criteria** allowing **to filter** a set of **controls** and related techniques, especially the *Perimeters* and *Testing Levels*, so that we can frame the necessary material for conducting a security assessment. We also **mapped** a sample of **controls with techniques** such that we can link the framework to its software implementation (as techniques are themselves linked to available tools).

In general, and as it can be seen in the standard we rely on (NIST SP 800-115), the described techniques require IT security practical experience. Moreover, the various tools related to these techniques unfortunately have the problem not to be supported on any environment and require, for most of them, a study for testing and scripting so that these can be plugged to our system. That is why we will explain in Chapter 4 only a few of them, but showing some quick-win and relevant ones. Furthermore, managing to **plug these tools to our system requires** a clean design such as **a plugin architecture**. We also show how our way to implement the system opens a broad scope of future projects.

All **these concerns address the What** (*do you remember the main questions of an audit ?* presented in Chapter 2) but **not the How**. That is where we propose the necessary **sources, mechanisms and technologies** to rely on for the implementation of the system upon the framework. Multiple data sources are possible, the implementable one with minimum effort being an interface to search engines, already addressed by the open-source project **GoogleScraper** [55]. Another quick-win, regarding the research of vulnerabilities and relying on an external system that can easily be deployed in our environment, is **CVE-Search** [58]. Among the possible technologies, we can also set up a Windows test environment with, for example, **ASA** [41] and MSBA [42] installed as internal tools, allowing to analyze the attack surface and setting changes when installing an assessed product.

Studying each possibly orchestrable tool requires a lot of time for testing and scripting. That is why we do not opt for performing such studies and thus concentrate on a generic way in which we could call scripted tools from our central system, that is, the plugin architecture. The few ones we select are for the proof-of-concept. Concerning the study of various tools, this will be addressed during an already foreseen internship for a student of the University of Mons (see future works in Subsection 6.4) during the first semester of the academical year 2016–2017.

We are now ready to describe the requirements, design and architecture of our system. The next chapter will try to bring to light these points.

4 System

"Intelligence is the ability to avoid doing work, yet getting the work done."

Linus Torvalds

Automation is the right way to avoid doing (too much) work by using a system that systematically runs repetitive tasks that could also eventually be error-prone and often time-consuming for humans. However, as told before, not everything is orchestrable and even when a part of the job can be automated or systematized, there still remains manual work. Furthermore, as the number of users grows (especially for different specialities whose representatives are not co-located), such as for APL involving managers, technicians and security assessors, a collaborative platform is required.

This chapter proposes the design of such a system applied to the scope of this thesis, whose interface is Web-based, starting by enumerating the requirements then proposing a design. We finally describe our implementation with its available features for the prototype.

Our Approach

In this chapter, we define our system following standard templates of software development, first describing the technical thread that as led us to its implementation. We try to provide links and pointers to previous chapters to make the reader figure out how we made our choices and then finally present what we were able to build in our first prototype. We emphasize the fact that this is only a proof-of-concept, thus surely still presenting some shortcomings in the sense of an experienced Software Developer but this will be refined as this project is only in its early stage.

The system, called **Security assessment for Computing APL (SCAPL)** (pronounced “scap-l"), is an open-source framework for automating tasks in the scope of security assessment of software products. The main goal is to make users spare as much time as possible in researching the required information for efficiently assessing software products. The main benefits of this system are the followings :

- An organization will be able to setup a customized structure for the APL files.
- A product assessment task will be achievable in a collaborative way.
- Research and automation features will help stakeholders in order to considerably reduce the time required to lead an assessment task to completion.

In order to keep the development process as light as possible on a management point of view, the Agile method was used. However, for the sake of standard compliance, we decided to use some IEEE Software Engineering standards. Sections 4.1 and 4.2 of this chapter respectively rely on IEEE830 [70] for Software Requirements Specifications (SRS) and IEEE1016 [71] for Software Design Descriptions (SDD).

4.1 Requirements Specifications

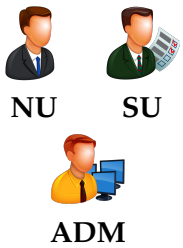
This section is a brief overview of the SRS made according to the structure of the IEEE830 [70] template. It presents a description of the expected system built upon the SAPLAFv1 framework with its actors, interfaces, desired features and non-functional requirements. The interested reader can consult Appendix F for a more complete extract of the SRS.

DO NOT BE CONFUSED !
In the sense of SAPLAFv1, actors are the *Manager*, the *Technician* and the *Assessor*, as the term “actor" is generic inside a process. We will refer to roles instead of actors in the remainder of this chapter to avoid any ambiguity. In the sense of SCAPL, actors are the *Normal Users*, *Security Users* and *Administrators* (explained hereafter), as the term is standard and commonly used in Software Development.

4.1.1 Overview

The chosen architectural pattern is a classical three-tiers, as explained hereafter, with the purpose of separating the main functions, that is, the presentation, processing and data management. This is considered as a best practice in development, especially for security reasons but also regarding the flexibility, scalability, maintainability and extensibility.

Actors



Normal Users (NU) and **Security Users** (SU) are the clients of the system, they execute the APL process. In the sense of SAPLAFv1, NU regroups the roles *Manager* and *Technician* and SU represents the *Assessor*. They have their respective features, including graphical wizards for gathering the information about a software product, a search capability for consulting former reports and other more advanced capabilities for the SU.

Administrators (ADM) technically manage the system and tune the APL process. They are not related to any actor of SAPLAFv1 but have an inescapable responsibility of system administration. They have dedicated features available for customizing the layout of the APL process according to what the *Manager* wants.

This interface relates to users' Web browsers, whose communication is achieved through the set of HTTP/HTTPS protocols.

Presentation

The **Front-End** (FE) is the component holding the user interface by running a web server. It handles actors requests and contacts a dispatch server in the application tier for triggering asynchronous tasks.

Hardware: Virtualization server
Software: Linux distribution with a web server

This interface encompasses the protocol dedicated to asynchronous task management, as explained in Subsection 3.3.3. A file transfer protocol is used to provide storage for the software installation packages or to collect documentation.

Application

The **Search Engine** (SE) is the component responsible for retrieving data about the analyzed product according to the mechanism presented in Subsection 3.3.2 about search engines.
The **Automation system** (AS) is the component responsible for setting a testbed and dispatching automated tasks to the right one, mainly relying on mechanisms and technologies presented in Subsection 3.3.4.

Hardware: Virtualization server
Software: Linux distribution with workers

This interface consists of a dedicated protocol for data transfer management according to the selected back-end database management system.

Persistent

The **Back-End** (BE) is the component that stores the long-term data in support of the application tier.

Hardware: Virtualization server
Software: Linux distribution with a NoSQL database

The APL process, as described in Subsection 3.1.1, can be instantiated in what we call a role-based sequence of inputs lists. Once the sequence is complete, a report is automatically generated and stored on the BE and then available via the FE.

EXAMPLE

An implementation of the APL process could look like, in the sense of SCAPL, a succession of three phases (each driven by a wizard), as illustrated in Figure 4.1.

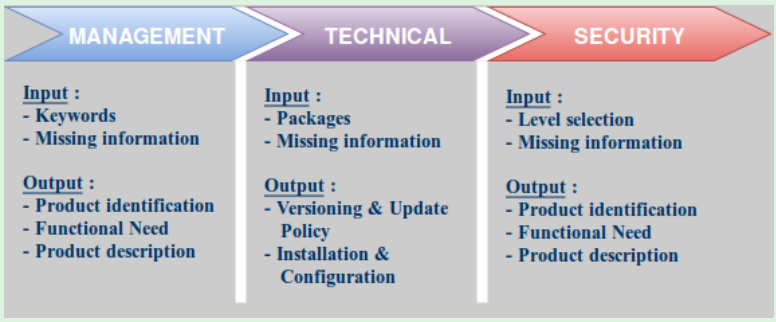


Figure 4.1: Sample APL process implementation in three phases

1. **Management** : For gathering information about the software product with SE.
2. **Technical** : For gathering installation and configuration information with SE.
3. **Security** : For testing and evaluating the product with AS.

With such an implementation of SAPLAFv1's main process, we separate the features according to roles :

- The *Manager* applies his/her sub-process and thus initiates the main process by providing some *Keywords* that identify the product and its version and fills in the essential information that should be known by any role regarding what the requirements the product has to fulfil.
- The *Technician* then builds the *Package* according to the installation and configuration mechanisms he/she uses in the production environment.
- The *Assessor* determines the *Testing Level*, lets the system perform its tasks accordingly, retrieves the assessment results and finally analyzes these for deciding on the approval.

This separation allows us to iteratively work on the implementation of each sub-process. Taking into account that, in SCAPL, we define two main actors (NU and SU), the sub-processes of the *Manager* and the *Technician* are, to a certain extend, merged in a single one gathering software asset, configuration and other non-security related matters.

4.1.2 System Features

The features offered to the users, that is, exposed by the **FE**, are the followings :


Sign up


Sign in/out


Profile edition


Search for reports


Submit info through a wizard

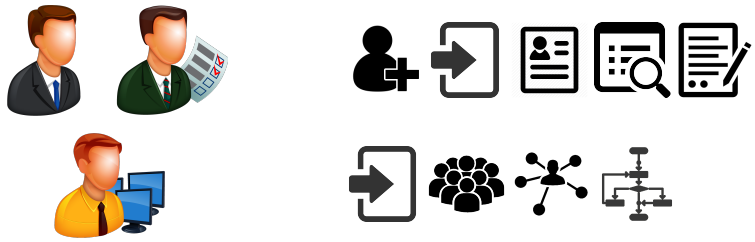

Users management


Roles management


Process management

These gather all the essential required functionalities for presenting the famous collaborative platforms to the stakeholders. Moreover, *Process management* must allow to customize the structure of APL files.

The features, per actor, are as follows :



Internal system's features are not presented here but can be consulted in Appendix F. The detailed requirements (with identifiers and dependencies) can also be found in this appendix. Note that the internal features take the principle of *Perimeter* (presented in Subsection 3.2.1) into account for the sake of separation of functionalities according to the different integrated tools (whose categorization is explained in Subsection 3.2.6).

4.1.3 Non-Functional Requirements

Non-functional requirements address the following characteristics :

Performance	• Regarding web page loading time which should be less than 5 seconds. • Tasks feedback must be given immediately at the page load. • FE must maintain an availability degree of at least 99% of the time (that corresponds to about 36' unavailability a week, i.e. for maintenance).
Ergonomics	• Attention is to be given on the user-friendliness by providing a simple and intuitive graphical interface. • Generic error messages must be provided with sufficient relevant information for the user (e.g. administrator contact email).
Security	• Access control must be managed according to the actors. Unexpected users should not have access to any feature, system or data. • Attention is to be given to secure coding and especially classical security flaws in web applications (e.g. CSRF[1], XSS[2])
Testing	• The final product should be tested for common security flaws. • Moreover, it should be tested, at least, on its extensibility and, if possible, also for scalability and portability.

As a convention, from this point, the following color code is applicable : the Presentation tier is represented in orange, the Application tier is in green and the Persistent tier is in blue.

[1] Cross-Site Request Forgery ; a common flaw in web applications that allows the attacker to execute an unauthorized request with the rights of an impersonated user.

[2] Cross-Site Scripting ; a common flaw in web applications that allows the attacker to inject content on a web page, causing unexpected behavior when users visit the page.

4.2 Design Principles

This section is a summarizing overview of the SDD made according to the structure of the IEEE1016 [71] template. It presents some essential information for understanding how the system is designed and how its architecture is built to fit to the SAPLAFv1 framework. The interested reader can consult Appendix G for an extract of the SDD.

4.2.1 Foundations

SCAPL's main idea relies on a flexible and customizable data scheme shaping what the end-users will see such that it implements the APL process of SAPLAFv1. The rest is just a bunch of systems aimed to support the tasks to be achieved behind the data scheme (that is, a mix of existing solutions as presented in Section 2.3, integrated according to the mechanisms presented in Section 3.3). So, the famous scheme simply consists of a few nested levels, each depending on its parent level with the top level sticking to a specific end-user type. These levels are, from down to top :

- **Data Item (DI)** : This is the lowest piece of data, holding its identification and content parameters but also specific parameters that can call an automated task in a component of SCAPL if it is automatizable. If so, some internal functionalities are triggered, e.g. in order to provide suggested values or offer an interface on a test bed to the user.
- **Data List (DL)** : This contains a series of related DI's, e.g. for the software product description or its installation information.
- **Data Sequence (DS)** : This regroups several DL's, for the aim of defining the structure of the wizard that the end-user allowed for this DS can see.

From these definitions, all **DS**'s stirred together then define the structure of the generated report. With such definitions, some DS can, of course, overlap between multiple end-user types ; in this case, the intersecting DL's are then **collaborative** and, in the other case, that is the DL's with no overlap, they are **exclusive**.

Such foundations allow to flexibly instantiate the APL process such that the information is shared among the actors. Especially, the *Assessor* can easily assess what the *Manager* and *Technician* fill in while they are doing so. This way of designing the foundations of our system is also an application of the well-known principle of **Separation of Duties** (SoD) in Information Technologies, stating that a single task should not be completed by only one person but well shared across multiple individuals. However, it should be noted that, by design, the *Assessor* has exclusive access to the assessment data and related features so that it reduces the possibly interfering actors to this potentially sensitive information. For this particular actor, in order to respect the SoD, it should be considered to assign an assessment team as the *Assessor* actor instead of a single individual.

The remainder of the design is made around these foundations using a classical software design methodology (as according to the used standard, that is, IEEE1016).

4.2.2 Overview

The system is described using the well-known 4+1 architectural views model (taught in [72]). This choice comes from the fact that this model allows to use multiple concurrent views that encompass all important aspects required to make a sound architecture. Using this model is commonly recognized as a best practice in the world of Software Development [73] [74].

Logical View As a first logical layer, we start from the output of SCAPL ; the *Report* object. This can be hierarchically defined as shown in Figure 4.2.

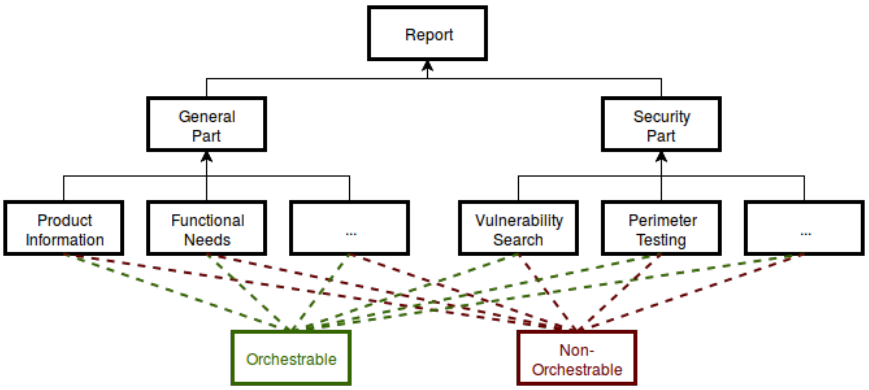


Figure 4.2: Object diagram for a report

On this diagram, the *Report* object is divided in two parts, each holding a series of sub-parts corresponding to the DL's. The lower level points out that not all DI's linked to the DL's are orchestrable according to the definition of the possible *Modes of Operations* (as explained in Subsection 3.3.1).

