D1.3 State of the art report

Contract number:	957212
Project acronym:	VeriDevOps
Project title:	Automated Protection and Prevention to Meet Security Requirements in DevOps Environments
Delivery Date:	31/03/2021
Coordinator:	ABO
Partners contributed:	All
Release Date:	31.3.2021
Version:	01
Revision:	01 (IKER, MI)
Abstract:	This deliverable will update the state-of-the art wrt to new approaches and technologies that appeared since the project proposal was submitted.
Status:	• PU (Public)

X

Table of Contents

List of Acronyms

A&T	Analysis and testing
AIDS	Anomaly-based Intrusion Detection Systems
BERT	Bidirectional Encoder Representations from Transformers
CAPEC	Common Attack Pattern Enumeration and Classification
CASE	Computer-Aided Software Engineering
CERT	Community Emergency Response Team
CI	Continuous Integration
CICD	Continuous Integration and Continuous Delivery
CID	CVE ID Detector
CiRM	Security Commits Regression Model
CMDB	Configuration Management Database
CNN	Convolutional Neural Networks
COTS	Commercial Off-The-Shelf
CPE	Common Platform Enumeration
CPN	Customer Premises Network
CRM	Comment Complexity Regression Model
CSP	Communicating Sequential Processes
CVE	Common Vulnerability and Exposure
CVSS	Common Vulnerability Scoring System
DDoS	Distributed Denial of Service
DevOps	Software Development and IT Operations
DoS	Denial of Service
DT	Decision Tree
ECJ	Java-based Evolutionary Computation
ePurse	Common Electronic Purse
GA	Genetic Algorithm
GP	Genetic Programming
GPS	Global Platform Specification
HIDS	Host Intrusion Detections System
HLPSL	High-Level Security Protocol Language
HMI	Human Machine Interface
ICS	Industrial Control Systems

ICT	Information and Communication Technology
IDPS	Intrusion Detection and Prevention Systems
IDS	Intrusion Detection System
IHDS	Intelligent Harvesting Decision System
IoT	Internet-of-Things
IPC	Information Processing Component
ISMS	Information security Management System
IT	Information Technology
LDA	Linear Discriminant Analysis
LR	Logistic Regression
MBST	Model-based Security Testing
MBT	Model-Based Testing
MIT	Massachusetts Institute of Technology
ML	Machine Learning
NBC	Naive Bayes Classifier
NFR	Non-Functional Requirements
NFR-C	NFR classifier
NIDS	Network Intrusion Detection System
NIST	National Institute of Standards and Technology
NL	Natural Language
NLP	Natural Language Processing
NVD	National Vulnerability Database
OISF	Open Information Security Foundation
OSS	Open Source Software
OT	Operational Technology
OWASP	Open Web Application Security Project
PenTest	Penetration Testing
PLC	Programmable Logic Controller
POS	Part of Speech
PSO	Particle Swarm Optimization
PUTs	Parameterized Unit Tests
QoS	Quality of Service
RBAC	Role-Based Access Control
RCA	Root Cause Analysis
RE	Requirements Engineering

ROI	Return On Investment
SAMM	Software Assurance Maturity Model
SAT	Boolean Satisfiability
SBSE	Search-based Software Engineering
SBST	Search-based Security Testing
SCADA	Supervisory Control And Data Acquisition
SCAP	Security Content Automation Protocol
SDL	Security Development Lifecycle
SET	Social Engineering Toolkit
SLA	Service Level Agreement
SMT	Satisfiability Modulo Theories
SOAP	Simple Object Access Protocol
SOORs	Seamless Object-Oriented Requirements
SRM	Stakeholder Complexity Regression Model
SUT	System Under Test
THOE	The Threat Oracle Engine Tool
UML	Unified Modelling Language
URM	Security URLs Regression Model
V&V	Verification and Validation
VAPT	Vulnerability Assessment and Penetration Testing
VM	Vulnerability Management
XSS	Cross-Site Scripting

Executive Abstract

This deliverable provides an overview of the state-of-the-art related to methods and tools for automated generation of security requirements, prevention at development and reactive protection at operations. The overview builds upon and extends the ambition described in the Full Project Proposal. In addition, it identifies needs for improvement from the surveyed literature and highlights the directions on which the innovation activities of the VeriDevOps project will focus on. For brevity, the deliverable does not explicitly include the work performed in other similar European research projects, but the results of such projects are discussed via their published research papers and datasets. This deliverable will provide input for all the other technical work packages in the project and enhance the innovation of the project's exploitable key results.

1. Introduction

The VeriDevOps project assumes a DevOps development process, starting from the initial security requirements of the system under test, ensuring preventive security at development and last but not least providing reactive protection at operations. Therefore, the current deliverable D1.3 is divided in three corresponding parts:

- the *automated generation of formal security requirements* section, in which we discuss the use of natural-language processing for security requirements, datasets for security requirements and formal patterns for security requirements specification.
- the *reactive protection at operations* section, in which we address overview techniques for vulnerability scanning, signature-based intrusion detection, runtime monitoring and detection using search-based testing, anomaly detection based on machine learning and root-cause analysis. We complement the above with proposed methods for remediation.
- the *prevention at development* section, where we describe techniques used to enforce security aspects at development. For instance, we overview techniques for formal analysis and verification for security, security testing methods (model-based security testing, penetration testing, mutation and fuzz testing approaches, and search-based security testing), localization and debugging techniques, and the integration of security and software development life cycle.

Each of the above sections is concluded with an analysis of research gaps towards industrial needs and suggestions on which of those needs will be addressed in the VeriDevOps project.

The works discussed in this deliverable do not include review of deliverables of other projects related to VeriDevOps. Instead, we have attempted to identify academic publications in which the results of such projects have been disseminated.

2. Automated Generation of Formal Security Requirements

2.1. Context and Motivation

We would like to start this section by laying down the terminology and presenting the context of Requirements Engineering.

Requirements Engineering (RE) is a crucial element in software development to meet customers' expectations for a software product that should be delivered on time and within a budget. Practically, RE enables capturing users' needs for the system to be developed by transferring these needs into precise and clear statements that will be the basis for design, development and validation [1].

An important part of developing any system is ensuring a required level of security. Security needs are usually associated with some resources or assets involved in a system that stakeholders naturally want to protect from any harm. In particular, assets are considered as all the information resources that are stored or accessed by the system or physical resources such as computers. In some cases, assets may consist of other assets, e.g. system backups are a good example. Despite that empirical evidence is not fully convincing, it appears that appropriate security requirements would have a positive impact on system security as sufficient general requirements would have on system development success.

In order to integrate security within requirement engineering, we usually have to consider separately security requirements [2]. Specialised research showed that early analysis of security requirements can be beneficial in the context of software development as this may enable cost reductions in the area of 12-21% [3]. Usually, security requirements are processed as functional requirements that can considerably influence system architecture. In practice, this requires specific security expertise. Nowadays, security requirements are handled by specially trained people, independently from other requirements. This does not map well to the reality in which the systemic concerns are tightly connected (e.g., we cannot discuss security in isolation from safety). With the invention of DevSecOps, the situation is starting to change. In the same way as DevOps has unified the two formerly distinct silos, development and operations, DevSecOps is supposed to break the security silo and merge it into DevOps. The resulting process will guarantee security-by-design, but it calls for advanced tool support that is only starting to emerge now. The whole process of manual identification or extraction of security-related requirements from an entire requirements specification is very complex and error-prone, causing the need for automatic analysis. This is associated with several practical challenges.

Firstly, there is no exact definition for security requirements, since different people may interpret security requirements in various ways. In practice, different industry subjects - organizations define security requirements based on their own conventions and templates. Secondly, the intrinsic ambiguity of natural language makes it even more complicated to identify security requirements. Primarily, various people may use different syntax and terms to define or describe security requirements [4] [5].

The main point is that security cannot be considered as just a quality requirement, as it is difficult to answer whether a problem is security-related or not. Usually, stakeholders do not tolerate any kind of risk. The main task of security requirement engineering is to identify, and document requirements for developing secure software systems. The identification of security requirements heavily depends upon the context of the system and the analyst's assumptions. These assumptions can be explicit or implicit and relate to expectations over the system or environment behavior with a significant impact on the security of a system. Considering the framework of security, its goals and assumptions, one can define security requirements as constraints on the functions of the system, where these constraints define one or more security goals. In other words, security requirements ensure security goals by constraining the system's functional requirements. Security requirements, like functional requirements, are prescriptive, providing a specification to achieve the desired effect [6].

Based on these goals we conclude that we want to minimize chances for any threat or potential attack aimed at our assets. We consider assets as something that is valuable to an organization (e.g. resources, data) and typically is the main concern of security requirements. Practically, a security property determines a security characteristic (e.g. confidentiality, availability) that indicates a security objective that a requirement intends to achieve. A threat is a risk that a swindler may potentially exploit a vulnerability to attack the system, harming assets or their respective security properties. In this context, a countermeasure is considered as a solution that should address security requirements. A countermeasure fundamentally combines security constraints that must hold in the developed system, and the particular security mechanisms that are supposed to guarantee these constraints. A constraint may take the form of either a restriction on the system, assurance of some specific properties, permission to do certain actions, or regulation to comply with the established norms of the society where the system will be used. Thus, we can define security requirements as a substantial concept that is connected through specific relations (Fig. 1). Each concept shows some perspective of the security requirements and implies a possible way of defining the concept accordingly [4].

In order to follow this structure and achieve initial goals, we must somehow determine whether those requirements have been satisfied. This is difficult for quality requirements in general while security requirements present additional challenges [6]. An important element of requirements engineering is associated with the role of natural language (NL). Despite that there is no proof that natural language is the best option, multiple evidence shows that it is the most common way of expressing requirements in industry practice. The dominance in describing and specifying software and system requirements in natural language was also confirmed by recent research [7]. Therefore, based on the past and current empirical evidence, we can safely assume that NL will continue to serve as the common way of expression for requirements in the future as well [8]. Conceptually, that implies that solutions should deal with problems like lexical, syntactic and pragmatic problems that natural language poses for requirements engineering. It is stated that the biggest problem is ambiguous semantics, which remains a common challenge for practitioners arguing that the source of trouble is the information from which the requirements must be formulated [9]. Consequently, computer-aided software engineering for processing natural language looks promising in the context of requirements analysis [8].