The link with subsection 4.2.1 is thus that, on the one hand, the layer beneath the *General* and *Security* parts represents the DL's and, on the other hand, DS's are combinations of these DL's.

As a second logical layer, we define the *Wizard* object. That is what materializes a DS in the user interface and then makes the link with the data scheme. It holds the series of DL's, each interpreted as a step in the completion of the wizard and listing the different items that are in relationship with SCAPL's internals.

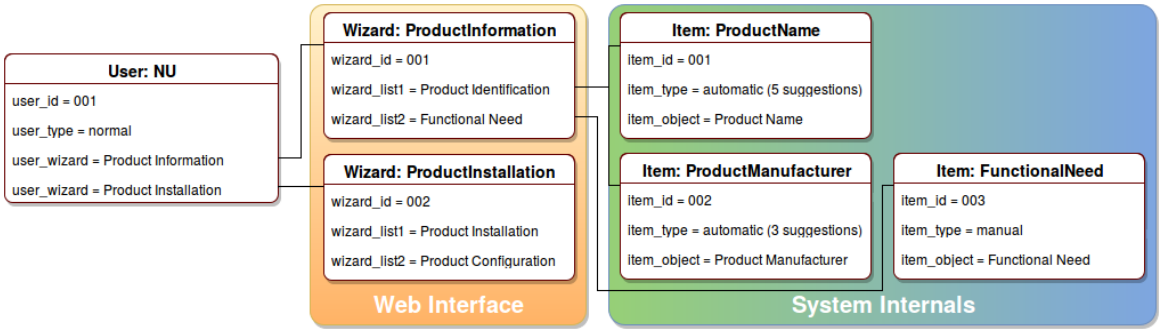


Figure 4.3: Object diagram for a wizard

Figure 4.3 depicts the relationships between a *User* object, some *Wizard* objects (corresponding to a DL, then consisting of several lists corresponding to DL's linked to DI's) and *Item* objects (holding DI's information and logics for the interactions with SCAPL's internals).

Process View The interactions we are interested in are these between the user and the system through the user interface. We only present one activity diagram for this view as the other ones are not absolutely required to understand SCAPL and are especially development details. The interested reader can consult the SDD for more information.

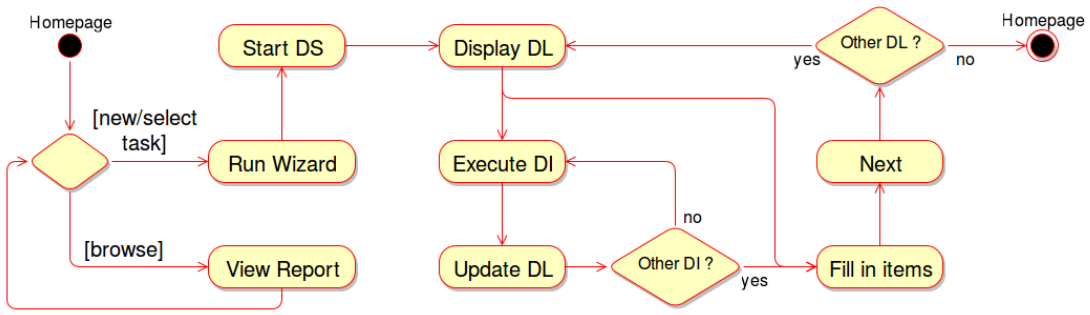


Figure 4.4: Activity diagram for browsing reports and starting a wizard

Figure 4.4 shows the activity diagram for some functionalities starting from the end-user's homepage. The most interesting part is of course this involving the wizard (*Run Wizard*). The user can choose to select or create a task, initializing a DS (*Start DS*) displaying several DL's as steps (*Display DL*), then triggering in turn DI's (*Execute DI*). As items' logics is aimed to assist the end-user, he/she can use the collected information to fill in items' content (*Fill in items*).

Development View For the sake of conciseness, this view is restricted to the package diagram as it holds more than the information presented by the context diagram (available in the SDD in appendix). This diagram is presented in Figure 4.5.

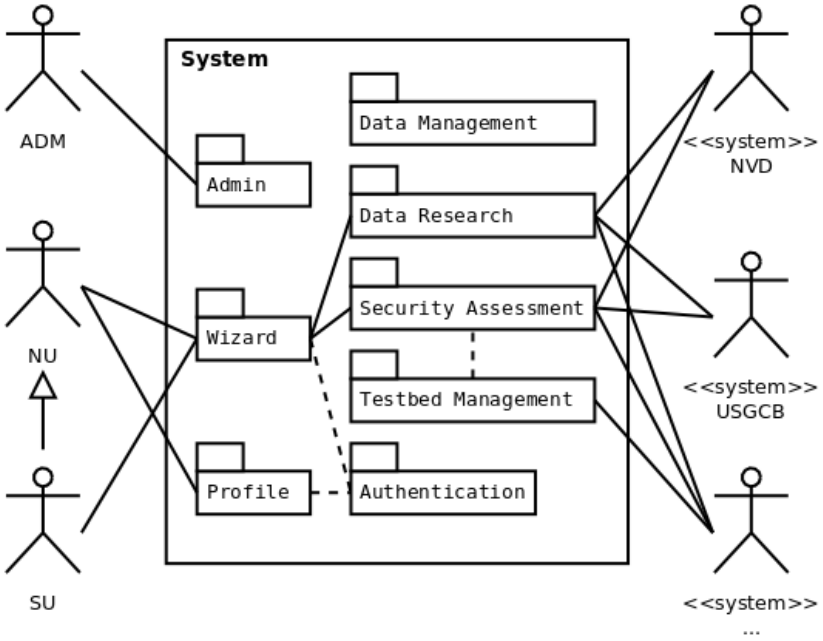


Figure 4.5: Package diagram of SCAPL

The system is decomposed in categories of functionalities with primary packages (on the left in Figure 4.5), contained in FE, including :

- *Admin* : it regroups functionalities for the ADM, especially for managing the whole APL process (through the data scheme) and the users
- *Wizard* : it holds the features that manage an instance of the process (that is, an opened APL for a given product and its related triggered tasks)
- *Profile* : it is a more or less standard set of features for end-users

The system also contains secondary packages (on the right in Figure 4.5), spread among the components other than FE, including :

- *Data Management* : it contains the report generation feature
- *Data Research* : it holds the search features to be managed by SE
- *Security Assessment* : it contains the security perimeter testing features to be managed by AS
- *Testbed Management* : it contains the virtualization features (as described in Subsection 3.3.4) to be managed at the level of AS by the ADM (in support of *Security Assessment*)

Physical View As stated in the requirements in Section 4.1, the system follows the three-tiers architectural pattern. This view thus states how we intend to build the system on a practical point of view, providing choices of supporting platform for each component. SCAPL's internals design is depicted in the deployment diagram in Figure 4.6.

In this figure, one can note that FE is supported by a web server made with Django [75], a very convenient framework for building powerful and user-friendly web interfaces, using Apache as the web server and a MySQL database server for supporting data (namely related to the data scheme and user profiles). An important point is that FE, SE and AS are communicating through AMQP (explained in Subsection 3.3.3). This requires the addition of one more underlying component in the Application tier, the **Backbone** (BB),

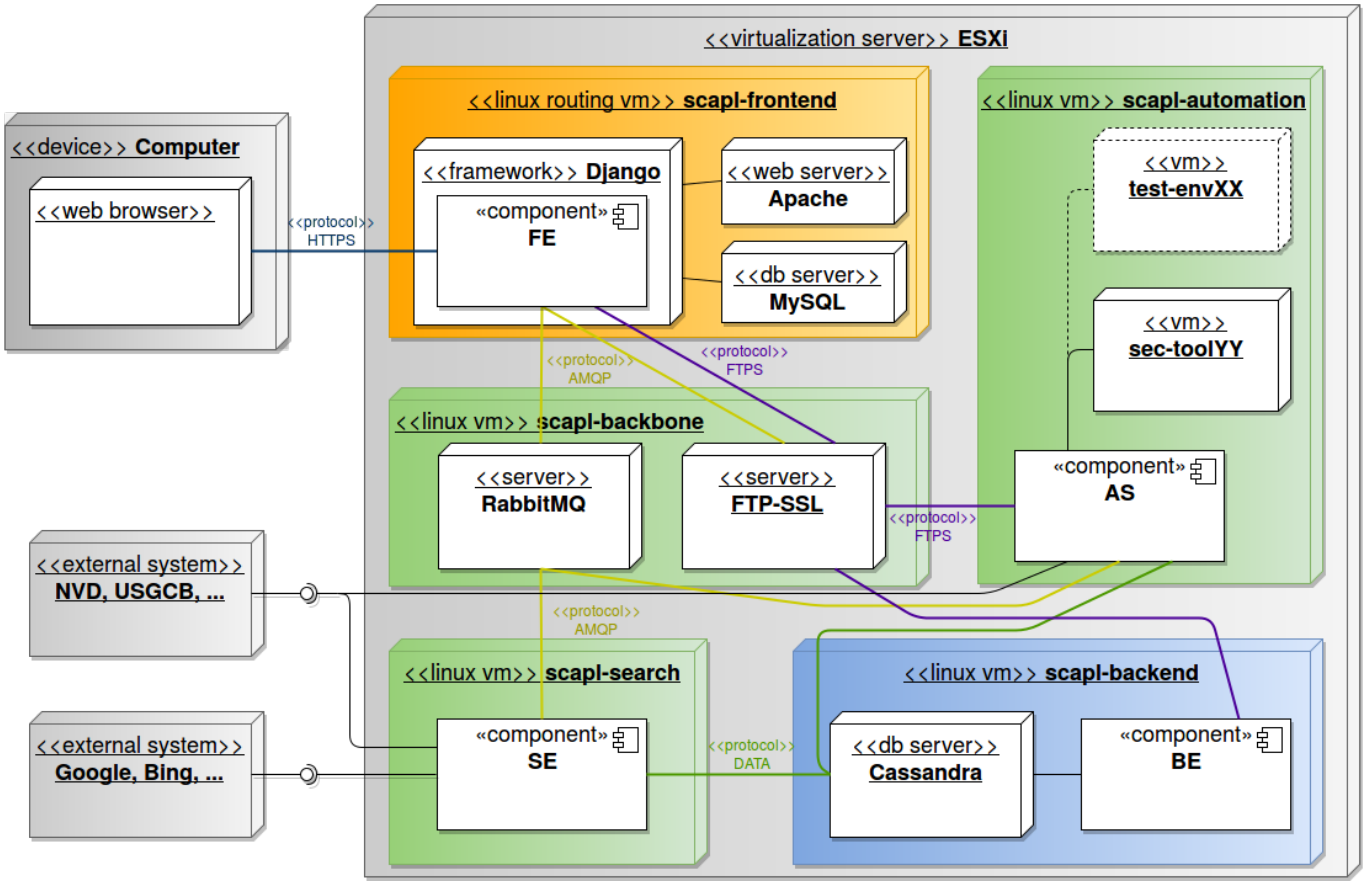


Figure 4.6: Deployment diagram of SCAPL

running a RabbitMQ [76] server for coordinating the asynchronous tasks through AMQP (that is, the *Broker* according to the explanation in Subsection 3.3.3) and an FTP server for storing various files in support of FE. SE and AS communicate with BE through a dedicated protocol function of the chosen NoSQL database management system (either Cassandra or MongoDB).

Some external sources can be contacted through interfaces, namely search engines for SE but also common sources in application security such that the database of vulnerabilities from NIST NVD [31] or the database of security configuration baselines from NIST USGCB [53] (see Subsection 3.3.2). The search engines are aimed to help searching for the information related to a software product to be assessed and the other external sources can, for example, be used to check the baseline that test environments rely on.

As a first experience in mechanized deployment of virtualized environments, we decided to spend a bit of time working with Vagrant and Ansible (presented in Subsection 3.3.4) so that SCAPL's development environment can be automatically deployed. The interested reader can check on GitHub our *scapl-install* project [77] for more information. For the time being, this has been tested on VMware Workstation. Note that the plugins of Vagrant for working with VMware are charged.

Use Cases These are not presented here as these are more development details, not really suitable and too long to be completely described in this dissertation. They can be consulted in the SRS in Appendix G. Note that the use cases are willingly not described in the common way for the sake of conciseness. Indeed, they should be described according to a template, namely stating the post- and pre-conditions, nominal, alternative and exception scenarios and end states.

4.2.3 Architecture

Given what we describe in the previous subsection with the 4+1 architectural views model, SCAPL's architecture can now be more simply and conveniently depicted as in Figure 4.7.

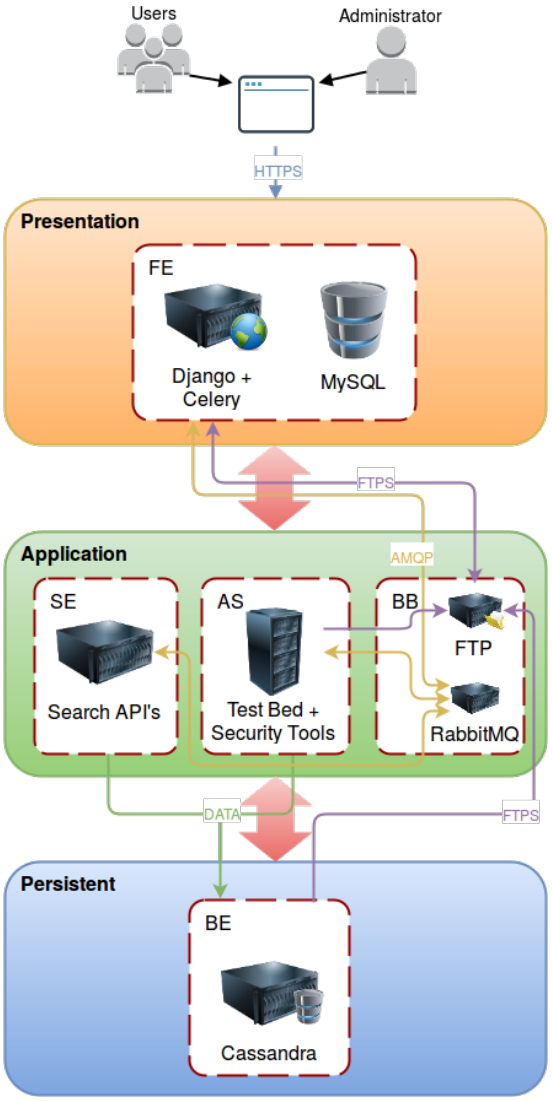


Figure 4.7: Three-Tiers Architecture of SCAPL

This figure depicts a summarizing view of SCAPL. Indeed, one can see the three-tiers architecture and the communication protocols between the components. The pictures point out instantiated sub-systems for supporting the components. Detailed responsibilities per component are provided in the SDD in Appendix G.

SCAPL's architecture aims to be designed with a few important principles in mind :

- **Security** : From a security point of view, a three-tiers architecture is a best practice (easier to harden, more difficult to cause data leakage). Moreover, secure protocols (i.e. HTTPS and FTPS) are used between each component.
- **Portability** : Each sub-system is made to run on a VM such that it can be deployed on any operating system supporting a virtualization software or hypervisor.
- **Scalability** : It is possible to distribute the components across multiple separated hardwares. Especially, according to the figure aside, any component can scale horizontally.
- **Extensibility** : One the one hand, components can be added in the future by respecting the chain FE-[Component]-BE. On the other hand, each component can be structured to receive plugins containing interfaces able to receive asynchronous tasks from FE through BB.