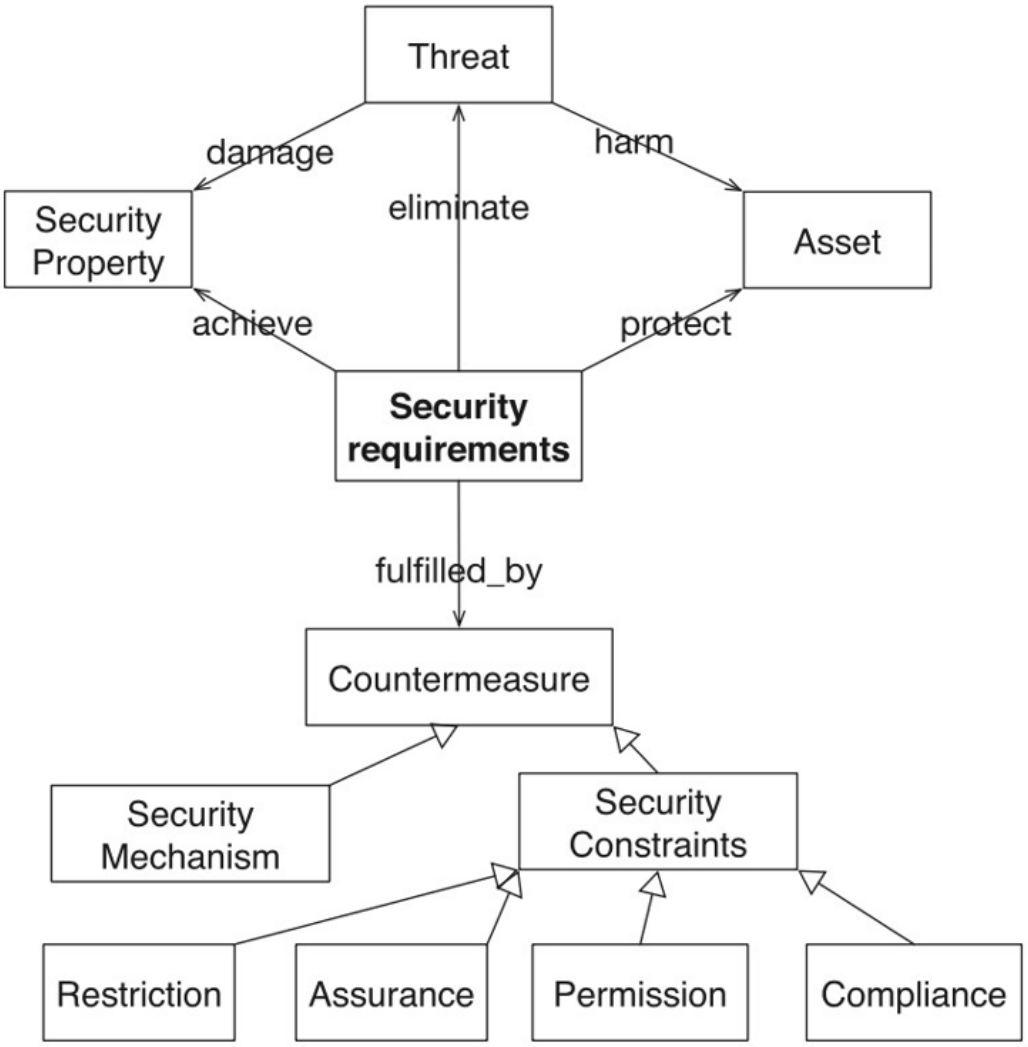

Figure 1: A conceptual model for security requirements, adapted from [4]

Applying Natural Language Processing (NLP) techniques that are very well suited for comprehensive linguistic analysis seems natural in the context of the engineering approach that suggests using linguistic tools to narrate descriptions of user requirements. NLP is a field that addresses various approaches in which computers can deal with natural, that is, human language. Usually, NLP deals with techniques for analyzing, representing naturally-occurring texts for the purpose of achieving human-like language processing for a range of tasks or applications [10] [11]. This has led to the emergence of a separate field of applying NLP to support requirement engineering processes as well as various tasks at different RE phases [12]. Dealing with the inputs to the RE process is a complicated task, as it requires analyzing a wide variety of documents. Such documents might include different artifacts like interview transcripts, codes of practice, standards, legislation, etc. In practice, the methods for RE automation greatly differ depending on the stage of RE they are applied at. To illustrate the difference, at later stages, such as requirements validation, the methods deal mainly with

documents that are products of the RE process, whereas at early stages the methods typically process raw information [9]. By applying those methods, the engineers intend to solve different kinds of tasks like detecting language issues, identifying key domain concepts and establishing traceability links among requirements, etc. However, when we split the developed NLP solutions by problem that they solve, they are mainly focused on detection, classification, clustering, patterns extraction and modeling [12]. Those instruments are intended to increase analysts' productivity when working with requirements.

Having requirements expressed in natural language is not enough to implement these requirements and verify the implementation against them. Formalization of security requirements is necessary to rigorously validate and verify candidate designs and implementations against requirements.

2.1.1. Key NLP Methods

Let us outline the key method categories. **Classification** task in machine learning (ML) is usually associated with predicting a categorical class [13]. As for the context of RE, this task aims at classifying different categories of requirements. For example, we can classify requirements based on their functional category or based on their quality category, to identify non-functional requirements that may be hidden within functional ones. Another example is applying classification to users' feedback in order to identify new requirements referring to specific features of interest possibly including sentiment analysis. **Extraction** generally tries to retrieve some specific single or multi-word terms from requirement texts for domain or project glossaries, as requirements usually contain complex terms that are not commonly used. Those extracted glossaries may be further applied for other problems including consistency checking, classification, modeling or product comparison. **Clustering** or cluster analysis, as its name suggests, is focused on organizing data, in our case, documents into some cohesive subsets or clusters. Methods are focused on organizing the data into meaningful and useful information. **Detection** typically deals with ambiguities in requirements to make them clearer and unequivocal. The range of problems may include detection of different lexical issues from the debatable usage of grammatical rules, to the occurrence of vague phrases (e.g., after some time), weak verbs (e.g., may, might), and the appearance of syntactic ambiguities. In addition, some specific tasks such as following predefined templates and recognizing equivalent requirements can also be included in this task, as the main goal is still to maintain correctness to requirements texts. **Modeling** relates to the extraction task, but with some additional usage of extracted data like generation of Unified Modeling Language (UML) models to support analysis, design, feature synthesis in product-line engineering, generation of models for early requirements and generation of software tests to maintain a necessary security level [12].

In addition to the above-mentioned generic problems that NLP solves for requirement engineering, one can outline several approaches that are entirely focused on the security context. Despite the lack of studies in this area, we can highlight initial progress in developing and implementing such systems. Security risks can be analyzed through different perspectives that will define a practical context of the problem. **Vulnerability detection** is focused on identifying vulnerable

software components prior to deployment, either by statically analyzing software code, or by executing security testing tools on a running instance of the software. The approach concentrates on applying NLP techniques to code to prevent or identify various vulnerabilities in the code. **Vulnerability repair** tries to transform a vulnerable code into a non-vulnerable code by learning from a set of source examples. Millions of lines of legacy code are analyzed to identify the ways to improve security. When a new class of vulnerability is found, the training dataset for patches and fixes is quickly updated. This is intended for creating an automated system that can clean code with certain types of vulnerabilities that would allow it to efficiently treat large software repositories. Finally, **specification analysis** assumes that we can deal with security risks in products before the code is even written. Recent advances in NLP have provided expert methods to automatically process vulnerability descriptions or product specifications to assess security risks. Instead of code, we can apply such methods to requirement documents and textual vulnerability descriptions in this paradigm to ensure a required security level for the developed software [14]. Our main interest is associated with this security perspective.

2.1.2. NLP Models Evaluation Process

As it was shown previously, we deal with various methods for applying NLP to analysis of security requirements. However, practically each method is focused on a specific aspect of processing security requirements. The process of applying these methods includes several phases such as training phase, validation phase and testing phase. On the first stage, methods based on ML principles require training using input datasets to adjust parameters of a model. During this process training errors are measured to know how well the model is fit to the data. Still, we want to know how well a model learns from the data to predict unseen data and this is where the testing phase comes for [15]. Practically, we could hold some part of the data as a testing set to use it for a performance calculation. However, sometimes it is not possible due to scarcity of data. One way to address this problem is to apply K-fold cross-validation, which uses part of available data to feed the model, and a different part to test it [13]. Consequently, we can use each part for training and then for testing phases to calculate an average for a specific measure. The last principle is very common in the area of Natural Language Processing for Requirements Engineering (NLP4RE). This research mainly consists of studies that are focused on the classification problem. Classification performance itself is being measured by such metrics as accuracy, precision, recall and F1-score. Accuracy is defined as a ratio between correctly classified samples to the total number of data samples, precision shows the proportion of positive samples that were correctly classified to the total number of positive samples, recall represents positive correctly classified samples to the total number of positive samples. To summarize overall performance, F-measure (e.g. F1-score) is applied, which represents the harmonic mean of precision and recall accordingly [15].

2.2. NLP Methods and Models

Each type of task for applying NLP to security requirements that were mentioned in the previous section refers to certain models and methods that we will consider below. This outline covers a discussion of methods for classification, extraction as well as advanced architectures for the methods

based on transfer learning. This section is divided into three parts that separately cover statistical and basic ML approaches, deep learning and transfer learning, as well as methods for dealing with security risks through specification analysis.

2.2.1. Statistical and Classical Machine Learning Methods

NLP addresses several practical problems in the area of Requirements Engineering. To start, let us consider the problem of distinguishing functional requirements from non-functional ones. Abad et al. [16] propose text preprocessing as the main tool of dealing with that task. To address the generalization problem for the input requirements texts, they proposed to preprocess the texts and replace all context-based names related to products and users with general keywords, such as 'PRODUCT' and 'USER', respectively. Then they apply the Part-Of-Speech (POS) tagger of the Stanford Parser [17] to assign parts of speech to each word in each requirement. In the next step, they extract some trivial features including number of adjectives, number of adverbs and number of cardinals, as well as specific metrics, such as number of degree adjectives to adverbs. In addition, for each feature, they define its rank based on the probability of its occurrence in the requirements. The final feature list for the processed dataset consists of the following nine features: number of cardinals, adverbs, adjectives, modal words, determiners, verbs, prepositions, singular nouns, and plural nouns. In [16] the authors compare results of six different algorithms and use a simple decision classifier to achieve an extra 4.5% accuracy of classifying functional and non-functional requirements. This effect becomes even more visible for classifying groups of requirements. Abad et al. insist that Binarized Naive Bayes works best for classifying non-functional requirements.

Another example of NLP application to Requirements Engineering is identifying critical features in specifications. Boutkova et al. [18] propose a lexical analysis-based technique that could help automate the identification of features in specifications. They propose to extract features in a semi-supervised fashion by applying certain Part-of-Speech (POS) tagging approaches. The whole process is divided into several steps. At the first step, the user chooses the specification in which the features must be found. At the second step, requirements from the chosen specification get decomposed into individual words, and only nouns are left; this step requires lemmatization of each word. At the final step, the user should evaluate the candidate's list and choose features for the feature model. The main problem is that the experiments were conducted for German – a morphologically complex language. This approach generates a lot of false positives that need further analysis.

It is possible to improve the performance by combining different NLP developments from different disciplines. Malhotra et al. [19] proposed an approach combining NLP, ML, and graph analysis. This approach identifies appropriate narrative structures that may underlie the security requirements of industry standards and publicly available software documents. First, the authors of [19] apply text processing that includes tokenization, sentence splitting, POS tagging, morphological analysis, noun phrase chunking. Then they create an ontology to define connections between words, phrases, and concepts. They construct features from key narrative structures - phrases, such as 'user must register', 'user must contain a password', 'password must have complexity' using a special tool called Protégé [20]. Subsequently, each of these processing structures is used to determine the relationships among

features such as “encryption” or “authentication”. After, it is checked whether the requirement sentence is found among gold standard requirements. That way it is determined whether an organization follows those standards.

The idea of checking whether security requirements conform to specific standards was also presented in Hayrapetian et al. [21]. This study might be considered as an advancement of the previous study, focusing on empirically evaluating the conformance of security requirements to specific standards like ISO and OWASP. The main goal was to assess completeness and ambiguity by creating a bridge between the requirement documents and their compliance with standards. For this purpose, they proposed a unique two-stage architecture. Initially, every statement within a standard is evaluated against every statement within a test document. To maintain the robustness of an entailment assessment, they proposed nine different configurations and digested each pair through those components. Each configuration consisted of the Linguistic Analysis Pipeline and Entailment Decision Algorithms from Excitement Open Platform [22]. The entailment decision and confidence results from each transaction were collected along with other data about the transaction, such as the statements involved, entailment configuration used, processing type (e.g. parallel), and the time duration of the comparison. These annotations were used as features during the neural network model training phase to design a classifier to further determine whether the entailment results for a statement pair indicate a “complete”, “ambiguous”, or “none” match, with respect to the corresponding semantic meaning. This approach allowed us to achieve 0.79 in terms of F1 score (F1 = TP / (TP + 0.5 * (FP + FN)), where TP stands for “true positives”, FP -- “false positives”, and FN -- for “false negatives”).

One of the main challenges on the way of making all-purpose NLP methods is the problem of generalization of a model to be applied to several domains. Li et al. [4] present the idea of creating a model that could generalize security requirements extraction for all domains. They stated that the main source of good detection lies on a good theoretical basis and tried to construct ontology specifically for security requirements. They defined a set of linguistic rules and security keywords that are normally used to describe security requirements and used them to train classifiers applying classical ML algorithms. They proposed a specific approach that involves a two-level preprocessing with a conceptual layer and linguistic layer. The process of matching the linguistic features consists of three steps: generate parse trees, keyword matching and linguistic rule matching. Each step is explained in detail as a part of text processing to a feature vector. They decided to compare different algorithms like Decision Tree (DT), Naive Bayes Classifier (NBC) and Logistic Regression (LR) using six different datasets. Results showed that Precision/Recall differs among datasets. Only DT and LR showed promising characteristics. In particular, the average F1 score of all classifiers trained with DT was approximately 0.77. For the case of classifying security requirements from different domains, when training data was used from one document set and the test data from the other, this approach showed 0.75 in precision and 0.58 in recall. The authors argued that their approach behaves significantly better than the existing approach and potentially can give promising results. They also argued that the main challenge was that different people, including security experts, can have various diverse definitions of security requirements.

Another example of dealing with the generalization problem is the work by Wang et al. [23]. They address aspects of generalization from a different perspective such as creating all-domain classifiers.

The authors developed methods for extracting security requirements for open source projects (OSS). They stated that previously proposed approaches were unsuitable for this kind of project due to their specifics. Notably, requirement specifications in OSS projects are usually organized by functionality, with non-functional (NFR) requirements scattered widely across multiple documents. Hereby there is no exact boundary to distinguish between FRs and NFRs. Moreover, the requirements stored in issue tracking systems are unstructured and seldom obey grammar and punctuation rules. The authors proposed to rely not on the text, but on different external resources. To define features, they applied a stack of several sources that then were used as an input for a linear classifier based on Linear Discriminant Analysis (LDA). Initially, each requirement is processed by an Information Processing Component (IPC) to obtain so-called metrics. Metrics are information about a requirement extracted by IPC, which includes complexity and external resources. Complexity is extracted from comments of the project assuming that higher intensity of discussion might be associated with vulnerabilities. In its turn, external resources are the links and other references provided by stakeholders where they discuss the rationale for refinements and explain their solutions. Subsequently, this information is digested directly by four regression models: Comment Complexity Regression Model (CRM), Stakeholder Complexity Regression Model (SRM), Security URLs Regression Model (URM), Security Commits Regression Model (CiRM) In addition, the authors apply NFR classifier (NFR-C) and CVE ID Detector (CID). Each regression model generates a weight between 0-1 for each requirement that signifies the likelihood of whether this requirement is a security requirement. In order to summarize weights from NFR-C, CID, and all RMs, the authors applied a linear discriminant function in a binary setting that indicates whether a requirement is a security one or not. They were able to achieve F1 scores of 0.83, 0.88, 0.81 for Axis2, Drools, GeoServer projects respectively, which looks promising given the relative simplicity of the proposed approach.

In 2017, the RE Data Challenge event was conducted in relation to the problem of requirements extraction and classification. This event produced a set of NLP4RE studies. Kurtanovic et al. [24] used the dataset from the challenge [25] to solve the problem of binary classification for functional (FR) and nonfunctional (NFR) requirements. Simply, they transformed a multiclass dataset into a binary case. Unlike previous papers, the authors did make a research of an effect from applying only word features and automatically chosen features on binary and multiclass classification. The whole approach is based in the Support Vector Machines. As a result, they achieved an F1-score of 0.92 for binary cases classifying FR and NFR. As for classifying security requirements in a binary case, the effect was a bit worse. If applying only words features, F1-score was about 0.88 and 0.74 with applying all kinds of features. They found that POS tags are among the most informative features, with cardinal number being the best single feature. As an additional aspect, authors argued that only word features provide higher recall for classifying NFRs than employing additional syntax and meta-data features, but lower precision accordingly.

2.2.2. Deep Learning and Transfer Learning Methods

Deep learning and transfer learning are the most recent areas in NLP. Deep Learning is assumed as a sub-area of neural networks in ML and is popular for vision-based classification and NLP tasks. Deep

Learning is based on the representation-learning methods obtained by applying non-linear modules that transform a representation at one level into a higher, more abstract level [26]. Zhang and Wallace [34] proposed a convolutional neural network [27] for the purpose of sentence classification. They provided a simple method that is based on Word2Vec representations [28] of each word by applying a set of consecutive convolution filters. Specifically, the process starts with tokenized sentences which are converted to a sentence matrix, the rows of which are word representations. By this approach authors achieved significant accuracy improvements compared with a baseline on all datasets. A similar approach was applied to the context of requirements classification. Winkler et al. [29] applied the same principle to the DOORS requirements database. Specifically, they applied it for the binary classification task to differentiate requirements from information sentences. This approach was able to classify requirements with a precision of 0.73 and a recall of 0.89 and information with a precision of 0.90 and a recall of 0.75 accordingly. The authors argued that performance could be improved by increasing the amount of training data as well as by improving the quality of requirement specifications. A similar approach was applied to the previously mentioned NFR dataset. Dekhtyar et al. [29]presented an idea of combining two methods that are very popular at the moment, Word2Vec and Convolutional Neural Networks (CNN). They used two datasets, SecReq dataset [5] and The Quality Attributes (NFR) dataset [25], to compare results of applying Word2Vec with CNN with a baseline approach. The goal was to observe the performance of CNNs on these datasets compared to the baselines and measure the impact of pre-trained Word2Vec embeddings on the model. As a baseline method they considered an already mentioned approach based on Naive Bayes Classifier [5] with TF-IDF and Word Counts as feature vectors. For the SecReq dataset applying Word2Vec provided an overall boost in scores. By applying 30 filters with 100 training epochs, they scored an F1-score of 91.34%. This configuration allowed to achieve an overall improvement up to 13.5% compared to the baseline. For the NFR dataset, Word2Vec again contributed comparable improvement with 50 filters and 100 epochs accordingly. The authors stated that CNN classifiers can be successfully applied on relatively small collections of requirement documents to identify various requirements properties.