4.2.4 Data Scheme

The data scheme managed through FE is decomposed in two main categories :

1. (Nearly-)**Static** : corresponds to the data scheme supporting the dynamic wizard. This is qualified as dynamic for emphasizing the fact that, through this particular data scheme, the final report structure can be customized and tailored to organization's needs. It is translated at the data level into a scheme that, normally, should not change very often, hence the category (nearly-)static.
2. **Volatile** : corresponds to the data scheme for supporting user and tasks management. Indeed, user profiles have to be stored somewhere and available for FE. Moreover, once a wizard is started with keywords matching a new software product, a new task (with its metadata) has to be recorded in order to keep track of it. Therefore, in contrast to the previous category, the data is volatile.

The EER[3] model is provided in Figure 4.8.

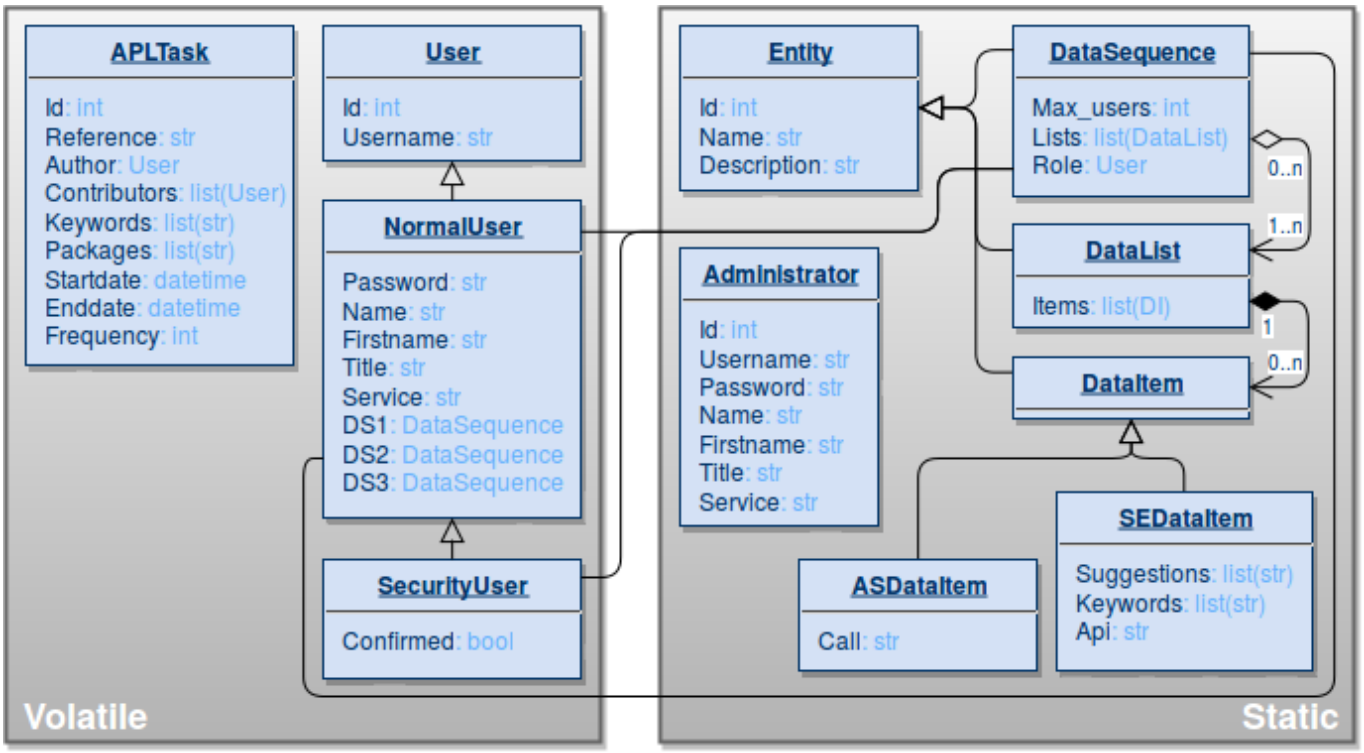


Figure 4.8: EER model for the volatile and static data schemes

From this model, we point out the following elements :

- **APLTask** : is the structure containing the information of a software product assessment and is thus an instance that uses the defined static data scheme.
- **User**, **NormalUser**, **SecurityUser** : are trivial entities corresponding to NU and SU. Note that *SecurityUser* inherits *NormalUser* as it holds the same information with a *confirmed* attribute, meaning that as it has more rights in the whole system, it must be flagged by an *Administrator* in order to instantiate a valid account.
- **Administrator** : is also a trivial entity corresponding to ADM. Note that this entity is part of the static scheme as administrators should not often change and do not interact with the APL process.
- **DataSequence**, **DataList**, **DataItem** : wer already explained.
- **ASDataItem**, **SEDataItem** : are particular types DI, containing different attributes according to their purpose characterized by a responsible component.

4.2.5 Components

Each component of SCAPL is built according to a template, as presented hereafter in Figure 4.9.

FE is built using the Django framework and then respects a particular structure inspired from the documentation of an experimented web developer [78]. It namely contains a folder **apps** with each package as defined in the package diagram in Figure 4.5. The other components, SE and AS, follow a very simple structure with only one type of folder, the plugins. These folders contain external project adapted to provide an API for the communication between FE and the related component through the RabbitMQ server.

[3] Enhanced Entity-Relationship

Figure 4.9: Generic component internal structure

- **[component]** : is the folder containing the source code of the component.
- **config** : contains the settings of the supporting system.
- **docs** : holds the relevant documentation for the given component.
- **tests** : is folder with unit tests for assessing component's source code.
- **...** : designates other possible folders, specific to the component (e.g. **static** and **media** files for FE).
- **license** : holds the copyright and, if relevant, an open-source license such as GPL[4].
- **readme** : is the common basic help document for any project.
- **requirements** : is the list of required packages of the component for Python.
- **...** : designates other possible files, specific to the component (e.g. **tasks.py** and **worker.py** files for SE and AS for the communication with the RabbitMQ server).

4.3 Implementation

This section presents SCAPL in its current state regarding what was defined in the SRS and the SDD. It first gives an overview of the prototype and then explains the satisfied requirements and available features regarding Subsections 4.1.2 and 4.1.3.

4.3.1 Prototype

The prototype is available on GitHub [77] and is separated into multiple projects. As already pointed out in Subsection 4.2.2, one more project exists for the automated installation of each component using Vagrant [67] for the automated creation of VM's (referred to as the *Virtualization Management Tool* in Subsection 3.3.4) and using Ansible [68] playbooks for the provisioning of these (referred to as the *Configuration Management Tool* in the same subsection).

Current state of SCAPL Dissected by component :

- **FE** : As it is the central component, a particular attention was given to its development. The Web interface is ready and offers the desired user-friendly interface thanks to the use of Bootstrap[5] [79]. The asynchronous tasking with Celery [80] is in place and effective.
- **SE** : This component currently contains its main plugins, GoogleScraper [55] (which uses multiple search engines for providing information suggestions to the end-user) and CVE-Search [58] (for getting the known vulnerabilities for a given product).
- **AS** : This component currently contains its main plugins, GateOne [81] (which allows the end-user to connect to a test bed VM) and Attack Surface Analyzer [41] (which takes snapshots of a Windows machine in terms of attack surface and is able to generate reports making the difference between these snapshots).
- **BB** : This extra component currently runs a RabbitMQ server and correctly coordinates asynchronous tasks between the other components.
- **BE** : This component is not implemented yet as it only has an interest in the production version for storing collected data when a lot of APL tasks will be managed.

[4] GNU Public License, a typical license for free software
[5] A well-known HTML/CSS/Javascript framework for Web development

Plugin architecture As already mentioned in Subsection 3.2.6 and in the discussion of Chapter 3, related to the mechanism explained in Subsection 3.3.3, an important concept to be emphasized is the *plugin architecture* of SCAPL, namely crucial for its extensibility. For this purpose, a specific element called *TaskRunner* was developed in order to provide the link between the Celery worker (that is, the *Consumer*, operating in SE or AS) that receives the asynchronous tasks from the RabbitMQ server (in other words, the *Broker*, running on BB) and the project that is plugged in SCAPL (that is, the *Producer*). Working this way allows to wrap the entry point of the plugged project without modifying it such that its output can be formatted as expected by FE for displaying the result to the user. As this is more technically detailed, no example of plugin is presented in this subsection but the interested reader can consult Appendix H for the one we implemented for using the project CVE-Search.

Wizard It can be decomposed in two parts : the *starting step* and the *process execution steps* (sticking to the defined data scheme). Currently, the *process execution steps* are ready-to-use with practical dynamic HTML elements supporting the end-user navigation but the *starting step* only handles pre-defined entries for a sample of software products. This last point remains problematic as, even using CPE[6] (which are not defined for all software products as this input comes from the community and is in general only defined for very common products), the problem of getting the right information for starting an assessment (i.e. the right product name, vendor and version) addresses data mining (as addressed in Subsection 3.3.2), which is not in the scope of this master thesis. But this should lead to an ancillary project in a near future (more information in Section 6.4).

The plugin architecture is the most important feature regarding the flexibility of SCAPL for implementing the automation characteristic of SAPLAFv1. It provides an easy to learn and use way to plug more and more external projects. In the same way, the data scheme provides an easy to learn and use way to tune the wizard to organization's needs according to the controls it requires for the security assessment of a software product.

4.3.2 Satisfied Requirements

In more details, especially for FE, the available required features (as defined in Subsection 4.1.2) are listed hereafter.

Functional Requirements

- **[NU | SU]** *Sign up/in/out* are trivial features in any web development projects.
- **[NU | SU]** *Profile* edition is relatively simple and, in order to gain in user-friendliness, it is possible to customize the interface theme thanks to the integration of Bootstrap themes [82].

[6] Common Platform Enumeration

- **[NU | SU]** *Search* feature is made to deal with any recorded data, either in APL tasks or stored reports (so, more powerful than required).
- **[NU | SU]** *Information submission through a wizard* is managed by using very practical HTML elements in order to improve the end-user experience. The suggestions collected through the asynchronous tasks are displayed inside the wizard using sandboxed HTML elements (these capabilities are available from HTML5 [83]).
- **[ADM]** *Sign in* is a trivial feature. A default account *admin* with password *admin* is created at the installation of SCAPL (this password must obviously immediately be adapted after it !).
- **[ADM]** *Users management* is made easy through the administration interface. Note that new administrators must be added through the administration interface by another administrator.
- **[ADM]** *Process management* is also made simple and convenient through the administration interface.
- **[ADM]** *Roles management* still lacks configuration capabilities (hence, in orange in the pictures before) which mainly rely on a complex problem of rights and permissions on every object in the Django framework. This problem will be addressed in the future.

Non-Functional Requirements

- **Performance** : This is managed by the Django framework and the installed additions. In order to improve the performance, some optimizations are used (mostly inspired from [84]), namely caching (foreseen for the production version) and CSS/Javascript compression (for reducing the amount of data to be transferred between the browser and the web server). The asynchronous tasking performance depends on Celery but an immediate response is sent to FE at each task triggering in order to provide immediate feedback to the end-user without slowing down the web page load.
- **Ergonomics** : FE is implemented using a lot of Django packages [85] that greatly improve the user-friendliness of the interface. One can namely point out (**A**: Admin interface, **U**: User interface) :
  - **[A]** *django_admin_bootstrapped* for modifying the default interface to make it Bootstrap-fashioned
  - **[A]** *admin_reorder* for reorganizing categories of settings in a convenient way
  - **[A]** *adminsortable2* for providing dynamically sortable lists, useful for defining the data scheme
  - **[U]** *bootstrap3* for adapting the user interface to make it Bootstrap-fashioned
  - **[U]** *bootstrap_themes* for allowing the end-user to custom his/her interface
  - **[U]** *django_summernote* (the Django package allowing the easy integration of Summernote [86], an excellent WYSIWYG[7]editor) for providing an HTML editor for each DI in the wizard
- **Security** : This is mostly inherently ensured by Django built-in features [87] such as, for example, a middleware for preventing CSRF attacks. SCAPL also uses a Django package named *admin_honeypot* that fakes an administration interface for recording attempts to connect from malicious users.
- **Testing** : The extensibility was tested by using the plugin architecture of SCAPL. By lack of time and hardware means, the scalability and the portability were difficult to test. SCAPL was tested by installing its components on a VM in VMWare Workstation and will be tested on an ESXi server in a near future. For the security testing, an audit will be executed on the production version.

[7] What You See Is What You Get

Design and Implementation Constraints

- **Extensibility** : This is satisfied through the customizable data scheme and the related plugins in the underlying components.
- **Portability** : This is inherently satisfied through the design of SCAPL.
- **Scalability** : This is inherently satisfied through the use of scalable frameworks and systems (i.e. Django, Cassandra or MongoDB).
- **Modularity** : This is satisfied by design through the standard Django project structure and the plugin architecture.

4.3.3 User Interface

The result of our development is the web interface depicted in the following screenshots. We present the five most important screens of the application, make the relationship with the requirements and explain the functionalities.

Index

Figure 4.10 shows the index page any user sees when browsing on the web server of SCAPL. It exposes several links whose buttons providing the *Sign In/Out* (1) and *Sign Up* (2) functionalities.

Figure 4.10: SCAPL Screenshot – Index

APL Overview

Figure 4.11 shows the page an end-user (NU or SU) sees when he/she selects an APL task. This page displays the information filled in by him/her or a contributor (1). One can already point out the search feature (2) integrated in the toolbar.

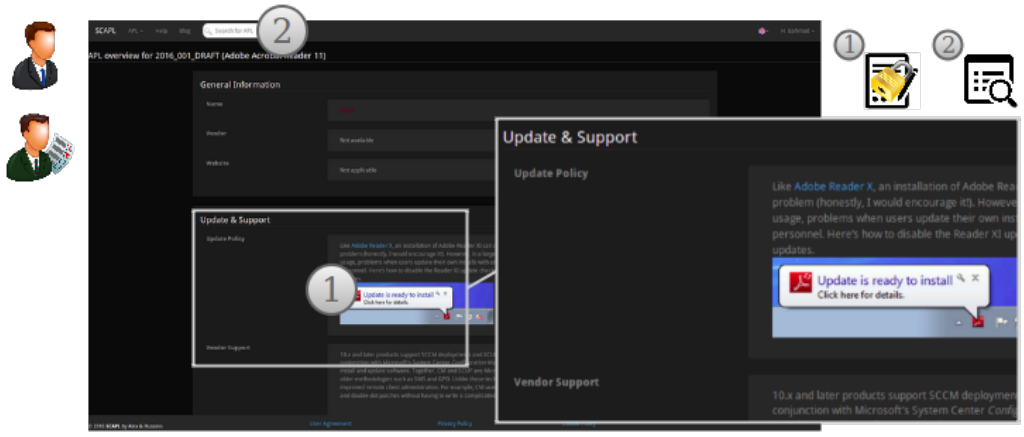


Figure 4.11: SCAPL Screenshot – APL overview

At this point, if he/she is the author or a contributor of the APL task, the user can click on any information on the page to open the modal holding the wizard (1), like shown in Figure 4.12. Otherwise, the modal will not open as he/she is not allowed to open it (hence, the lock on the *Submit Information* functionality icon).