Recently the transfer learning method was applied as a new promising approach to deal with generalization problems. Hey et al. [30] stated that the performance of existing automatic classification methods decreases when applied to unseen projects because requirements usually vary in formulation and style. This means that such systems are impractical to use, as they are either overfit for a specific dataset, which is heavily relying on wording and sentence structure, or require a processing step (usually manual) for new text samples. Moreover, usually, authors do not report whether their approaches are able to generalize or do not generalize sufficiently to be practically applicable. One reason is the lack of available training data in the requirements engineering community. The authors stated that a possible solution can be found in transfer learning. Nowadays transfer learning approaches are heavily used in NLP. They are trained on huge datasets to capture underlying concepts and meanings of natural language texts. Afterward they can be adapted and fine-tuned to a specific task. The authors stated that this helps to overcome the problem of generalization, as these approaches promise both better performance and generalizability with less training data. That is achieved by fine-tuning of Bidirectional Encoder Representations from Transformers (BERT) [31], a language model based on deep learning. BERT, which is pre-trained on a large text corpus, can be

fine-tuned for specific tasks by providing only a small amount of input data such as requirements classification in our case. For experiments, NFR dataset [25]was chosen as a gold standard coming from RE Data Challenge'17. This whole process is common for BERT-based studies. Specifically, the BERT model is applied with a single layer of NN for classification purposes. The resulting model is called NoRBERT. The authors provided detailed information about experiments, which would help to replicate their results in the future. For binary tasks, NoRBERT achieved comparable results with an F1-score of 90% for functional and 93% for non-functional requirements. As it was expected, the BERT-based method outperformed all existing approaches at the moment. Specifically, NoRBERT outperforms all approaches that do not preprocess the data and, at the same time, the problem with unweighted data does not significantly impact performance. Thus, the transfer learning approach clearly increases the performance for classifying requirements. As for Security Requirements, NoRBERT was able to achieve about 0.91 in F1-score given multilabel classification, which might look promising for a further application.

2.2.3. Specification Analysis Approaches

Analyzing the role of security requirements, one can argue that they define new features or additions to existing features to solve a specific security problem or eliminate a potential vulnerability. This perspective is usually focused on revealing potential vulnerabilities of the system based on a specification even before coding starts. This is implemented by vulnerability detection, which enables specification analysis – the latter is associated with document analysis relying on NLP techniques to be applied to various textual sources. The section provides interesting examples of such an approach.

Bozorgi et al. [32] present the idea of applying ML to classify exploitable and non-exploitable vulnerabilities. The authors argued that the main goal was to create a method that could help to prioritize work on patches for new vulnerabilities. Their technique relies on an NLP technique known as 'bag-of-words'. For example, they record whether particular tokens like "buffer", "heap", or "DNS" appear in specific text fields like "title", "solution", or "product name". In addition, they used meta-data extracted from each vulnerability description, which gave them about 90 000 different descriptors for each vulnerability. Labels for each vulnerability were preliminary extracted from the corresponding section of descriptions. To classify exploitable and non-exploitable vulnerabilities, they applied a linear Support Vector Machines algorithm and achieved about 90% accuracy, which seems to be decent given the simplicity of the applied approach.

Applications themselves may represent a security risk. Pandita et al. [33] focused on detecting potential risks related to the usage of an application by analyzing the application description. Specifically, the work concentrated on the analysis of permissions required for a given application. The authors examined whether application descriptions provide any indication for why a given application would need those permissions. The authors propose the Whyper tool that takes a description of the application from the market, digests it to a semantic model and determines which sentence could indicate the use of permission. Thus, the tool tries to catch a semantic model inside an application's description. The purpose is to raise awareness for security and privacy problems and lower the sophistication required for concerned users to take control of their applications. This tool was tested

with a dataset describing 581 popular applications. Authors argued that Whyper can effectively identify the sentences that describe needs for permissions with an average precision of 82.8% and an average recall of 81.5%.

2.3. Datasets for Security Req

In this section, we will present an explanation for each dataset that we have found during our literature review process. Given that our interest is focused on designing systems for extraction of security requirements, an important step is to collect as much data as possible to represent this domain in the best way.

Name of the dataset	Information on a domain and dataset structure
PROMISE NFR dataset[1]	The PROMISE NFR dataset is commonly used in the community and addressed in the RE'17 Data Challenge. The 625 requirements include 255 functional and 370 non-functional requirements.
PURE dataset[2]	Dataset of Public Requirements Documents comprising 79 available resources. The documents are labeled to the fields, which provide the required information.
SecReq dataset[3]	A dataset containing 3 Requirements Documents, where each requirement is labeled either as Security related or non-related. In total 187 security and 323 non-security requirements.
OSS projects: Apache Axis2/Java (Axis2), Drools, and GeoServe	The dataset originated from an effort to build a classifier to distinguish security requirements from non-security ones in open-source projects. Links: Axis2/Java , Drools , GeoServe
Healthcare set of documents CCHIT (Certified 2011 Ambulatory EHR Criteria, Emergency Department Information Systems Functional Document)[4]	Criteria for Healthcare products certification published by CCHIT. 283 functional and non-functional requirements, including security class.
Slankas PROMISE-like[5]	11,876 requirements and 3568 of them are labeled with 14 classes, including security class, compiled from several

[1] https://zenodo.org/record/268542#.YEHj9i0Rq_s
[2] https://zenodo.org/record/1414117#.YEHkOi0Rq_s
[3] https://gist.github.com/iambackend/e8c68469c79204872ce475a64f663973
[4] https://bit.ly/3InPxOI
[5] https://github.com/RealsearchGroup/NFRLocator

	medicine-related documents, including PROMISE, CCHIT, and others.
Slankas security[6]	10,963 & 5050 of them are security-relevant and labeled with 6 security objectives, compiled from several medicine-related documents.
An ontology-based learning approach for automatically classifying security requirements[7]	Keywords, phrases and tf-idf tables for security mechanisms and threats from An ontology-based learning approach for automatically classifying security requirements
SRS Concordia corpus[8]	9 NFR classes, 3064 manually labeled sentences from 6 sources

Table 2.1: Summary of security requirements datasets.

2.3.1. PROMISE NFR Dataset

NFR dataset [25] is widely used in different researches of applying NLP techniques for requirement engineering and was part of the RE'17 Data Challenge. Overall, the dataset consists of 12 different classes, including security, with 625 requirements. This dataset was applied in different settings as multiclass and binary perspectives. Potentially, this data set can be considered as a typical benchmark for classification and extraction tasks in NLP4RE.

2.3.2. SecReq Dataset

This dataset [5] was assembled by Knauss et al. to research automation of security requirements detection as a collection of three industrial security requirements documents: Common Electronic Purse (ePurse), Customer Premises Network (CPN), and Global Platform Specification (GPS). Each document is divided into individual requirements, labeled either as security-related or not. The composition of the SecReq dataset allows a straightforward binary classification task.

2.3.3. PURE Dataset

PURE (PUblic REquirements dataset) [34] is a dataset designed as a collection of 79 publicly available natural language requirements documents from the Web. Overall, it includes about 34 000 sentences, which might be applied for NLP tasks that are typical in requirements engineering like model synthesis, document structure assessment, etc. Additionally, it can be adapted as a benchmark dataset to other tasks, such as requirements categorization, ambiguity detection, etc.

[6] https://github.com/iambackend/Riaz-Dataset
[7] https://www.dropbox.com/s/ruagh6bcxu8u8dh/jss.zip
[8] https://www.semanticsoftware.info/system/files/NFRClassifier.tar.gz

2.3.4. OSS projects: Apache Axis2/Java (Axis2), Drools, and GeoServer

This dataset [23] originated from an effort to build a classifier to distinguish security requirements from non-security ones in open-source projects. It was manually created by analysing three projects: Apache Axis2/Java (Axis2), Drools, and GeoServer. It was stated that those projects were chosen because they had existed and had been maintained for a long time, all resources including requirements and source code are available and, finally, those projects are web-based meaning that security concerns are very important.

2.3.5. 15 requirements specifications developed as term projects by MS students at DePaul University

This collection of requirements [35] was used in a set of studies associated with requirements classification. This dataset belongs to the PROMISE corpus, which consists of 15 documents, developed as term projects by MSc students at DePaul University. Overall, these specifications contain 326 non-functional requirements and 358 functional requirements. One important challenge with this dataset concerns statements that could be classified into more than one type of category.

2.3.6. Customer requirements for an integrated engineering toolset (IET) under development at Siemens Logistics and Automation plant

This document consists [35] of free form text pages that were used for the proof-of-concept for systems for identifying requirements. The IET document contained 137 pages, 2,250 paragraphs, and 30,374 words. In the practical example to classify the NFRs in the document, it was parsed and preprocessed to obtain 2,064 sentences. Some sentences are not grammatically complete, but they included bullet points and text extracted from tables. Some sentences correspond to actual requirements in the text and others to a less structured narrative.

2.3.7. SRS Concordia corpus

During PROMISE dataset [36] examination, Rashwan et al. noticed that this dataset has several problems. For example, it covers only a part of all requirements artifact types, while sentences may have multiple or no labels. Thus, they created their own corpus from 6 document sources, resulting in 3064 manually labeled sentences.

2.4. Formal Patterns for Requirements Specification

The present work analyzes the state of the art in the reusable formalization of security requirements with the focus on security verification. Verification always assumes the presence of some specification against which the verification is performed.

We thus just say “security requirement patterns” when we actually mean both specification and verification. By “verification” we mean both static and dynamic (testing, run time monitoring) methods. Security requirement patterns are expected to contain reusable specification mechanisms for applying

them to arbitrary software systems. They are also expected to contain mechanisms for their own verification against candidate designs and implementations of the specified system.

This section overviews pattern-based approaches to verifying the security of software systems. Our ultimate goal is to create a data set that will map early software specifications, e.g. security requirements, which are far from being formal, to the security verification patterns identified in the literature. Having such a data set will let us develop advanced machine learning (ML) based tools to automatically propose formal security verification patterns from early requirements in natural language. The formal patterns will then be instantiated into actual security verification attempts. What form these attempts will take is an open question that we will answer at a later stage of the VeriDevOps project.

Our intention was not to perform an exhaustive literature search – it rather was to identify conceptually different approaches to pattern-based security verification. In the future, it will be possible to perform a deeper literature search in each of the identified conceptual clusters, but for now, we just want to see the directions for further exploration. Basically, what we present here is a breadth-first search with the intent to cast a wide net on the literature.

2.4.1. Security requirements and verification patterns in literature

We have identified the following publications that have to do with security requirement patterns. Konrad et al. [37] formalize security patterns in the style of the "Gang of Four" [38] and then enrich them with LTL constraints in the spirit of specification patterns by Dwyer et al. [39]. A candidate system for verification is specified in class, sequence, and state UML diagrams. The practitioner of the approach then chooses a pattern and instantiates it based on the candidate system's model. The resulting instantiation of the chosen pattern is then submitted to the SPIN model checker for verifying its conformance to temporal properties. The full list of security patterns specified in this way may be found in [40].

Several other works build on top of the results by Konrad [37] and Wasserman [40]. The work by Yoshioka et al. [41] surveys approaches to security patterns. More specifically, it identifies key activities in security patterns extraction and application processes, then assessing different approaches based on how they contribute to the respective activities. The survey only mentions two approaches to security patterns that enable precise checking of security properties: the already reviewed one by Konrad et al. [37], and another one by Jurjens et al. [42]. The work in [42] proposes encoding security properties in UMLsec [43], an extension of UML. The resulting UMLsec specification is then submitted to AutoFocus – a CASE tool that is capable of generating test sequences. These test sequences need to be instantiated in the context of a candidate system to actually test the said system.

Work [44] by Ouchani and Debbabi surveys approaches to specification, verification, and quantification of security in model-based systems. Within the present document, we are mostly interested in pattern-based approaches that support verification and have tool support. Among the works assessed in [44], the following ones meet our criteria: [37] (discussed earlier), [40] (discussed earlier), [45] [46] [47] [48] [49] [50] and [51]. All these works share common roots, in the sense that they do model checking of UML models in one or another way. The work by Ouchani et al. [51],

however, has brought to our attention CAPEC – Common Attack Pattern Enumeration and Classification. The official CAPEC website[9] gives the following introduction:

"The Common Attack Pattern Enumeration and Classification effort provides a publicly available catalogue of common attack patterns that helps users understand how adversaries exploit weaknesses in applications and other cyber-enabled capabilities. "Attack Patterns" are descriptions of the common attributes and approaches employed by adversaries to exploit known weaknesses in cyber-enabled capabilities. Attack patterns define the challenges that an adversary may face and how they go about solving it. They derive from the concept of design patterns applied in a destructive rather than constructive context and are generated from in-depth analysis of specific real-world exploit examples. Each *attack pattern captures knowledge about how specific parts of an attack are designed and executed and gives guidance on ways to mitigate the attack's effectiveness. Attack patterns help those developing applications or administrating cyber-enabled capabilities to better understand the specific elements of an attack and how to stop them from succeeding."*

Ouchani et al. model both the target systems and the CAPEC patterns as SysML activity diagrams. They then compute the probabilities of a given system being vulnerable to each CAPEC pattern by submitting the resulting activity diagrams to the PRISM [52] probabilistic model checker.

Many other results build on top of the CAPEC repository. Kotenko and Doynikova [53] present a technique and an accompanying tool for generating random attack sequences and security events based on CAPEC. The technique relies on network configurations as the main input. Kanakogi et al. [54]propose a natural language processing-based method to automatically trace the related CAPEC patterns from CVE entries. This work is especially relevant to our project because it formalizes natural language too; our project is different in that it will formalize natural language that is even less formal than CVE descriptions. Kanakogi et al. experimented with TF-IDF [54] and Doc2Vec [55] and concluded that TF-IDF was more accurate for the task of tracing the related CAPEC patterns from CVE entries. Yuan et al. [56] reported a then-ongoing effort of developing a tool that would take on input a STRIDE [57] threat model and automatically [58] propose CAPEC attack patterns sorted by relevance to the input threat model. Kaiya et al. [59] proposed a method using which a requirements analyst can automatically acquire the candidates of attacks against a functional requirement. We found the said method especially interesting because it was the first CAPEC-based method to work with requirements as inputs. Also, based on our personal experience, real work on security requirements starts when functional requirements already exist in some form. Sometimes only functional requirements are specified, with security concerns being postponed till the later stages of the software process.

Williams [60] [61] builds his work on top of the results by Kaiya et al. [59]. He proposes an ontology-based collaborative recommender system for security requirements elicitation. The proposed system takes use cases on input and identifies relevant CAPEC patterns. It then connects the identified CAPEC patterns with the system-specific vocabulary to construct abuse cases [62] for the system in question.

[9] https://capec.mitre.org/about/index.html

We had a discussion in the process of working on the present document, after which we decided to give special attention to the security testing of APIs because of their widespread use. Sudhodanan et al.[63] proposed a methodology in which security experts can create attack patterns from known attacks. Then they describe a security testing framework that leverages attack patterns to automatically generate test cases for security testing of multi-party web applications. The approach relies on proxy-based web security scanners to record client-server interactions and automatically detect the applicability of attack patterns to the recorded interactions. Sudhodanan et al. implemented their approach on top of OWASP ZAP[10] proxy-based web security scanner and uncovered twenty-one previously unknown vulnerabilities in well-known multi-party web applications. Bozic et al. [64] capture attack patterns as UML state diagrams. They use ACTS [65], a combinatorial testing tool for the generation of test input strings based on the attack patterns and domain-specific parameters and constraints. Bozic and Wotawa [64] encode security testing patterns as UML state diagrams and submit them to a tool that automatically generates test cases from these diagrams. They implemented a prototype tool on top of the WebScarab[11] proxy-based web security scanning framework (one of the most mature tools in the field). The two above works authored by Bozic originated from a project called DIAMONDS (ITEA2 project on Development and Industrial Application of Multi-Domain Security Testing Technologies). Several more works found in the literature happened to originate from that project.

Smith and Williams [66] developed six black-box security test patterns – for (1) input validation vulnerability tests, (2) force exposure tests, (3) malicious file tests, (4) malicious use of security functions tests, (5) dangerous URL tests, (6) audit tests. They also developed a tool called Security Test Pattern Instantiator (STPI; we could not find the tool online) to help software testers instantiate security test patterns based on functional requirements. Finally, Smith and Williams conducted a user case study in which 21 graduate and 26 undergraduate students used the STPI tool to develop a black-box security test plan. The study revealed that the novices' decisions were very close to the "golden standard" developed by a committee of experts.

A comprehensive review of security testing techniques by Felderer et al. [67] let us identify another conceptual cluster of pattern-based approaches – risk-based approaches. The risk-based approaches use numerical evaluations of risks' severity to define the required level of test coverage when generating test cases for the associated risks. That is to say, the higher the risk's severity is, the more coverage will be required from the test cases generated from that risk. Großmann et al. [68] described a tool-based iterative approach that combines the CORAS [69] approach to model-driven risk analysis with automated security testing based on patterns such as CAPEC. In every iteration of the approach, the risk analysis results are fed into the process of identifying relevant security test patterns and then instantiating these patterns into actual test cases. The testing results are then fed back into the risk analysis process, and so forth. Botella et al. [69] (originating from the DIAMONDS project) propose an approach that starts with risk analysis that relies on an approach similar to CORAS [69] and concludes with automated security testing of the target system. The test generation process relies on

[10] https://www.zaproxy.org/

[11] https://github.com/OWASP/OWASP-WebScarab

CertifyIt [70], existing model-based testing (MBT) software. CertifyIt takes on input behavioral models of the system expressed as UML statecharts and risk-based test purposes – formalizations of vulnerability test patterns. The work in [71] relies on an existing catalog of security test patterns by Vouffo Feudjio [72]. Each of these patterns consists of a test procedure template surrounded by other parameters defining when and how to apply the pattern. Other contributions of Feudjio include test automation design patterns for reactive software systems [73] and his PhD thesis [72].

The work of Feudjio was supported by the DIAMONDS project. Other results achieved within this project include a work by Wotawa and Bozic [74]where they propose a planning-based approach to security testing. The authors represent security testing as a planning problem with the goal of breaking the application under test. Speaking of other pattern-based approaches, the survey only mentions one approach called VERA [75]. The website[12] by the survey, which was said to host a collection of patterns, is not available. We then decided to find contributions that cite the work of Feudjio [73]. Herzner et al. [76] present an approach for capturing best practices in integrating risk and safety analysis and testing by means of analysis and testing (A&T) patterns. Each A&T pattern encompasses a dedicated workflow starting from analysis down to test steps for achieving certain qualities of the system. The authors mention a repository of 17 ready-to-use patterns[13]. The work by Herzner et al. is cited by a work by Dghaym et al. [77] to originate from the same project. Dghaym et al. present a concept of verification and validation (V&V) patterns. V&V patterns are a unified format for capturing common verification and validation approaches, such as behavior-driven development, model checking, the Event-B method, and others. When applicable, each pattern contains a formalization of the corresponding verification and validation procedure. The language of the formalization depends on the approach assumed by the said pattern. The work is accompanied by a repository of ready-to-use V&V patterns[14]. Also, a whole PhD thesis focusing on pattern-driven and model-based vulnerability testing of web applications [78] by Alexandre Vernotte.

2.4.2. Analysis of findings

In the present section, we do a retrospective analysis of the found literature to identify the dimensions along which it would be possible to locate the different approaches to security requirement patterns. Since we are expecting to use the identified patterns for automated formalization and verification of security requirements, our analysis focuses on the verification aspect. Different approaches to specifying requirement patterns lead to different approaches to verifying the resulting requirements instantiated from the patterns.

In general, approaches to software security assurance can be mapped into the following categories. Static approaches, which work at the implementation level, without running the system under analysis and dynamic approaches (e.g., testing), which focus on generating and running security tests with properly generated test inputs.

[12] http://www.spacios.eu/index.php/spacios-tool/

[13] https://vvpatterns.ait.ac.at/the-at-patterns/

[14] https://vvpatterns.ait.ac.at/the-vv-patterns/

Static approaches include two categories. Model-checking based approaches, which take as input a formal model of the system and model-check the desired properties against that model. These approaches require an architectural model of the system on input and do not require that the development phase has already started. Code analysis-based approaches, which work with candidate program implementations of the system. Such approaches require that the development phase has already started.

Dynamic approaches include model-based testing and vulnerability testing. Model-based testing focuses on generating tests and their inputs based on design and architectural models of the system. These approaches may facilitate test-driven development of the system if the development phase has not started yet. Vulnerability testing performs attacks on running applications.

Vulnerability testing includes fuzzing-based testing and risk-based testing. Fuzzing-based testing directly attacks the application trying to break it using known attack patterns and malicious inputs. Risk-based testing, which attacks the application based on identified security risks that are specific to the given problem domain and behavioural description.

Another way to divide the different approaches to identification of security patterns is direct and indirect. Direct identification finds security patterns that are explicitly present in the input document in some form. Indirect identification finds security patterns that are implied by the input document but are not explicitly present in it.

2.4.3. Ready-to-use catalogs of patterns

We identified several publicly available collections of security verification patterns:

- Temporal patterns [40] in the style of Dwyer et al. [39] ; patterns are enumerated in the article itself.
- CAPEC repository of attack patterns[15].
- Analysis and testing patterns[16] by Herzner et al. [76].
- Validation and verification patterns[17] by Dghaym et al. [77].