APL Wizard

Figure 4.12 shows the overview page with the modal opened for the wizard (only accessible if the user is the author or a contributor). This modal contains, for each item, an HTML editor (1) and a panel (2) showing a research (that is, suggestions got from the search engine) or automation result.

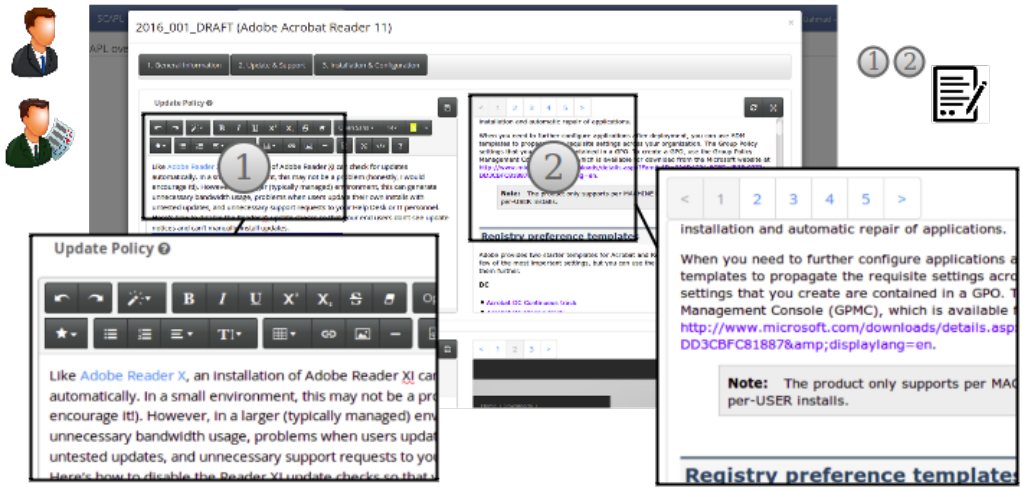


Figure 4.12: SCAPL Screenshot – Wizard

User Profile

Figure 4.13 shows the user profile edition page (1). Note that, contrary to the the other screenshots (except this of the wizard), this example is taken with a different theme to illustrate that user-friendliness is customizable. Again, one can see the search feature (2) in the toolbar.

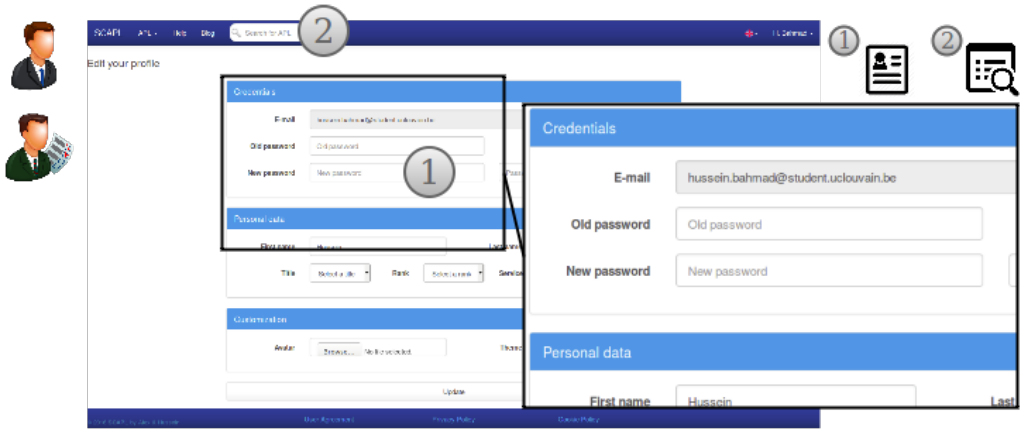


Figure 4.13: SCAPL Screenshot – User Profile

Administration

Figure 4.14 shows the administration board that is only accessible by the administrators and aims to provide a single board for all settings. The data scheme is then easily customizable (1) as well as the user and roles management (2).

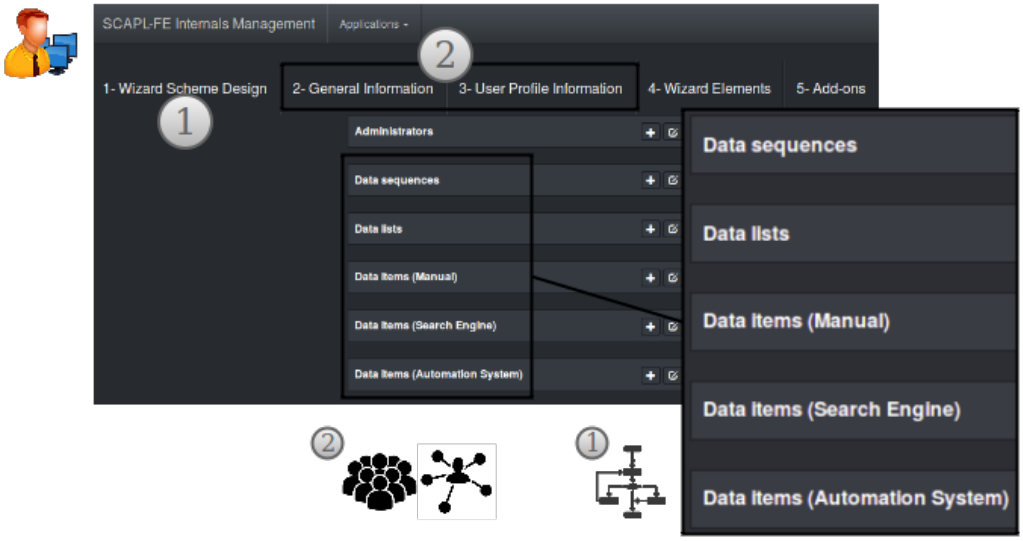


Figure 4.14: SCAPL Screenshot – Administration board

SUMMARY

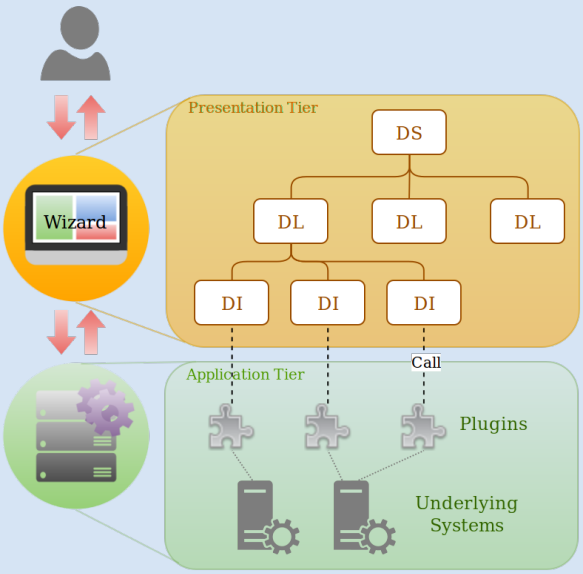
SCAPL is designed to offer to its end-users a **user-friendly web interface for managing an instantiation of the APL process** especially for software products. Through this interface, some particular **users can execute this process** whereas **others can tune it** according to the needs of their organization. The different types of users are :

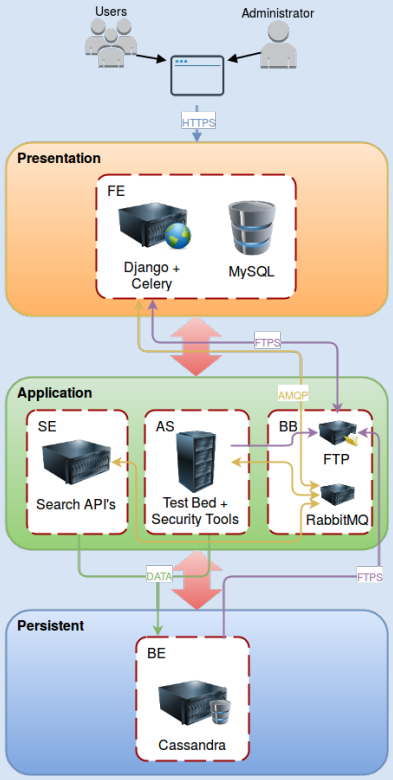
- **Normal User (NU)** : This type of user sticks to managers or technicians who fill in general information about software products.
- **Security User (SU)** : This type of user performs more advanced tasks related to the security assessment and are then allowed to fill in more specific information about software products.
- **Administrator (ADM)** : This type of user is responsible for the management of SCAPL.

SCAPL is built around a **particular data scheme** closely related to the APL process. It is tuned by ADM and consists of the following elements :

- **Data Item (DI)** : This holds parameters for calling an automated task in a component of SCAPL (if automatizable) such as providing suggested values or offering an interface on a test bed.
- **Data List (DL)** : This is a set of related DI's, e.g. for the software product description.
- **Data Sequence (DS)** : This regroups DL's, for defining the APL process for a specific user.

From the figure on the right, one can see that each DI can call a plugin, providing SCAPL with extensibility through this **plugin architecture**.





SCAPL is designed according to the **three-tiers architectural pattern** and contains the following components :

- **Front-End component (FE)** : This is the web server based on Django exposing the user interface and triggering asynchronous tasks to the components of the Application tier with Celery [80].
- **Search Engine (SE)** : This is responsible for managing research tasks for providing lists of suggestions to FE.
- **Automation System (AS)** : This is aimed to automate some security and configuration tasks and to offer interfaces on test beds for the security assessment.
- **Back-End (BE)** : This NoSQL database records the data collected during the execution of the APL process.
- **BackBone (BB)** : This essentially coordinates the communication of asynchronous tasks between FE and SE/AS thanks to a RabbitMQ server.

SCAPL's design holds some common characteristics such as the extensibility and the scalability but also emphasizes the security.

Discussion

In essence, SCAPL aims to solve the tedious problem of setting to music a framework. But its design is built upon our public framework, SAPLAFv1, which provides its toolbox of required concepts, principles and mechanisms. By including asynchronous tasking, virtualization and bindings to various data sources, **SCAPL** becomes an **open-source** system that provides a beautiful and user-friendly Web interface that **can gather a great quantity of relevant information for** supporting software **security assessments**.

With its **tunable data scheme** and **straightforward plugin architecture**, it provides to an organization the flexibility for adapting the APL process from SAPLAFv1 to its functional needs. Moreover, with this **flexibility and extensibility**, the overall system can continuously be improved, covering more and more security controls.

SCAPL is **still in** its early stage, in the state of a **prototype**, but it is **now ready to live its heyday**. And it is going to start very soon, with the **openings** it offers thanks to its extensible and flexible design and the broad scope of knowledge it could still cover. However, given the mechanisms and technologies that SAPLAFv1 proposes, there still remains some major improvements to be developed such that SCAPL can become a solution deployable into organization environments. For now, we will try to promote the project so that we can establish collaborations as much as possible with some academical or professional organizations.

According to the requirements in Section 4.1 and the design in Section 4.2, SCAPL is an ambitious project and does not fit to a team of two developers in the limited time available for this master thesis. That is why we focused on some essential features for building a proof-of-concept. However, this project opens various opportunities to plan collaborations with other research entities (such as, for example, internships for students from universities into the enterprise developing SCAPL). See Section 6.4 for an overview of our ideas for continuing the project.

Some more remarks about SCAPL :

- The **data scheme** defines a clean and simple structure, only based on three different types of entities (the data sequence, list and item), that confers it a certain **malleability**. Easy to manipulate and directly related to SCAPL's **plugin architecture**, it allows **quick changes of structure and reporting** adaptation. By providing new plugins according to the related components (each assuming a different mode of operation), the sophistication of the overall system is then inherently increased. To be convinced of this, the interested reader can consult Appendix H to see how we implemented our plugin CVE-Search in SE.
- The asynchronous tasks operate in several different manner, called the **modes of operation**. **Each component** of SCAPL **uses a particular mode** (except *Manual*, which is trivial). In order to implement a new plugin, it must first be chosen which mode it will use so that the plugin template of the right component can be selected and tuned. It is to be noted that, in the previous chapter, we also mention a particular mode, *Interactive*, which is not formally implemented yet, while there exists an interface that now assumes this functionality, but in a non-optimal way, using GateOne in AS. We plan to add a dedicated component to address this lack and to mechanize interactive accesses to external test platforms.
- A not insignificant feature that SCAPL offers thanks to the usage of Bootstrap in FE is that its **Web interface** is therefore **responsive** and can thus also be used in a convenient way on tablets.

5 Experiments

"If you spend more on coffee than on IT security, you will be hacked. What's more, you deserve to be hacked."

Richard Clarke

Investment in software security is today an essential point in modern companies, especially regarding the hackings revealed in the news these last years (see namely WikiLeaks). With our implementation, we aim to propose a new mean for investing human resources in the software security process through SCAPL, our system implemented upon the SAPLAFv1 framework.

This chapter provides an example of methodology and an instantiation of the data scheme for trying experiments on some common software products. The data items of this scheme are chosen according to some basic criteria for the sake of simplicity for our proof-of-concept.

Our Approach

In this chapter, we now use SCAPL and instantiates its wizard structure so that we can conduct our security assessment on some common software products. We expect to exploit our solution in such a way that we can show to the reader that it is really convenient and straightforward to use and that it can make spare a lot of time. It should be noted that the focus is not on the complexity of the security assessment but well on the quality of SCAPL.

The experiments, that we limit to applications (no operating system, driver or related product is tested), are based on some very commonly used ones that are free of charge, i.e. Adobe Reader, CCleaner and Skype. We thus perform a full sequence of actions, that is, the methodology determination, the data scheme definition and finally the experiments. We use them so that we can emphasize the following benefits of SCAPL :

- The data scheme is very easy to configure given assessment's required controls.
- A product assessment can be led in a collaborative way.
- The provided features greatly help to spare time in researches.

5.1 Methodology

This section presents our methodology for performing our experiments on SCAPL. This includes defining a context with its actors as if it was a (more or less) real situation, the steps that lead them to the assessments and a scenario with a detailed description that involves the actors. Note that this methodology and the experiments in the next section more relate to illustrated user stories (in the sense of Software Development) with metrics provided to evaluate the benefits of SCAPL.

5.1.1 Context

Cutty Panties Inc. (fictive, we precise...), a manufacture of underwear, wants to invest in information security as it suffered, a few months ago, from a hacking revealing its trade secrets about its last model of cute pants made out of silk and worn by some eminent movie stars. Its top manager, Sam, then decides to ask to his IT director, **Mike (the *Manager*)**, to implement a process that allows to make an APL of the software products running on the systems of the company, knowing that the budgets for this purpose will be thin considering the sales revenue loss that the recent cyber-attack has already caused.

Mike, who can rely upon its IT technical staff, especially **Tom (the *Technician*)**, and its IT security collaborator, **Alan (the *Assessor*)**, decides to start working with SAPLAFv1, an emerging framework that seems to address all his requirements and he thus tasks Alan to work this out.

Alan then applies to studying SAPLAFv1 and realizes that there already exists a practical ready-to-use solution, SCAPL, for implementing the APL process. After a bit of reading and with the help of Tom, they install this system at their organization and discover how to use it, especially how it can instantiate the process.

Mike, delighted by this solution, organizes a kick-off meeting with Tom and Alan and requests them to do the test of performing some security assessments on **CCleaner, Skype and Acrobat Reader** as a starting point, then applying the APL process discovered in SAPLAFv1. He also designates Alan as the administrator of the newly installed system.

5.1.2 Steps

Alan, in his findings, discovers that a very interesting dissertation was written about SAPLAFv1 and SCAPL and deduces from it that the methodology can simply be summarized in only a few steps, which are (with their references in the found document) :

1. **Determine what we want in our assessment** (relying on SAPLAFv1 in Chapter 3) : This is done by Mike, assisted by Alan (for the security matters), in order to instantiate the APL process. It will provide the structure of the assessment information that he wants in the reports.
2. **Define DS/DL/DI** (relying on SCAPL in Chapter 4) : This is done by Alan (who was designated by Mike as the administrator of SCAPL). Altogether, the defined data entities will implement the famous structure and format the wizard for all the stakeholders.
3. **Apply the APL process** (relying on the definition in Chapter 3) : That is, Mike will start the tasks for each software product and let SCAPL do its job to expose its collected information and suggestions. Collaboratively, Mike, Tom and Alan will then fill in their respective items so that Alan can finally assess the products and decide on their approval based on the generated reports gathering the whole information collected during the process.

5.1.3 Scenario

Assumptions

- SCAPL was set up by Tom and Alan ; we do not take the time of installing it into account. Anyway, a *scapl-install* project is included on GitHub and the system is considered straightforward to deploy.
- Mike, Tom and Alan have already signed up and Alan has assigned them the right roles in SCAPL.
- Everyone has configured a different Web interface theme in his profile (that thus distinguishes the different screenshots in the remainder of this chapter).

Step 1 Mike decides to start with a light structure, taking into account some simple information about the software products, that he defines as follows :

Product Identification
- Name
- Vendor
- Website
- Platform
- License
- CPE
- Version

Product Description
- Description
- System Requirements
- Language
- Existing Literature

Security Data
- Integrity Validation
- Known Vulnerabilities
- Comment

Moreover, Mike determines that *Product Identification* and *Product Description* should be editable by either him or Tom.

By looking at SAPLAFv1 in the dissertation he has found, especially the *Security Assessment* sub-process, Alan also determines that he can limit his scope to the *System* perimeter as the selected products do not require more. Furthermore, as he has noticed in the foreword of the dissertation, the mantra of an IT auditing teacher, James Tarala, "*start small, then grow your scope*", he decides to limit his assessments to the *Testing Level 1 – Basic Search*. Anyway and unfortunately, SCAPL does not seem to currently support deeper analysis regarding the assessment techniques but, for a prototype, it could prove its value and generate some investments in contributions to the project. So, no test environment this time, but it should be addressed in a near future.

Step 2 This is detailed in Section 5.2 addressing the data scheme and achieved by Alan.

Step 3 This is addressed in Section 5.3 and is achieved by everyone according to his role.

5.1.4 SCAPL Evaluation

In order to evaluate to what extent SCAPL keeps its promises regarding its benefits (presented at the beginning of this chapter), we will provide our observations while using the system for each experiment, especially :

- The fact that the data scheme is very easy to configure is emphasized in the next section.
- The collaborative way in which the assessments can be led is implicitly shown through a good design of the data scheme and the role of each actor will be clearly mentioned in each experiment.
- The time in researches spared will be illustrated through screenshots of SCAPL and explained through rough estimations of time spent to perform the researches manually and letting the system do its job.

5.2 Data Scheme

First, Alan (who has two types of powers ; ADM for administrating SCAPL and SU for using the security assessment wizard) connects on the administration interface.

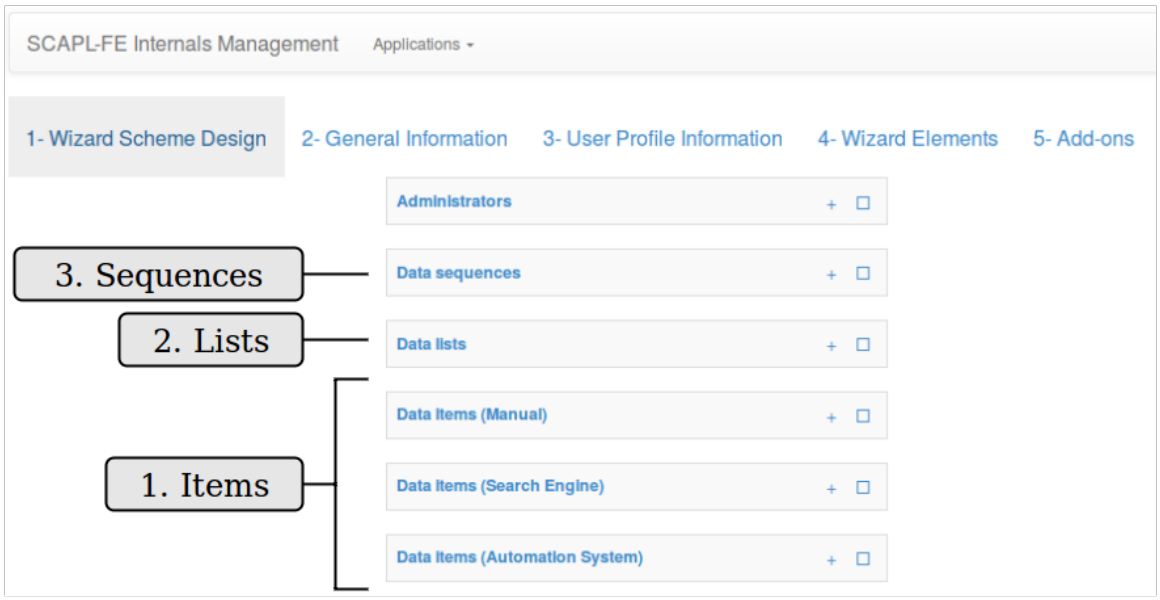


Figure 5.1: SCAPL Administration - Defining the data scheme

The data scheme is defined in the following order (as depicted in Figure 5.1) : DI, DL then DS.

The next subsections show a few screenshots that illustrate the administration interface for creating the different entities to build the scheme.

5.2.1 Items

At this point, he starts by creating the *Data Items (Manual)* (the identifiers are automatically assigned) like depicted in Figure 5.2. He must simply (a) fill in the fields and (b) click on the save button.

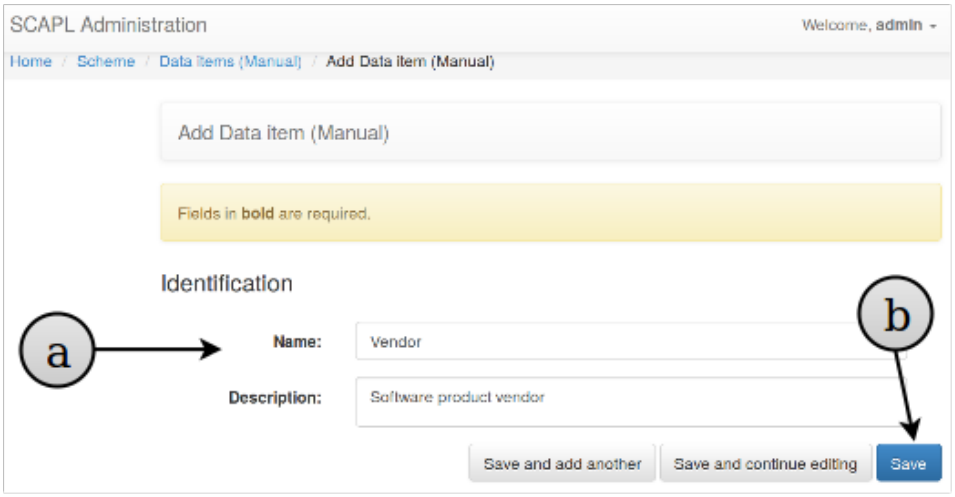


Figure 5.2: SCAPL Administration - Creating a manual *Data Item*

He applies the same procedure for every item in the list below.

ID	Name	Description
DI00001	Name	Complete name of the software product
DI00002	Vendor	Software product vendor
DI00003	Description	Product description
DI00004	Comment	Comments and conclusion of the assessor

Then, he creates the *Data Items (Search Engine)* related to the plugin GoogleScraper/CVE-Search like depicted in Figure 5.3. He must simply (a) fill in the fields, (b) fill in the API parameters (related plugin, keywords associated to the DI and the number of suggestions to be displayed) and (c) click on the save button.

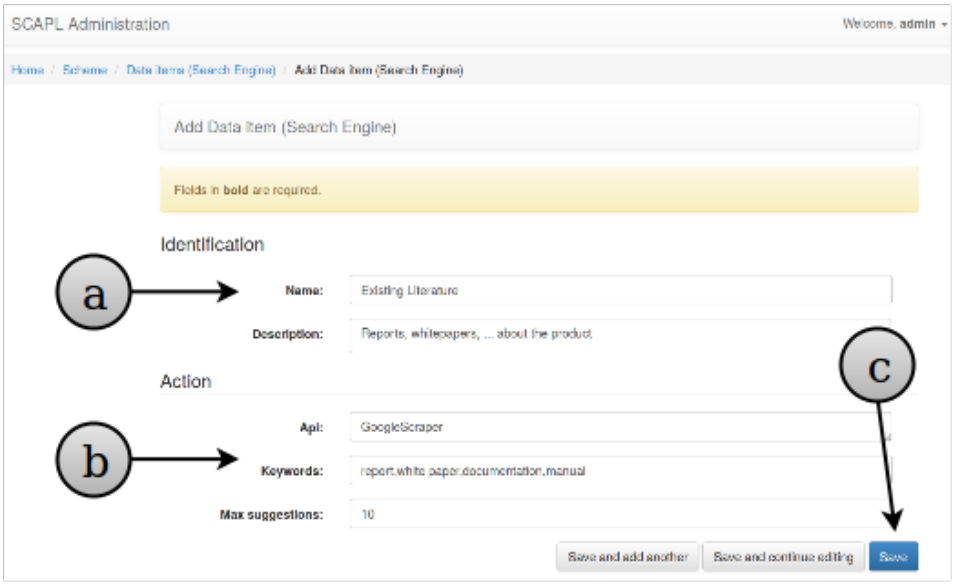


Figure 5.3: SCAPL Administration - Creating a search *Data Item*

He applies the same procedure for every item in the lists hereafter.

With the GoogleScraper plugin selected :

ID	Name	Description	Keywords	# (*)
DI00005	Website	Website of the software product vendor or the product repository	website	2
DI00006	Platform	Supported platforms	supported platforms	5
DI00007	License	Licensing information	license	5
DI00008	CPE	Common Platform Enumeration (for retrieving CVE's)	cpe	4
DI00009	Version	Current version of the product	last version	5
DI00010	System Requirements	System requirements for the given product	system requirements	5
DI00011	Language	Supported languages	supported languages	5
DI00012	Existing Literature	Reports, whitepapers, ... about the product	report,whitepaper, manual	10
DI00013	Integrity Validation	File integrity checking of the product's downloaded installation files	installer checksum	5

(*) Number of suggestions

With the CVE-Search plugin selected (no keywords require for this plugin) :

ID	Name	Description	# (*)
DI00014	Known Vulnerabilities	Found known vulnerabilities from common security sources of Common Vulnerabilities and Exposures (CVE)	10

(*) Number of suggestions

As a reminder, the interested reader can find the way to make a plugin for SCAPL applied to the example of CVE-Search in Appendix H, illustrating the ease of adding new capabilities to the system.

5.2.2 Lists

For now, Alan must create the *Data Lists* according to the table below and as shown in Figure 5.4 :

ID	Name	Description	Data Items
DL001	Product Identification	Base information about the product	DI00001, DI00002, DI00005, DI00006, DI00007, DI00008, DI00009
DL002	Product Description	Technical information about the product	DI00003, DI00010, DI00011, DI00012
DL003	Security Data	Software security assessment information	DI00013, DI00014, DI00004

The creation of the *Data Lists* is trivial as Alan must just (a) fill in some basic fields but, for the bindings of items, he can easily (b) add new items to the list, (c) use the drag-and-drop feature to order them and then (d) click on the save button.

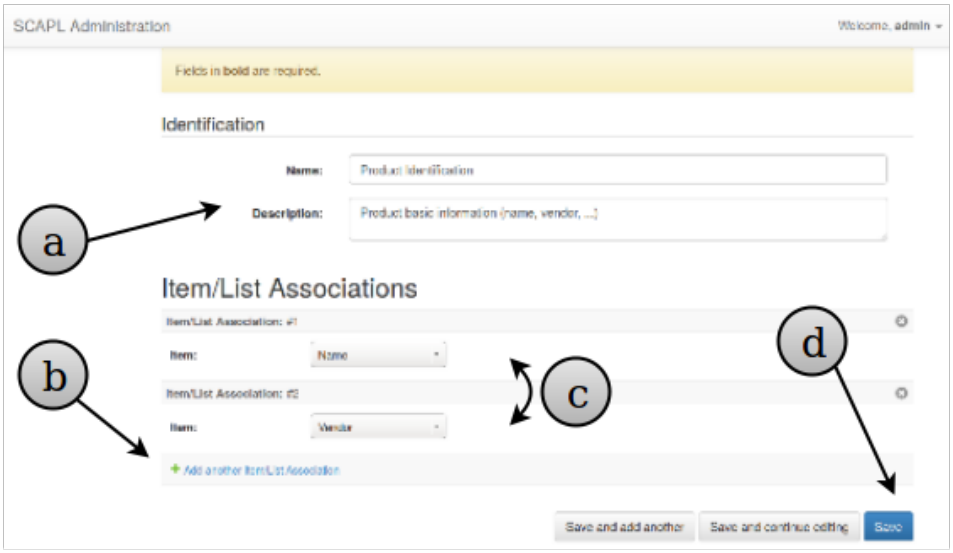


Figure 5.4: SCAPL Administration - Creating a *Data List*

5.2.3 Sequences

Third, Alan now creates the *Data Sequences* to chain the lists according to the table below and as shown in Figure 5.5 :

ID	Name	Description	Data Lists	Roles
DS01	General information	Base and technical information about the product	DL001, DL002	Manager, Technician
DS02	Security Assessment	Software security assessment information	DL003	Assessor

The creation of the *Data Sequence* is trivial as Alan must just (a) fill in some basic fields but, for the bindings of lists, he can easily (b) add new lists to the sequence and (c) use the drag-and-drop feature to order them. A last thing is to (d) add the right roles (e.g. *Manager*) to enable the sequence and finally (e) click on the save button.