We want to encode these patterns in the form of seamless object-oriented requirements (SOORs) [79] that makes it possible to reuse them (in the object-oriented sense of “reuse”) across many projects and components. SOORs are generic classes[18] that contain parameterized unit tests (PUTs) [80] whose arguments’ types are defined by the generic parameters of the enclosing class. They also contain methods for automatically generating their natural language representations, with the possibility to program automated generation of any other mark-up-based representation as necessary.

[15] https://capec.mitre.org/about/index.html

[16] https://vvpatterns.ait.ac.at/the-at-patterns/

[17] https://vvpatterns.ait.ac.at/the-vv-patterns/

[18] https://github.com/anaumchev/requirements templates

2.4.4. Integrating security requirement patterns into the SDLC

The current section gives an example of integrating the approaches from section 2 into the phases of the software development life cycle. We do not consider testing as an isolated activity because in practice it goes hand-in-hand with implementation due to the high incrementality of modern software development.

At the analysis stage, we may practice identification of expressly security-related requirement statements and inference of potentially relevant security risks based on functional descriptions of the system [59] [66] [76].

During the design phase, when abstract models of the system become available, numerous approaches apply to such models. The model checking approach verifies the models against security-related temporal properties [37] [40]. Some approaches identify security risks potentially applicable to the models [68] [81]. Other approaches propose attack vectors relevant to the identified security risks; such attack vectors may come from a CAPEC-like repository [68] [81].

When compilable and runnable prototypes of the software start emerging, the more direct security verification approaches that rely on testing become applicable. In particular, it becomes possible to record and analyze actual interactions with the software to identify interaction patterns that are known to be prone to certain security risks and CAPEC-like attack vectors [63] [82]. A more straightforward approach consists of scanning the running software to find its entry points and submit malicious payloads to the identified entry points (lots of scanners are available for this task). Combining the two approaches is also possible.

Software maintenance is probably the most expensive and troublesome part of the software development life cycle. When the software is deployed to a production environment, it is necessary to detect attempts to break it. The only way to do so seems to be in identifying common attack patterns in the log of the software. Repositories similar to CAPEC may be especially handy for this purpose. Also, it is necessary to monitor user forums and handle external incident reports. This is where an NLP-based approach, similar to [54], may help.

2.5. Gap Analysis

At the moment, a sphere of NLP4RE is presented by many approaches, models, and ideas of extracting features from requirements in a textual form. Generally, each approach was designed to solve a specific problem using a corresponding dataset depending on the type of the requirement context. Our main focus is designing and adapting successful NLP practices for security requirements. As it was stated the main problem of applying NLP methods for that type of requirement is that they are vulnerable to the generalization - ability to apply the same NLP model to different datasets. This heterogeneity of requirements is usually associated with different domains, contexts and an absence of a generally accepted definition of security requirements as well. Proposed ideas usually involved widespread ML models with various feature extractions and text representations. Still, for the context of security requirements, there is a lack of studies and practices of thorough data engineering for constructing a joint dataset that would generalize a context of security requirements. This might be achieved by a wide collaboration of experts to design a general dataset for security requirements. Also, the

perspective of transfer learning methods such as BERT and deep learning applied for security requirements is still not completely investigated.

Our initial conclusion is that the future directions should be to progress by combining and analysis of those specific datasets of security requirements while including those of the VeriDevOps partners. In addition, it appears reasonable to thoroughly analyze transfer learning and deep learning methods applied to those curated joint datasets.

Our ultimate goal is to seamlessly proceed from natural language requirements to their verification. Among the pattern catalogues, only the temporal patterns [37] [40] are immediately reusable for verification. Our most important task is then to take the remaining catalogues and convert them into a form that is closer to verification. Seamless object-oriented requirements (SOORs) [79] may become such a form. SOORs are reusable classes that already contain verification procedures in the form of parameterized unit tests [80] and are applicable to arbitrary software components through genericity and inheritance. The next task would be to train a model the world map natural language statements to verifiable patterns from the resulting catalogue. For doing that, we will need a not-yet-existing data set. Constructing such a data set is a Security Requirement Patterns 9 ambitious task that will require considerable amounts of manual analysis of requirement documents and security verification procedures resulting from these documents. Automatically instantiating the SOOR style patterns into actual verification attempts is another big task. It represents a separate research direction on its own. In our opinion, it makes much sense to perform this task in parallel with the task of mapping natural language requirements to the patterns because the two tasks are loosely connected. The task of instantiating the patterns into verification attempts requires, however, that the SOOR representation of the patterns is already in place. On the other hand, the task of mapping natural language requirements to requirement patterns does not depend on the actual form of these patterns and can be started as soon as possible.

3. Reactive Protection at Operations

Managing security risk during operation is one of the main objectives of dynamic risk management that targets evolving ICT systems and evolving risk landscape. In this sense, security at runtime has become an important part of today's software development projects where different pieces of code from several providers can be used and integrated into evolving ICT environments [83]. In particular, in ICS the cyber-physical infrastructures must guarantee the protection of the traditional (physical) elements, such as sensors, controllers and actuators; as well as the novel (cyber) capabilities, in terms of computing and communication protection[84]. The topic has been attracting increasing attention since the Stuxnet incident when the successful cyber-physical sabotage of a uranium enrichment plant in Iran took place [85]. More recently, a similar incident struck a steel mill in Germany [85], [86].

But procuring cybersecurity in ICS becomes more critical over time because of the imperfection of the existing protection tools, and the increasing presence of vulnerabilities. For example, in 2018 the number of vulnerabilities identified in different ICS components and published on the US ICS-CERT's[19] website were 415, 93 vulnerabilities more than in 2017. Even more, compared with the previous year's data, the proportion of vulnerabilities that have a high or critical severity score has grown. More than half of the vulnerabilities identified in ICS systems (284, compared with 194 in the previous year) were assigned CVSS v.3.0[20] base scores of 7 or higher, corresponding to a high or critical level of risk.

Improving software security requires that software administrators acquire relevant knowledge and skills to secure software deployment and operation such that they can resist attacks and handle security errors appropriately [87]. They also need to be supported by tools to ensure dynamic risk management techniques [88], [89], for the automatic detection of vulnerabilities and their mitigation at runtime. In the matter of standardization, a comprehensive overview of risk management techniques is given in the ISO/IEC 31010 standard [90]. Furthermore, the ISA/IEC 62443 offers a system risk-oriented approach to solving the tasks of providing the security of industrial control systems (ICS) at all stages of the life cycle.

In the litterature, there are many different risk management techniques well adequate for the development phase. Most of these techniques are applicable within a wide range of domains, including cybersecurity of modern industrial systems which is addressed in VeriDevOps. Brainstorming is a common risk management technique and is a means for collecting a broad set of ideas from different experts and sources. Examples of such techniques are Delphi technique [91] and HAZOP (Hazard and Operability) [92], SWOT [93] and FMEA/FMECA [94]. Other techniques are based on risk modelling - a structured way of representing an incident with its causes and consequences by means of graphs, trees or block diagrams [95]. Examples of well-known languages for modelling risks and their causes are Markov models, Bayesian networks [96], Fault Tree Analysis (FTA), Event Tree Analysis (ETA), Cause-consequence diagrams [97], and CORAS [98]. These techniques need to be adapted to the operational phase and be used by SecOps teams to tackle security risks at runtime.

[19] https://us-cert.cisa.gov/ics

[20] https://www.first.org/cvss/calculator/3.0

SecOps (Security + Operations)[21] is a movement created to facilitate collaboration between IT security and operations teams and integrate the technology and processes they use to keep systems and data secure — all in an effort to reduce risk and improve business agility. As illustrated in Figure 3.1, the WP3 of the VeriDevOps project will incorporate security practices into operational environments. Our objective is to integrate operation and security activities to perform dynamic risk management in running environments. Typically, the problem of risk management is addressed in several steps (NIST[22]): identification, protection, detection, response and recovery, that have been approached from different perspectives in industry and academia. In the following sections of this chapter, we present a summary of the recent innovations in each of the risk management stages.

First, in Section 3.1 we will present an overview of the recent methods that have been taken to scan the vulnerabilities of the system. The following sections will address the detection phase, which is commonly divided into Signature-based techniques, depicted in Section 3.2, and Anomaly-based algorithms. In particular, Genetic algorithms are targeted in Section 3.3, and Machine Learning network-based algorithms in Section 3.4. In Section 3.5 we introduce the Root-Cause Analysis technique, which is not traditionally used in the risk management process but is essential in the context of VeriDevOps. Finally, in Section 3.6, we will expose the reaction and remediation methods currently used , followed by Section 3.7 where we depict our contribution to the subject .

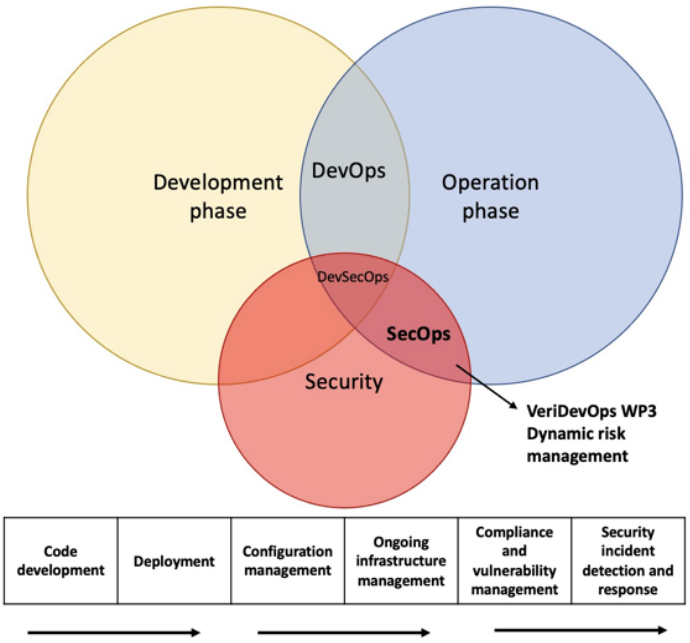

Figure 3.1 Context of the dynamic risk management process done in the VeriDevOps WP3

[21] https://saltproject.io/the-power-of-secops-redefining-core-security-capabilities/

[22] Guide for Conducting Risk Assessments.
https://www.nist.gov/publications/guide-conducting-risk-assessments

3.1. Vulnerability scanning

Several studies highlight that the number of cyberattacks on Operational Technology (OT) is increasing. OT refers to the set of technologies, software and hardware that organizations use for managing physical industrial equipment, assets, processes and events. Industrial Control Systems (ICS) are widely used in these OT environments to monitor and control industrial processes, including manufacturing, transport and pharmaceutical sectors, and critical infrastructures, such as electricity, water treatment plants, and oil and gas refineries [99]

Historically, ICS ran on proprietary hardware and/or software that were physically isolated from external connections; today the situation is totally different. ICSs have adopted Information Technology (IT) solutions, such as commercial off-the-shelf (COTS) components, standard operating systems, and remote connectivity as well as cloud solutions. This evolution , together with the use of unsecure industrial protocols, such as, DNP3, OPC, MODBUS, increases the likelihood of security vulnerabilities and incidents[23] [100] [101]. As a result, ICSs are subjected to the same type of vulnerabilities as any other system: including, buffer overflows, hardcoded credentials, authentication bypass, cross-site scripting, missing authentication, and vulnerabilities in hardware chips [102], among others.