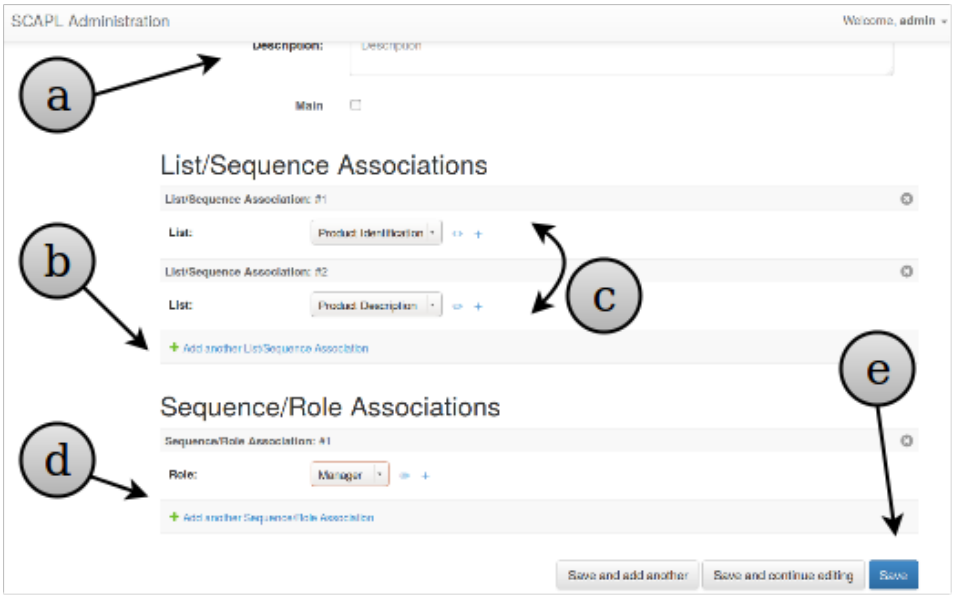


Figure 5.5: SCAPL Administration - Creating a *Data Sequence*

Once done, the wizard is now ready-to-use for each actor.

OBSERVATIONS

In order to set the **data scheme** up, it just takes Alan a **thirty minutes** to create all entities based on the inputs of Mike. Furthermore, this time is invested once as the scheme is now defined for every assessment. It can also very easily be enhanced later by just creating some new entities and by binding them together.

5.3 Experiments

This section shows some relevant features of SCAPL and their related benefits for each assessment planned by Mike. It also mentions some interesting results that can already be found with the plugins in place. As explained before, our observations for evaluating the added value of SCAPL address the collaborative way in which the actors can conduct an assessment and the time spared when using the system.

According to the given scenario, as each assessment always use the same structure on a common application and as the results are globally similar, we choose to associate each experiment with a specific evaluation goal, emphasizing each time a specific role and its related *Data List* (matching the related sub-processes as explained in SAPLAFv1 in Subsection 3.1.1) :

1. **CCleaner** : In this assessment, we emphasize the *Product Identification* filled in by Mike and the ease with which he creates a new assessment task and uses the wizard.
2. **Skype** : For this case, we show the *Product Description*, essentially filled in by Tom (while it could also be filled in by Mike if one remembers how the related *Data Sequence* is bound to the *Product Identification* and *Product Description* lists), and how a search *Data Item* could be refined to get a better result, in close coordination with Alan.
3. **Adobe Reader** : As we expect to find some critical known vulnerabilities about this product, we emphasize the *Security Data* filled in by Alan and his comments to conclude to product's rejection.

CONVENTION

In order to evaluate the time spare in a not too-rough way, we use some timing conventions, related to a search either manual or with SCAPL :

ID	Type	Action	Time
A1	Manual	Open Google, type keywords, wait for result to be loaded	20"
A2	SCAPL	Time required to load results NB: for SCAPL, this time is required once	5"
B	Both	Open a link and parse the page for interesting information	30"
C1	Manual	Copy a relevant text and format it, e.g. in a Word document	2′
C2	SCAPL	Copy-paste relevant objects from search frame to editor	20"

The estimated time required per item to perform a search and to fill in the item is then computed as follows :

Manual : A1 + B + N * C1 **SCAPL : B + N * C2**

where N is the number of results to be parsed for finding the first relevant information (meaning that, once found, the search stops).

5.3.1 CCleaner

APL Initialization Mike starts with CCleaner, a tool that allows to clean up a workstation by removing temporary files, unnecessary backup copies and various other maintenance operations.

He thus first connects to SCAPL, clicks on the APL menu and selects *New task* and then arrives at the starting point of the APL task (this opens an HTML object upon the page that is called a modal).

He then (a) fills the keywords in, (b) no package (as the assessment is anyway limited to *Testing Level 1 – Basic Search*) for the software product and (c) clicks on the *Create* button.

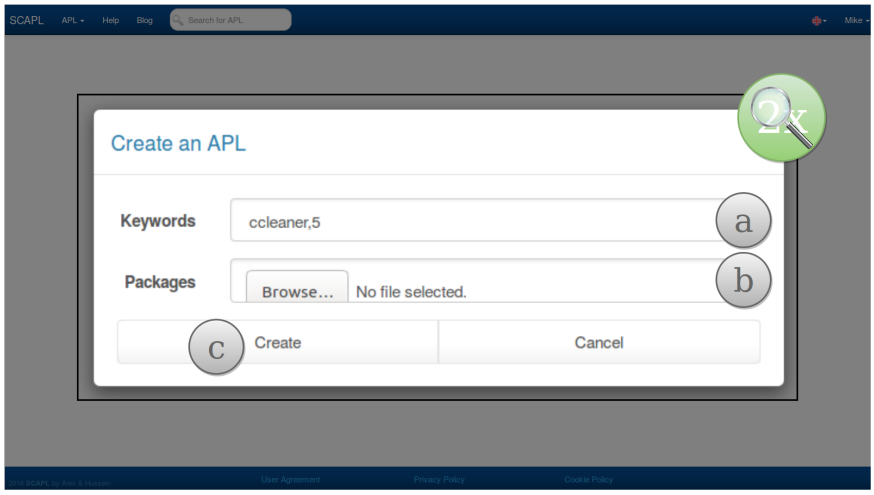


Figure 5.6: SCAPL Usage - Creating a new task

Now, he gets the main page of the APL file ; the overview.

He just has to (a) click on the frame of the *Data List* he wants to open as the current step in the wizard (this opens in a modal).

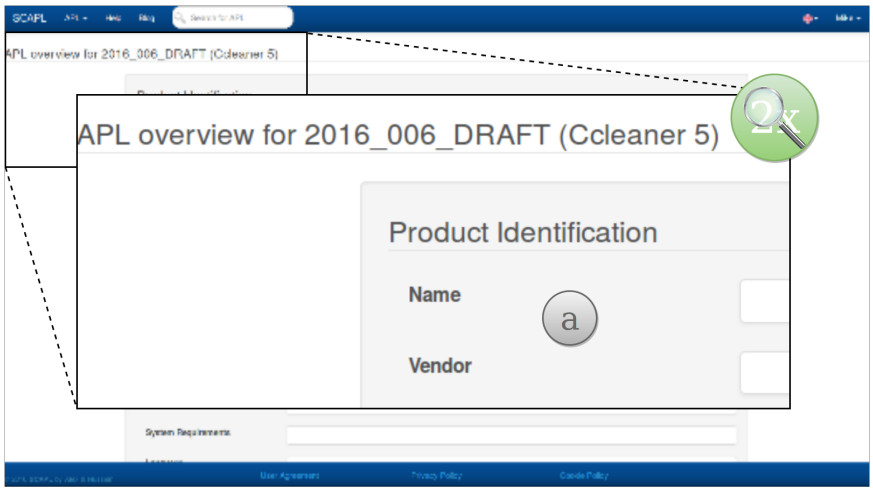


Figure 5.7: SCAPL Usage - APL task overview

Now, the wizard is opened and the real work can start.

He just has to (a) browse the suggestions (here, 2 as configured for DI00005) for finding the right data. Afterwards, he can (b) copy-paste the relevant information (text, images, links, etc) inside the editor and (c) refine the result before saving it by clicking on the top-right button of the editor frame.

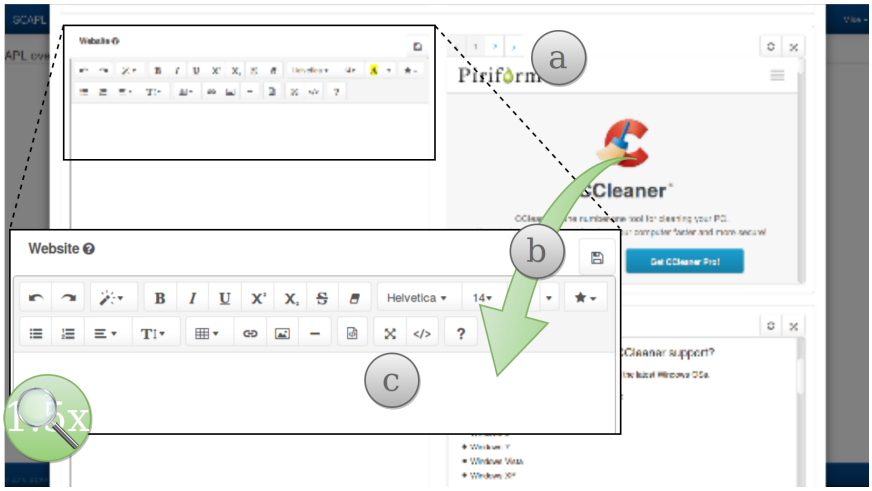


Figure 5.8: SCAPL Usage - APL task wizard

During his research, Mike (who cares for *Product Identification*) finds the following attributes for CCleaner :

Item	Value	Rank (*)
Name	CCleaner	0 (**)
Vendor	Piriform	0 (**)
Website	https://www.piriform.com	1
Platform	Windows XP, Vista, 7, ...	1
License	https://www.piriform.com/legal/software-license/ccleaner	2
CPE	cpe:2.3:a:piriform:ccleaner:5.00.5050	4
Version	5.21.5700	1

(*) Available as the Nth result in Google (manual) suggestion
(**) Manual item

OBSERVATIONS

Thus, given the timing convention, Mike required the following time to proceed :

- **Manual** : 2*C1 + 5*(A1+B) + (1+1+2+4+1)*C1 = 2*2′ + 5*50" + 9*2′ = **26′10"**
- **SCAPL** : A2 + 2*C2 + 5*B + (1+1+2+4+1)*C2 = 5" + 2*20" + 5*30" + 9*20" = **6′15"**

That is, using SCAPL took him **24%** of the time required to perform the same actions manually.

Product Preparation For the part of Tom, *Product Description*, the execution of his sub-process only concerns filling in some information given the *Testing Level* and no package preparation yet.

Security Assessment For the part of Alan, *Security Data*, he does not notice anything special egarding the security assessment except a former CVE that does not apply to the proposed version. Alan thus concludes this APL file with the status APPROVED.

5.3.2 Skype

APL Initialization This time, Mike creates the APL task for Skype, the well-known software that allows its users to call for free, to have a video conference or also to chat. He fills in his part, *Product Identification*, without any problem.

Product Preparation For the *Product Description*, Tom fills in his complete list (in the same way as illustrated in the previous subsection) except one item for which he realizes that it does not provide very relevant results. This concerns the *Existing Literature* which gives poor results. But he reminds that he has already seen an interesting report about Skype's internals so that he is sure that there should be relevant information.

Tom then contacts Alan (related to his power of SCAPL's administrator) to request for a refinement of the concerned item. After a bit of research on his side, Alan figures out that a Google dork (see Subsection 3.3.2) could be used to only target Portable Document Format (PDF) files.

Figure 5.9 (referring to the administration interface page illustrated in Figure 5.3) illustrates how to refine the related *Data Item* so that it uses a Google dork.

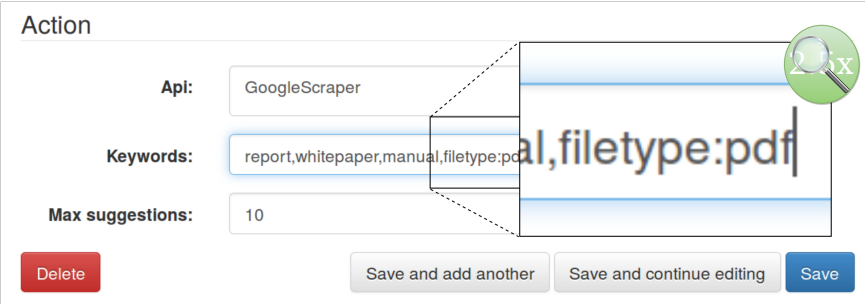


Figure 5.9: SCAPL Usage - *Data Item* refinement

By modifying the current item's keywords to :

*report,whitepaper,manual,**filetype:pdf*** instead of simply *report,whitepaper,manual*

... only PDF files will be selected through the GoogleScraper plugin.

He thus applies this change into SCAPL and Tom can now continue his work like explained in the experiment of CCleaner in the previous subsection.

During his research, Tom (who cares for *Product Description*) finds the following attributes for Skype :

Item	Value	Rank (*)
Description	This product allows to call for free...	0 (**)
System Requirements	Processor: Min 1 GHz – RAM: Min 512 Mo – ...	1
Language	(not found)	10
Existing Literature	(multiple links)	10

(*) Available as the Nth result in Google (manual) suggestion
(**) Manual item

OBSERVATIONS

Thus, given the timing convention, Tom required the following time to proceed :

- **Manual** : 1*C1 + 3*(A1+B) + (1+10+10)*C1 = 1*2′ + 3*50" + 21*2′ = **46′30"**
- **SCAPL** : A2 + 1*C2 + 3*B + (1+10+10)*C2 = 5" + 1*20" + 3*30" + 21*20" = **8′55"**

That is, using SCAPL took him **19%** of the time required to perform the same actions manually.

Security Assessment For the part of Alan, that is, *Security Data*, all items are filled in and nothing special is found regarding the security assessment except some known vulnerabilities on another similar product from the same vendor but not on the assessed one. Alan thus concludes this APL file with the status APPROVED.

5.3.3 Adobe Reader

APL Initialization For this experiment, once again, Mike starts the APL task (according to what is shown in the first experiment) for Adobe Reader XI, one of the most used PDF reader worldwide. He fills in his part, *Product identification*, without any particular problem.

Product Preparation Meanwhile, Tom cares for his part, *Product Description*, and also completes his work with no problem about the relevancy of the search results.

Security Assessment Then comes the assessment of Alan with his part, *Security Data*. He also reviews the information filled in by Mike and Tom. During the assessment, he opens the wizard and starts to fill in the *Integrity Validation* item. After defining the MD5 checksum of the application, he continues with the known vulnerabilities and is surprised while discovering several vulnerabilities that could relatively easily be exploited by an attacker if the product was installed in his organization's environment according to its current state. Indeed, CVE-search plugin provides some recent overwhelming CVE's, namely identified by CVE-2016-1009 and CVE-2016-0940, showing maximum severity scores.

The analysis is addressed after a few more information about the findings of Alan using SCAPL and the related timing observations.

During his research, Alan finds the following attributes for Adobe Reader :

Item	Value	Rank (*)
Integrity Validation	MD5: DFAB00BD67611E8377C31DB1D03765A4	5
Known Vulnerabilities	CVE2016-1009, CVE-2016-0940, ...	10
Comment	Multiple known vulnerabilities found [...]	0 (**)

(*) Available as the Nth result in Google (manual) suggestion
(**) Manual item

Note that, regarding the number of results to be parsed for the items of the *Security Data* list, we now consider that all the suggestions must be parsed (so, not up to the first relevant result anymore). Therefore, we consider the 10 first CVE's for the assessed product (as configured in the maximum of suggestions of the related *Data Item*).

Moreover, in the first instance, the checksum for the integrity validation was not found but, with another refinement thanks to a Google dork, he could finally find it on the website FileHippo [88], which provides checksums in various formats for miscellaneous recent software products.

OBSERVATIONS

Thus, given the timing convention, Alan required the following time to proceed (security assessment not included) :

- **Manual** : 2*(A1+B) + (5+10)*C1 + 1*C1 = 2*50" + 15*2′ + 1*2′ = **33′40"**
- **SCAPL** : A2 + 2*B + (5+10)*C2 + 1*C2 = 5" + 2*30" + 15*20" + 1*20" = **6′25"**

That is, using SCAPL took him **19%** of the time required to perform the same actions manually.

The parsing of the 10 first CVE's reveals a series of known vulnerabilities about the product addressing particularly worrying security concerns, especially arbitrary code execution (namely in CVE's CVE-2016-1009, CVE-2016-0940, CVE-2015-8458 and CVE-2015-7617).

All of these CVE's relate to arbitrary code execution which is a particularly frightening consequence for security professionals. The most serious vulnerabilities all completely impact the Confidentiality, Integrity and Availability, which represents the highest possible risk.

Furthermore, most of these vulnerabilities concern the network as a vector, and as the organization should effectively be connected to the Internet (as most of the organizations today), the overall risk remains very high.

So, looking at Figure 5.10, an overview of CVE-2016-1009 can be found with, especially, the CVSS[1] value at its maximum for the three used criteria, that is, 10.0. This CVE addresses a vulnerability discovered on a more recent version and confirms that it impacts former versions, thus including this of the assessed product. Note that the same applies for the three other previously mentioned CVE's.

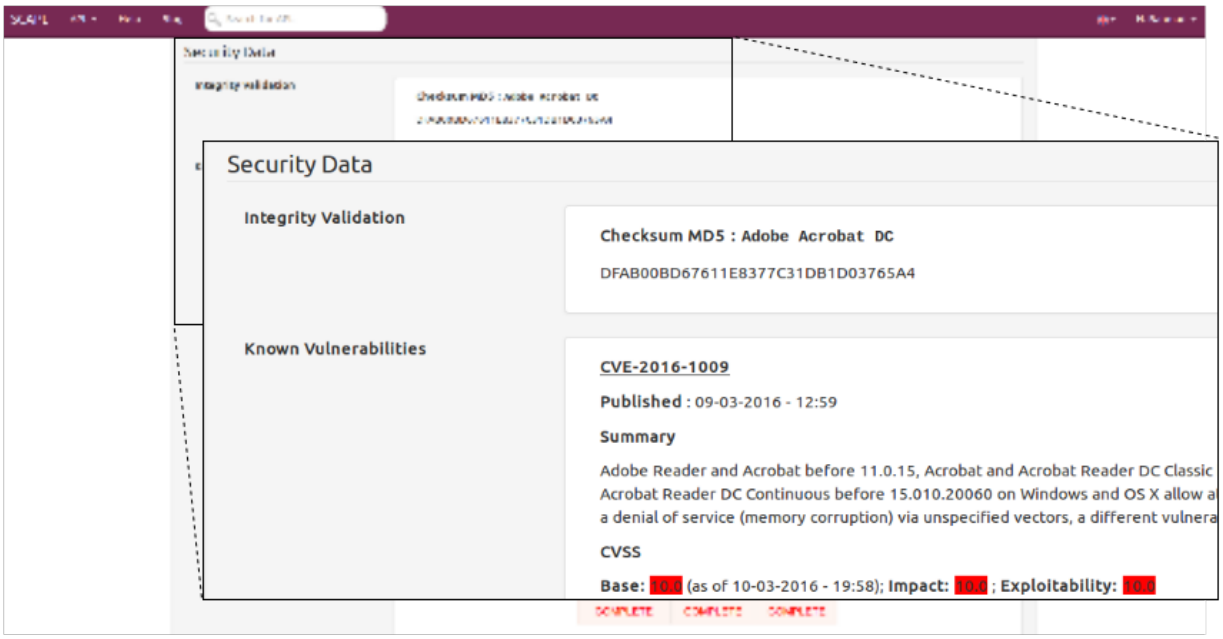


Figure 5.10: SCAPL Usage - Security-related *Data List*

In the current situation, Alan does not want to take any risk and therefore decides to conclude this APL file with the status REJECTED. He also warns Mike of this result and advices him to search for an alternative product fulfilling the same functional requirements.

The interested reader can refer to Appendix I for an example of APL report (that is, the related APL file in its final state) about Adobe Reader.

As a final note ; Alan really enjoyed working with SCAPL and he plans to write his own plugins in order to add some new data items for growing the scale of his security assessment. For this purpose, he has already some ideas to quickly collect some more security-related information. For example, he could write a plugin to search for reports provided by websites such Hybrid-Analysis [89], VirusTotal [90], etc. for the purpose of displaying existing vulnerability analysis reports.

[1] Common Vulnerability Scoring System

SUMMARY

Starting from the context of an ordinary organization wishing to invest in its IT security and especially in the scope of Software Security, **SAPLAFv1 and SCAPL can be exploited** in their current versions **to implement the APL process** that assesses the security of software products used in organization's environment with a certain level of assurance.

SCAPL aims to provide the following **benefits** to organizations :

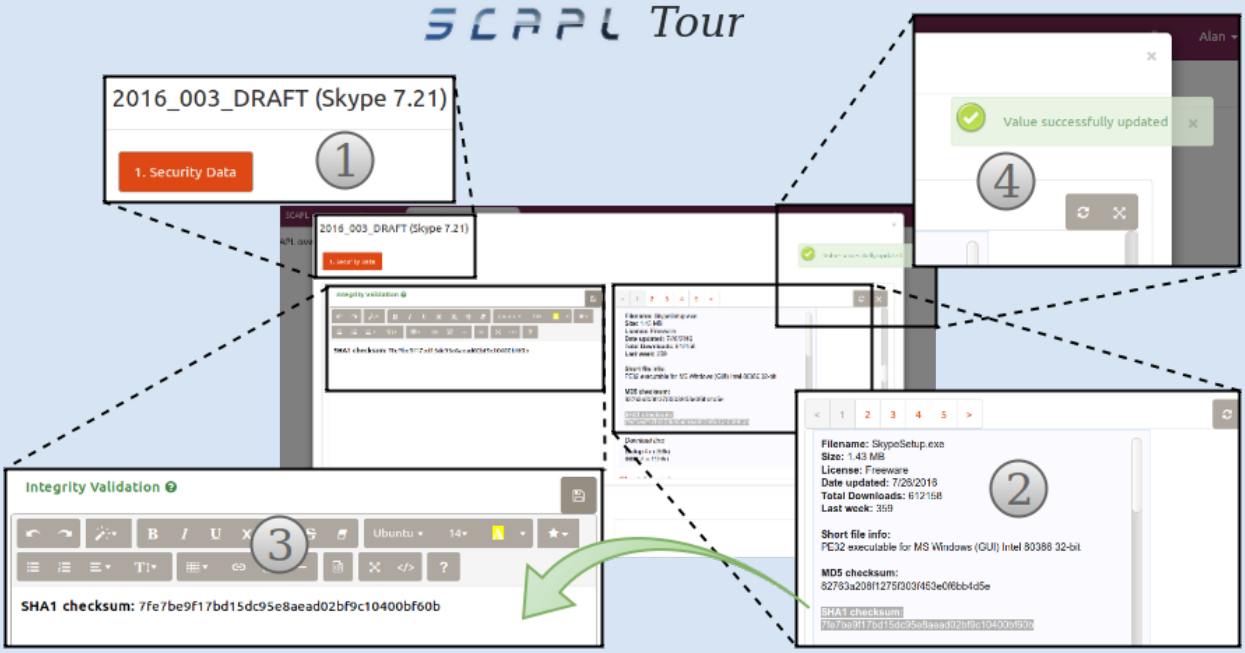
- The **data scheme** can **easily and quickly** be **configured** so that assessments can promptly start.
- Using SCAPL's **wizard** adapted **per role** allows to perform assessments in a **collaborative** way.
- Using the **wizard allows time savings** as everything is in a single interface and **search computations** are done **by SCAPL**.

By learning the APL process from SAPLAFv1, **roles and responsibilities** can **easily** be **assigned**. After a bit of reading, its implementation becomes straightforward by installing SCAPL with only a few efforts and defining its actors according to the defined roles. From this point, the following methodology in **three steps** is applied :

1. The process' **manager defines the wanted structure** for the assessment files.
2. SCAPL's **administrator translates** this structure **into a data scheme** and binds these to roles.
3. The actors then **apply the APL process** in order to conduct the assessments.

By applying these steps, the **structure** of the assessments **can** thus **be tuned according to** management's **requirements** with ease **and** can also be **iteratively refined**. Furthermore, **new plugins can be** internally **developed** so that new security controls can be applied.

For the easy of use during assessments, a picture says more than a thousand words...



1. The wizard is adapted to each actor so that he/she can play his/her role.
2. A search frame with (if configured so) a list of suggestions for each item is displayed, providing the result from SCAPL's computations. The information can then simply be copy-pasted.
3. The sophisticated edition box (based on Summernote [86]) allows to tune the result at will. Then clicking on the *Save* button allows to immediately asynchronously update the task.
4. Once done, the wizard can be closed and the result is immediately visible on the page.

DISCUSSION

In this chapter, a **simple assessment structure** is chosen relying on the currently existing plugins of SCAPL so that relevant security data can already be found **for assessing common software products**. The items of this structure are surely very simple and essentially **based on search features** but they relate to information that should be part of any software assessment file. Through this structure, **roles** are **assigned to** some **parts** of it and everybody can only care for his/her own part while collaborating on the same APL tasks.

From the **explanations and illustrations** that were presented in this chapter, we can now **confirm that** :

1. Whatever our scenario, the **data scheme** is really **straightforward to configure** and thus shows that such a **flexible** structure **can** help organizations to **tailor** the solution **to** their security **requirements**.
2. The **collaborative factor** greatly **helps** in the execution of the APL process, **harmonizing** the **interactions** of the stakeholders in the assessment process.
3. SCAPL's **wizard features** together (essentially ergonomics, information copy-paste between search results and an HTML editor) **ensure** a drastic **time saving** regarding the time it should take to perform all the researches manually.

From the experiments, we can point out that, regarding search quality and time savings, the following **observations** apply :

- Either manually or with SCAPL, **some researches could return** not very or even **irrelevant results** but, of course, it depends on the used search engine. However, the search ***Data Items***, even if complex to configure correctly at first sight, **can be easily refined** in a iterative way **through feedback** from the end-users to the system administrator(s).
- Regarding the timing convention and computations, the following table summarizes what we figured out, the **time required with SCAPL** being the proportion of time required regarding the same actions done manually :

Data List	Time with SCAPL
Product Identification	24%
Product Description	19%
Security Data	19%
Weighted Average	**21.5%**

  Given this result, **SCAPL** surely helps to **bring efficiency in software security assessment**.

  Note that, as it remains general and empirical, the computation presented in each experiment is the same regardless of the APL file.

However, we must emphasize the fact that the used plugins relate to search features through the *Systematization* mode of operation. We expect that, when we will implement **plugins using the *Automation* mode**, the time saving **will be even better** than what we observed during our experiments as these new plugins should cover far more time-consuming tasks, especially achieved by the *Assessor*.

Moreover, these **new plugins** should provide to SCAPL even **more relevant security features** so that the **quality of** software product security **assessments** can be greatly **improved**. As a first example, we will integrate, in a near future, projects such as OpenVAS (see Subsection 6.4 about the future works).

6 Conclusion

"If you think technology can solve your security problems, then you don't understand the problems and you don't understand the technology."

Bruce Schneier

Technology cannot provide 100% security for any problem whatever its sophistication. However, it can be used to add new layers of protection so that the level of security becomes acceptable. Software security assessment tools are technologies that can help to increase it, when providing sufficient data so that the assessors can reliably decide on the opportunity of using software products in their organization. SCAPL does not provide additional layers itself but allows to collect information by sticking multiple projects together, each solving particular problems, and by offering a collaborative solution for multiple security actors.

This chapter concludes this thesis by giving a summary emphasizing our contributions, parsing the achieved objectives and presenting the fulfilled learning outcomes. It finally provides multiple ideas of future works.

6.1 Summary

All along this thesis, we worked on reading various documents and acquiring or improving multiple skills in order to trim our scope so that we can provide our contribution to the field of Software Security Assessment in two ways ; by building an adapted framework and by implementing a dedicated open-source system upon it.

First, we parse a broad background to select some relevant standards and guidelines to be able to build our own framework. But yet another framework, the reader could say... SAPLAFv1 aims to propose a simple process, the software APL one, whose implementation is based on some targeted standards and guidelines, e.g. NIST SP 800-115 and the Australian government's ISM. But it also encompasses concepts, principles, techniques, mechanisms and technologies that could help to improve the effectiveness and efficiency of this process so that it is finally possible to build an open-source system upon it.

Afterwards, we designed and implemented a system from scratch, SCAPL, that is able to apply SAPLAFv1 so that the throughput of the APL process can be increased through the use of a convenient and user-friendly Web interface, supported by various mechanisms and technologies such as asynchronous tasking in order to make the system perform various time-consuming and error-prone computations in place of the end-users.

Finally, we put ourselves in a crafted scenario where we could show the benefits of SCAPL such that we could evaluate its maturity. While keeping its promises in terms of extensibility, ease of use, performance, ergonomics (and so forth), some efforts should now be given to the elaboration of new plugins so that SCAPL's capabilities can be increased, especially in the field of security assessment techniques.

6.2 Objectives

Our objectives were three-fold :

1. The **background** is fully stated.
   - A – We reviewed the current literature through facts & figures, standards, guidelines, books, articles, online sources and security solutions from a general point of view about information security to the particular field of software security.
   - B – We found various standards and especially selected NIST SP 800-115 and the Australian government's ISM.
   - C – We presented some existing solutions and used a few ones in SCAPL.
2. Our **framework**, SAPLAFv1, is ready.
   - A – The main process and its main sub-process about security assessment are stated.
   - B – Various concepts, principles, methods and techniques are included for supporting the process.
   - C – We mapped every new concept to the previous ones so that these can be used as a whole.
3. Our **system**, SCAPL, is ready in its prototype state.
   - A – Its design is complete and fulfils its requirements.
   - B – Two ready plugins exist, GoogleScraper and CVE-Search, and others are already being tested.
   - C – The features were parsed for fulfilment of the requirements.
4. Some simple **experiments** were presented for showing the power of SCAPL.
   - A – A full methodology including a context and a related scenario was outlined.
   - B – An example of instantiation of the APL process was given.
   - C – The results of experiments were discussed regarding the chosen software products.

6.3 Reached Outcomes

The level of fulfilment of the learning outcomes can be assessed as follows :

1. **Knowledge and skills** : were critical to the progress of this thesis due to the broadness of the related field and they were hugely improved in order to reach our objectives.
2. **Engineering approach** : was essentially related to the realization of our system according to what we learned about the software development during our studies.
3. **Research work** : was about parsing of a large literature and learning various projects (security ones as well as small Web projects for integration in our system) in the required domains and selecting the right things.
4. **Organization** : was implemented through regular meetings, follow-up reports, regular workout sessions and deliverables allowing us to distill the workload mixed with this of the other courses across the academical year.
5. **Communication** : was applied through the meetings, deliverables and a rigorous writing style.
6. **Impact** : is ensured by our public open-source project on GitHub.