According to Kaspersky [103], more than 150 industrial control systems-related vulnerabilities are discovered every year since 2012. Moreover, they showed that the most vulnerable ICS components were Human Machine Interface (HMI), electric devices, and SCADA systems. Vendors produced patches and new firmware for 85% of the published vulnerabilities. Most of the unpatched vulnerabilities, that is 74%, were considered being of high-level risk, but they were not addressed properly by vendors exposing a significant risk to the owners of the systems. Security challenges are emerging also in cloud environments, including a variety of issues such as misconfiguration issues and vulnerabilities in hardware chips[24].

Fast and effective detection and patching of known vulnerabilities are essential to effectively preventing cyberattacks. This issue was widely recognized in 2017 when the WannaCry[25] ransomware exploited vulnerabilities for which Microsoft had published a patch two months before. Unfortunately, some organizations had not installed it, while the patch was also not available for legacy Windows versions (i.e. Windows XP or Windows Server 2003) when the attack was produced. This highlighted the importance of monitoring vulnerabilities even though no attack cases are around.

Despite the efforts being made to improve security in products and services, the current complexity of software makes it highly unlikely to be produced without vulnerabilities. Therefore, even if there is meticulous attention in the software development process, there will always be a vulnerability that is not discovered or not remediated along the timeline, which may eventually lead to catastrophic losses. As stated by Bruce Scheiner, an internationally renowned security technologist,

[23] https://docs.broadcom.com/doc/istr-24-2019-en
[24] https://docs.broadcom.com/doc/istr-24-2019-en
[25] https://es.wikipedia.org/wiki/WannaCry

"Security is a process, not a product", and security countermeasures need to be applied until removing a product or service from use [104].

A vulnerability is a behavior or set of conditions present in a system, product, component, or service that violates an implicit or explicit security policy. Attackers exploit vulnerabilities to compromise confidentiality, integrity, availability, operation, or some other security property [105]. A vulnerability that has a Common Vulnerability and Exposure (CVE) number is a publicly known vulnerability. The CVE list represents MITRE's attempt to systematically name security vulnerabilities[26]. This is recognized as the standard for naming vulnerabilities and many vulnerability databases support CVE. The goal of Vulnerability Management (VM) is the iterative practice of identifying, classifying, remediating, and mitigating vulnerabilities. This research work aims at identifying publicly known vulnerabilities resulting from hardware, software, and configuration.

Vulnerability scanning or detection can be done manually, automatically, or by combining both techniques. One can also differentiate active and passive scans. In active scanning, the aim is to search for all existing vulnerabilities, those that are known and unknown. These scans are usually intrusive because they try to exploit all services, and are performed, for example, using fuzzing techniques. Some examples of vulnerability active analyzers are Achilles[27] and Nessus[28]. Unlike active scans, passive scans look for vulnerabilities without actively interfering with a system. This usually involves having a piece of equipment on a network switch to listen to the network traffic. These scans are focused on analyzing active hosts, ports and services used as well as connections. The advantage of active scanning is that it provides more information about assets than does passive monitoring. This additional information may include open ports, installed software, security configuration settings and known malware.

Some active scanning is based on agents, which are pieces of software running in a product. These services usually consume very little CPU resources and are used to monitor network configuration, and the state of software, among others. Agents provide deeper visibility and system efficiency than agentless scanning, while reducing overhead on the network. However very often agents cannot be used in industrial systems because they may conflict with services running on the target, an agent may not be available for the underlying operating system running in the industrial component, agents may not have sufficient privileges in local security policy to audit every configuration item, or because agents themselves can become a target of an attacker.

To overcome this problem, sometimes the solution may be to have an active scanning in a test environment, typically simulating a real system. This approach is obviously very costly both economically and logistically if every control system is replicated since every modification made to the actual infrastructure must be also replicated in the test laboratory (i.e. applications, software versions, existing communications).

Besides, web application security scanners are computer programs that assess web applications with penetration testing techniques. The benefit of automated web application penetration testing is

[26] https://cve.mitre.org/
[27] https://invent.ge/39qIPlH
[28] https://www.tenable.com/products/nessus

significant. A web application security scanner not only reduces the time, cost, and resources required for web application penetration testing but also eliminates reliance on test engineers or human knowledge. Nevertheless, web application security scanners are generally possessing weaknesses of low test coverage, and the scanners are generating inaccurate test results [106].

As a result, we can agree that existing tools for detecting vulnerabilities seem to be not suitable for industrial systems for the reasons argued above and for the following reasons: 1) they do not support industrial protocols to detect the presence of a target, 2) they need to have a vulnerability scanner connected to the equipment under vulnerability searching, 3) scanning tools are for IT environments and cannot be applied to OT environments, and 4) most of them are not able to discover software and hardware composition of a target. In general, they detect available network services, scan them for vulnerabilities and sometimes they can infer an operating system or programs used, but these results are not very reliable.

We need to design, implement and validate automated means to overcome the mentioned problems. The proposed Treat Oracle Engine (THOE) tool in VeriDevOps to be designed, implemented, and validated in this project will overcome these problems. THOE will be based on defined Common Platform Enumeration (CPE) identifiers for each hardware and software[29]. Unlike common vulnerability scanners that address target identification with limited results related to software and hardware, THOE will use a CPE-based asset inventory for hardware and software during system development to ensure that not only the operating system and installed applications are considered, but also open-source libraries, packages, use of cryptographic chips, and so on. This CPE-based asset inventory will be done during software development to ensure all assets are considered, and this will be used by THOE for automatic vulnerability searching. THOE will use the National Vulnerability Database (NVD) by the NIST that is fed by the CVE list published by MITRE and it is recognized as a standard[30]. This detection will be automated and identified vulnerabilities will be sorted based on their severity, so that later risk management, issue fixing, and reporting can be performed. The SCAP[31] standard will be used for the purpose of automating security compliance, configuration, and vulnerabilities evaluation.

3.2. Signature-based intrusion detection

Signature-based intrusion detection systems (SIDS) monitor events in a system and compare them with patterns and signatures of security policies to be respected, or attacks and vulnerabilities to be avoided that exist in a database. Therefore, if one of these previously recorded behaviors is detected, an alert is triggered. In general, this approach has the advantage of presenting a very low false-positive alarm rate compared to others [107]. Nevertheless, as the attacks are previously defined in a specific manner, a simple modification of the attack could make it undetectable by the engine. Furthermore, other problems still remain such as a choice of techniques and algorithms to accurately and efficiently detect malicious behaviors and intrusions [108], and how to identify attacks that span across several packets [109].

[29] https://cpe.mitre.org/
[30] https://nvd.nist.gov/
[31] https://csrc.nist.gov/projects/security-content-automation-protocol

Misuse detection is one of the main branches of the signature-based intrusion detection paradigm. To manage the inherent uncertainty and ambiguity of the intrusion detection data, fuzzy logic has been widely integrated into the misuse detection process and various fuzzy algorithms and classifiers are included in the security literature for accurately recognizing the attacks. Masdari et al.[108] provide a survey and taxonomy of the fuzzy misuse SIDS approaches designed to improve the security of computer systems. They include the use of techniques such as fuzzy clustering methods, fuzzy classifiers, fuzzy feature extraction, etc. to detect intrusions and malicious behaviors in presence of uncertain data.

In the domain of distributed systems, most existing approaches are not suitable for distributed and collaborative intrusion detection due to the geographical distribution. For this sake, Uddin et al. [107] proposes an optimized pattern recognition algorithm and IDS architecture for distributed heterogeneous IoT Environments. The authors propose an Intrusion Detection System (IDS) methodology and design architecture for Internet of Things that makes the use of this search algorithm to thwart various security breaches. Numerical results are presented from tests conducted with the aid of NSL KDD cup dataset showing the efficacy of the IDS.

Wang et al.[110] propose a privacy-preserving framework for signature-based intrusion detection, using Rabin fingerprint algorithm, in a network based on fog devices. The authors performed experiments in both simulated and real network environments, and they demonstrated that the proposed approach can help protect the privacy of data, greatly reduce the workload of the central server on the cloud side, and achieve less detection delay as compared with similar approaches like PPIDS.

Regarding industrial control systems, Richey's thesis [111] aims to leverage the static topology of ICS networks and those programs that define them to enhance the IDS's knowledge of the environment in which it is deployed. The author describes a method for automatically generating rules and signatures to detect possible intrusions, by parsing PLC ladder logic to extract address register information, data types and usage. Moreover, a Ladder Logic Parser program was created to test the proposed method, showing that it is not only applicable to a controlled test environment, but can also create a significant number of Snort rules that define abnormal behavior using real-world ladder files. Using a smaller test case ladder file, the functionality of this method was proven accurate and a sampling of the larger real-world files were found to be thorough and valid.

Currently, there are numerous SIDS the market, some examples include McAfee NSP[32], Palo Alto Networks[33], SolarWinds SEM[34], etc. In addition, there are several known open source solutions, some of which we briefly overview in the following subsections.