6.4 Future Works

As told on several occasions along this dissertation, we already have a few ideas and reflections on how we could improve SCAPL :

- **Automation plugins** : Notwithstanding the correction of little bugs that surely hide in our implementation yet, our primary concern will now be to address the creation of plugins for the Automation System.
- **Integration of Kali tools in SCAPL** : This is the subject of an internship titled *Automating vulnerability assessment tests with Python and Kali tools* agreed between the Cybersecurity service of the Belgian Defence and a student of the university of Mons. This aims to study how we could use some particular tools from Kali Linux so that we could orchestrate them in either mode of operation. The focus should be given first to OpenVAS.
- **Addition of a new component handling the *Interactive* mode** : This will match each component to each mode of operation as the Search Engine and Automation System now respectively handle the *Systematization* and *Automation*. Currently, the *Interactive* mode is pseudo-handled by the front-end with its integrated project, GateOne. As a new component, it will be easier to manage links from the front-end to multiple test environments or external testing system (like a Kali appliance).
- **SCAP integration** : SCAP is presented in the chapter about the background. It is partially used through CVE-Search as it already handles three specifications from SCAP (CVE, CPE, CVSS). But it could be more used provided that it is deeper studied for implementation. This could for example lead to the use of compliance checking and standardized reporting specifications. A future project will address this problem.
- **Data mining** : Some information researches could be interdependent through some learning mechanisms so that some particular ones providing poor results could be accordingly refined. Such a study could become an internship subject in a not-too-distant future.
- **Use of BPMN[1] in IT processes** : This is a more general subject but could eventually lead to potential improvements in SCAPL (especially regarding how more complicated external systems could be operated in an efficient way). This is already addressed as an internship subject.

[1] Business Process Model and Notation : Graphical notation that sketches the steps of a business process

References

[1] ISECOM. *Open Source Security Testing Methodology Manual (OSSTMM) version 3*. 2010.

[2] David Hoelzer. *AUD507: Auditing & Monitoring Networks, Perimeters & Systems*. SANS Institute, 2015.

[3] ICS-CERT. *Cyber Threat Source Descriptions*. URL: https://ics-cert.us-cert.gov/content/cyber-threat-source-descriptions (visited on 07/31/2016).

[4] S Keshav. "How to read a paper". In: *ACM SIGCOMM Computer Communication Review* 37.3 (2007), pp. 83–84.

[5] Olivier Bonaventure. *LINGI2349 – Network and communication seminar*. Université Catholique de Louvain. 2015.

[6] (ISC)^2. *About the (ISC)^2 CBK®*. URL: https://www.isc2.org/cbk/ (visited on 08/05/2016).

[7] (ISC)^2. *About (ISC)^2*. URL: https://www.isc2.org/aboutus/ (visited on 08/02/2016).

[8] Johannes Ullrich. "2015 State of Application Security: Closing the Gap". In: *SANS Institute InfoSec Reading Room* (2015).

[9] Jim Bird, Eric Johnson, and Frank Kim. "2016 State of Application Security: Skills, Configurations and Components". In: *SANS Institute InfoSec Reading Room* (2016).

[10] Stuart Jacobs. *Engineering Information Security: The Application of Systems Engineering Concepts to Achieve Information Assurance*. 2nd. Wiley-IEEE Press, 2016. ISBN: 978-1-119-10160-4.

[11] SANS Institute. *SANS Security Training and Your Career Roadmap*. 2016. URL: https://www.sans.org/media/security-training/roadmap.pdf (visited on 07/30/2016).

[12] Abraham Nyirongo. *Auditing Information Systems: Enhancing Performance of the Enterprise*. Trafford Publishing, 2015.

[13] Gary McGraw. *Software Security: Building Security In*. Addison-Wesley Professional, 2006.

[14] ISO. *About the ISO27k standards*. URL: http://www.iso27001security.com/html/iso27000.html (visited on 08/05/2016).

[15] TechTarget. *WhatIs.com – ISO 27001*. URL: http://whatis.techtarget.com/definition/ISO-27001 (visited on 08/02/2016).

[16] Vinod Vasudevan et al. *Application Security in the ISO27001 Environment*. It Governance Ltd, 2008. ISBN: 1905356358, 9781905356355.

[17] ISO. *27005:2011: Information security risk management*. 2011.

[18] ISO. *27034-1:2011: Application Security – Part 1 : Overview and Concepts*. 2014.

[19] NIST. *NIST Special Publications (SP)*. URL: http://csrc.nist.gov/publications/PubsSPs.html#SP%20800 (visited on 08/06/2016).

[20] Rhand Leal. *How to use the NIST SP800 series of standards for ISO 27001 implementation*. 2016. URL: http://advisera.com/27001academy/blog/2016/05/02/how-to-use-the-nist-sp800-series-of-standards-for-iso-27001-implementation/ (visited on 08/09/2016).

[21] ENISA. *ISO/IEC Standard 15408*. URL: https://www.enisa.europa.eu/topics/threat-risk-management/risk-management/current-risk/laws-regulation/rm-ra-standards/iso-iec-standard-15408 (visited on 08/07/2016).

[22] Australian Government Department of Defence. *Information Security Manual*. URL: http://www.asd.gov.au/infosec/ism/ (visited on 08/02/2016).

[23] Australian Government Department of Defence. *Strategies to Mitigate Targeted Cyber Intrusions*. URL: http://www.asd.gov.au/infosec/mitigationstrategies.htm (visited on 07/08/2016).

[24] OWASP. *Open Web Application Security Project*. URL: https://www.owasp.org/index.php/Main_Page (visited on 08/14/2016).

[25] OWASP. *Testing Guide 4.0*. 2016.

[26] Center for Internet Security. *Secure Configuration Benchmarks*. URL: https://benchmarks.cisecurity.org (visited on 06/25/2016).

[27] Alexandre D'Hondt and Hussein Bahmad. "Understanding SCAP Through a Simple Use Case". In: *Hakin9 IT Security Magazine* 11.3 (2016), pp. 100–110.

[28] S. Radack and R. Kuhn. "Managing Security: The Security Content Automation Protocol". In: *IT Professional* 13.1 (2011), pp. 9–11. ISSN: 1520-9202. DOI: 10.1109/MITP.2011.11.

[29] NIST. *Security Content Automation Protocol Specifications*. URL: https://scap.nist.gov/revision/1.2/ (visited on 11/29/2015).

[30] NIST. *7275 Rev 4: Specification for the Extensible Configuration Checklist Description Format (XCCDF) Version 1.2*. 2011. URL: http://csrc.nist.gov/publications/nistir/ir7275-rev4/NISTIR-7275r4.pdf.

[31] NIST. *National Vulnerability Database*. URL: https://nvd.nist.gov (visited on 07/27/2016).

[32] Red Hat. *OpenSCAP Portal*. URL: https://www.open-scap.org (visited on 07/28/2016).

[33] Offensive Security. *Kali – Our Most Advanced Penetration Testing Distribution, Ever.* URL: https://www.kali.org (visited on 08/05/2016).

[34] Offensive Security. *Training, Certifications and Services*. URL: https://www.offensive-security.com (visited on 08/03/2016).

[35] *Kali Linux Tools Listing*. URL: http://tools.kali.org/tools-listing (visited on 08/05/2016).

[36] Rapid7. *Metasploit*. URL: https://www.metasploit.com (visited on 08/20/2016).

[37] The Volatility Foundation. *Volatility Framework - Volatile memory extraction utility framework*. URL: https://github.com/volatilityfoundation/volatility (visited on 08/20/2016).

[38] Pentoo team. *Pentoo*. URL: http://www.pentoo.ch (visited on 08/05/2016).

[39] OpenVAS team. *OpenVAS – The world's most advanced Open Source vulnerability scanner and manager*. URL: http://www.openvas.org (visited on 08/06/2016).

[40] Quarkslab. *IRMA – Incident Response and Malware Analysis*. URL: http://irma.quarkslab.com (visited on 08/06/2016).

[41] Microsoft. *Attack Surface Analyzer*. URL: https://www.microsoft.com/en-us/download/details.aspx?id=24487 (visited on 08/06/2016).

[42] Microsoft. *Microsoft Baseline Security Analyzer 2.3 (for IT Professionals)*. URL: https://technet.microsoft.com/en-us/security/cc184924.aspx (visited on 08/06/2016).

[43] CIS. *CIS-CAT and Other CIS Benchmark Assessment Tools*. URL: https://benchmarks.cisecurity.org/downloads/audit-tools/ (visited on 08/21/2016).

[44] P De Filippi. "Cloud computing: analysing the trade-off between user comfort and autonomy". In: *Internet Policy Review* 2.2 (2013).

[45] Qualys. *Network Security and Vulnerability Management*. URL: https://www.qualys.com (visited on 08/07/2016).

[46] Tenable Network Security. *CyberScope*. URL: https://www.tenable.com/solutions/cyberscope (visited on 08/05/2016).

[47] Carbon Black. *Arm Your Endpoints*. URL: http://www.carbonblack.com (visited on 08/07/2016).

[48] Splunk. *Where Big Data Meets Big Ideas*. URL: http://www.splunk.com (visited on 08/07/2016).

[49] NIST. *SP 800-115: Technical Guide to Information Security Testing and Assessment*. 2008.

[50] Stephen Northcutt. *Security Laboratory – Security Controls*. URL: http://www.sans.edu/research/security-laboratory/article/security-controls (visited on 08/12/2016).

[51] Cynthia K. Veitch, Susan Wade, and John T. Michalski. *Cyber Security Assessment Tools and Methodologies for the Evaluation of Secure Network Design at Nuclear Power Plants*. Sandia National Laboratories, 2012.

[52] NIST. *National Checklist Program Repository*. URL: http://checklists.nist.gov (visited on 08/14/2016).

[53] NIST. *United States Government Configuration Baseline*. URL: https://usgcb.nist.gov (visited on 08/02/2016).

[54] TechTarget. *WhatIs.com – Google dork query*. URL: http://whatis.techtarget.com/definition/Google-dork-query (visited on 08/09/2016).

[55] NikolaiT. *GoogleScraper – Scraping search engines professionally*. URL: https://github.com/NikolaiT/GoogleScraper/ (visited on 08/05/2016).

[56] David J. Hand, Padhraic Smyth, and Heikki Mannila. *Principles of Data Mining*. MIT Press, 2001. ISBN: 0-262-08290-X, 9780262082907.

[57] Wikipedia. *Data mining*. URL: https://en.wikipedia.org/wiki/Data_mining (visited on 08/13/2016).

[58] cve-search. *CVE-Search – A tool to import CVE and CPE into a MongoDB to facilitate search and processing of CVEs*. URL: https://github.com/cve-search/cve-search/ (visited on 08/11/2016).

[59] ToolsWatch. *vFeed - The Correlated Vulnerability And Threat Database*. URL: https://github.com/toolswatch/vFeed (visited on 08/01/2016).

[60] Offensive Security. *Exploit Database Archive*. URL: https://www.exploit-db.com (visited on 08/02/2016).

[61] IBM. *IBM Security Services 2014 – Cyber Security Intelligence Index*. URL: http://www.slideshare.net/ibmsecurity/2014-cyber-security-intelligence-index (visited on 08/09/2016).

[62] ISO. *27001:2013: Information Security Management*. 2013.

[63] Punit Dwivedi and S Christobel Diana. "Analysis of Automation Studies in the Field of Information Security Management". In: *Analysis* 6.12 (2013), pp. 60–63.

[64] ISO. *19464:2014: Advanced Message Queuing Protocol (AMQP) v1.0 specification*. 2014.

[65] VMware. *VMware Virtualization for Desktop & Server, Application, Public & Hybrid Clouds*. URL: http://www.vmware.com (visited on 08/13/2016).

[66] Oracle. *VirtualBox*. URL: https://www.virtualbox.org/ (visited on 07/03/2016).

[67] HashiCorp. *Vagrant – Development environments made easy*. URL: https://www.vagrantup.com (visited on 12/19/2015).

[68] Red Hat. *Ansible, Automation for Everyone*. URL: http://www.ansible.com (visited on 12/22/2015).

[69] Puppet Labs. *Puppet*. URL: https://github.com/puppetlabs/puppet (visited on 12/22/2015).

[70] IEEE. *830-1998: Recommended Practice for Software Requirements Specifications*. 1998.

[71] IEEE. *1016-2009: Standard for Information Technology–Systems Design–Software Design Descriptions*. 2009.

[72] Kim Mens. *LINGI2255 – Software Engineering Project*. Université Catholique de Louvain. 2015.

[73] Alan Cline. *Agile Development in the Real World*. Apress, 2015. URL: https://books.google.be/books?id=L4dNCwAAQBAJ.

[74] Paul Reed. *Reference Architecture: The best of best practices*. URL: http://www.ibm.com/developerworks/rational/library/2774.html (visited on 08/17/2016).

[75] Django Software Foundation. *Django: The Web framework for perfectionists with deadlines*. URL: https://www.djangoproject.com (visited on 07/03/2016).

[76] Pivotal Software. *RabbitMQ - Messaging that just works*. URL: https://www.rabbitmq.com (visited on 07/08/2016).

[77] Alexandre D'Hondt and Hussein Bahmad. *SCAPL – Security assessment for Computing APL*. URL: https://github.com/dhondta/ (visited on 08/18/2016).

[78] Mischback. *django-project-skeleton*. URL: http://django-project-skeleton.readthedocs.io (visited on 07/05/2016).

[79] Web development community. *Bootstrap – The most popular HTML, CSS, and JS framework for developing responsive, mobile first projects on the web*. URL: http://getbootstrap.com (visited on 02/27/2016).

[80] Ask Solem and contributors. *Celery : Distributed Task Queue*. URL: http://www.celeryproject.org (visited on 03/21/2016).

[81] liftoff. *GateOne – An HTML5 web-based terminal emulator and SSH client*. URL: https://github.com/liftoff/GateOne/ (visited on 07/13/2016).

[82] Thomas Park. *Bootswatch – Free themes for Bootstrap*. URL: https://bootswatch.com (visited on 04/26/2016).

[83] Microsoft Development Network. *How to Safeguard your Site with HTML5 Sandbox*. URL: https://msdn.microsoft.com/en-us/hh563496.aspx (visited on 04/27/2016).

[84] Django Software Foundation. *Django Documentation – Performance and optimization*. URL: https://docs.djangoproject.com/ja/1.9/topics/performance/ (visited on 05/03/2016).

[85] Django community. *Django Packages : Reusable apps, sites and tools directory*. URL: https://www.djangopackages.com (visited on 05/07/2016).

[86] Summernote Team. *Summernote – Super Simple WYSIWYG Editor on Bootstrap*. URL: http://summernote.org (visited on 03/24/2016).

[87] Django Software Foundation. *Django Documentation – Security in Django*. URL: https://docs.djangoproject.com/en/1.9/topics/security/ (visited on 05/07/2016).

[88] FileHippo. *FileHippo*. URL: http://filehippo.com (visited on 08/21/2016).

[89] Payload Security. *Hybrid Analysis*. URL: https://www.hybrid-analysis.com (visited on 08/21/2016).

[90] VirusTotal. *VirusTotal*. URL: https://www.virustotal.com (visited on 08/21/2016).

[91] ISO. *27034-2:2015: Application Security – Part 2 : Organization Normative Framework*. 2015.

[92] Cuckoo team. *Cuckoo Sandbox: Automated Malware Analysis*. URL: https://www.cuckoosandbox.org (visited on 08/06/2016).

[93] VMWare. *Workstation Pro*. URL: http://www.vmware.com/fr/products/workstation.html.

[94] ISO. *Standards catalogue*. URL: http://www.iso.org/iso/iso_catalogue/catalogue_tc/ (visited on 08/08/2016).

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve **www.uclouvain.be/epl**