[32] https://www.mcafee.com/
[33] https://www.paloaltonetworks.com/
[34] https://www.solarwinds.com/security-event-manager

3.2.1. Snort[35]

Due to its popularity, Snort rule formats have been adopted in many cases as a standard for many other Intrusion Detection Systems (IDS), and also these are built to be compatible with it. Among its advantages, we can mention that it has a large library of pre-built detection rules, it allows deep visibility into network traffic, and it is usable on all operating systems.

3.2.2. Suricata

Suricata[36] was engineered by the Open Information Security Foundation (OISF), financed by the US Department of Homeland Security. It was designed as a competitor of Snort, and it is compatible with its file formats, rules, etc. It includes features not available in Snort such as the use of multi-threading to achieve a higher efficacy, the capability to perform network traffic analysis at the application level (which enables detection of malicious content spread over multiple packets), or the detection of common network protocols even though they are not operating over standard ports assigned to them.

3.2.3. Zeek

Zeek[37] or Bro[38] is a platform-independent framework that comprises multi-level modular architecture underlying network layer in the ISO-OSI seven-layer model. Zeek conducts activities such as multi-layer analysis, policy imposition, behavioral controlling, and policy-oriented detection. There are many significant advantages of using Bro IDS: It is able to manage capturing data from Gbps networks and can perform with great efficacy in high-speed environments, it allows the implementation of sophisticated and complex signatures, and it is highly customizable, though difficult to deploy.

3.3. Runtime monitoring and detection using Search-based testing

Search-Based Security Testing (SBST) is an approach that applies a search-based algorithm such as Genetic Algorithms to prevent systems from different attacks and provide security for the system. The search-based approach uses a fitness function that measures the quality of an individual within a population, guiding the evolution of individuals towards the desired solution. SBST is just not only an effective technique but also extremely flexible, allowing different test approaches by simply changing the fitness function. The technique also minimizes manual interactions and therefore reduces the overall cost to scale the industrial testing problems. Below mentioned are papers covering various search-based security testing.

Research at Queen's University, Kingston, Canada by Gong et al. [112] demonstrated the implementation of a genetic algorithm-based approach to detect network intrusions. This paper covered the implementation of an architecture based on genetic algorithms to tackle attacks. The technique used in this method excels others since it has the capacity to automatically cope with

[35] https://www.snort.org/
[36] https://suricata-ids.org/
[37] https://zeek.org
[38] http://www.bro-ids.org/

real-world problems, such as the change in the type of intrusions. The designed system can update the rules to the system as soon as new attacks are known. Genetic Algorithm (GA) is preferred rather than Genetic Programming (GP) in the classification of the network data. A fitness function with high efficiency and flexibility is used to detect the network intrusions or separate them based on their types. The system is implemented using Java language and a third-party software called ECJ (i.e A Java-based Evolutionary Computation Research System [113]). As for the purpose of evaluating the system, DARPA [114] data from MIT Lincoln Laboratory is used for the training and testing data.

Improving network application security using a stress testing approach was reported by Grosso et al. [115]. This paper covers the security problems in the network applications caused by Buffer overflow, which leads to its exploitation and unauthorized access to the program. The technique used is evolutionary testing which makes use of static analysis and program slicing to mitigate the buffer overflow threats. It also used GA (Genetic Algorithm) and three fitness functions which cover Vulnerable coverage fitness, Nesting fitness and Buffer boundary fitness. The fitness function was assessed using Random search to generate random data to analyze the effectiveness of the function.

Grosso et al. [116] extended the previous approach to detect buffer overflow via automatic test input data generation. The presented method is an upgrade to the previous approach of improving security in the network application using stress testing. The main change in this new technique is that it does not require manual intervention to initialize and tune GA and fitness function. The approach is composed of genetic algorithms, linear programming, evolutionary testing, and static and dynamic information to detect buffer overflows. The proposed technique helps to increase the efficiency in large industrial systems.

Dozier et al. [117] presented a paper on vulnerability analysis of immunity-based Intrusion Detection Systems(IDSs) using GENERTIA red teams (GRTs) [118] . The presented method compares 12 evolutionary hackers based on particle swarm optimization PSO) as vulnerability analyzers for IDSs. The research concludes GA followed by PSOs: ccSW4 and SW0+ as the best performers as they are capable of finding different types of vulnerabilities in entirely different IP addresses' search space, help to analyze the vulnerabilities before deploying and to heal themselves when leaks are discovered.

Campos et al. [119] investigated entropy-based test generation for improved fault localization. The presented prototype is ENTBUG which is an extended search-based test generation of EVOSUITE [120] to use fitness functions on its genetic algorithm and apply it to seven real faults. The presented method optimizes the quality of the ranking reports by focusing on producing detailed diagnoses rather than just finding faults.

Galeotti et al. [121] demonstrated a tool on how to improve search-based test suite generation with Dynamic Symbolic Execution (DSE). The presented method also extends EVOSUITE's [120] genetic algorithm to integrate DSE to increase the coverage. An adaptive approach that combines GA and DSE, and later classifying the suitability of the problem in hand using SBST (Search-based Software Testing) is used.

A safety-critical system requires high security. Tracey et al. [122] briefed a paper to demonstrate a search-based automated test-data generation framework for safety-critical systems. The technique is based on genetic algorithms or directed search approaches. To be more precise, it uses a fitness function that provides test-data suitability for specified criteria. Using a directed search approach, the

fitness function has access to all of the information that is available at runtime which is how SUT (Software Under Test) takes the input test-data at some test constraints and ultimately outputs the values. This framework targets industrial testing problems by allowing complete automation of error exploitation and reducing costs with traditional testing techniques.

In a paper by Afzal et al. [123] a systematic review of search-based testing for non-functional system properties, a thorough examination of existing work into non-functional search-based software testing (NFSBST) has been done. The review is based on a set of 35 articles published between 1996-2007, dealing with different fitness functions to guide the search engine for safety, usability, quality of service and security.

3.4. ML/AI based anomaly detection

Anomaly-based intrusion detection systems (AIDS) work by comparing the actual comportment of the system with a previously-established "normal" model of the behavior of the system. Any substantial deviance between the observed behavior and the model is considered as an anomaly, which can be translated as an intrusion or attack into the system. AIDS has drawn interest from a lot of scholars due to its capacity to overcome the limitation of the Signature-based intrusion detection systems (SIDS) [109].

Normally, in AIDS, the normal model of the behavior of a computer system is created using machine learning, statistical-based or knowledge-based methods. For the purposes of the VeriDevOps project, we will consider only the machine learning method.

The applications of machine learning techniques in the design of intrusion detection systems (IDS) have remained a trend in the last few years [124]. Therefore, there have been numerous anomaly-based IDS prototypes that implement these techniques.

In this section, we collect surveys that summarize the recent ML anomaly-based IDSs trends in diverse contexts. In such a manner, we list works on industrial systems, the direct target of the VeriDevOps project; in IoT environments, whose distributed nature can be expected also in industry systems; and in SDN-based networks, an increasingly common approach on networks outside and inside the industry. Moreover, we emphasize deep learning techniques as a prospective method for the next generation of intrusion detection techniques due to their capability of automatically finding correlations in data [124], [125]. Finally, we summarize and classify the mentioned surveys in Table 3.4.

Liu et al. [126] proposed an IDS taxonomy that takes data sources, i.e., logs, packets, flow, and sessions, as the main benchmark to present the numerous machine learning techniques (especially deep learning algorithms) used in the design of anomaly-based IDSs. Moreover, they listed the capabilities of the studied data source as follows: **(i) Logs** include exhaustive semantic information, which is appropriate for detecting SQL injection, R2L (Remote-to-Local) and U2R (User-to-Root) attacks; **(ii) Packets** deliver communication contents, which are suitable to detect U2L and R2L attacks; **(iii) Flow** characterizes the entire network setting, which can detect DOS and Probe attack; **(iv) Sessions** reveal communication between clients and servers, can be used to detect U2L, R2L, tunnel and Trojan attacks.

[127], [128] and Da Costa et al.[127], [128] summarized various intrusion detection mechanisms using a combination of machine learning (ML) and deep learning (DL) approaches. [127], [128] address software-defined networks (SDNs) and conclude that SDN-based intrusion detection systems using ML/DL techniques have many advantages in terms of security enforcement, virtual management, and Quality of Service (QoS) supervision.

Da Costa et al. [127] reviewed the literature on machine learning techniques applied in Internet-of-Things and Intrusion Detection for computer network security, based on the purposes of the reviewed papers, the communication protocols, the application protocols, the data format, the implemented machine learning technique, and the obtained precision rate (PR). Additionally, the authors listed the datasets used in the works considered in the paper.

	ML/DL techniques	Classification criteria	Contex
	SVM, OCSVM, NB, DT, R, DBN, ANN, KNN, K-means	Type of their targeted vulnerability: integrity, availability, confidentiality, authentication and authorization	SCADA IIoT systems
Liu et al. [[126]	SVM, K-means, Fuzzy C-means, CDNN, LSTM, GAN, decision tree, Naïve Bayes, KNN, Autoencoder and XGBoost, DNN, RNN, DBSCAN, Isolate forest	Data sources: Packet, Flow, Session, Log	Non-specific environment
[127], [128]	Unsupervised artificial neural network, deep learning algorithms, self-organizing map and learning vector quantization	Detection target: anomaly, intrusion, DDoS, Flooding	SDN based networks
Da Costa et al. [127]	K-means, SVM, MCLPDR, OS-ELM, Random Forest, LS-SVM, Optimum-path forest, Bat algorithm, Firefly Algorithm, Optimum-Path Forest Clustering, SA-IDSs, Naive Bayes, J48, ADAM, - CSF-KNN,OCSVM, NNs	Addressed communication and application protocols (e.g. TPC/IP and CoAP), and precision rate	IoT environments

	Deep learning: AE, RBM, DBN, RNN, CNN	Training datasets, Accuracy, Processing component, efficiency	Non-specific environment
	Neural network, Kalman Filter, One Class SVM, One-Class Classification SVDD	Type of data source (the network flows (Industrial-NIDS); the memory of the equipment (Industrial-HIDS); - or both (Industrial-NHIDS))	Industrial Control Systems (ICS)

Table 3.4: Summarize of surveys of ML anomaly-based IDSs, classified according to their deployment context

3.5. Root-cause analysis

Root-cause analysis (RCA) is a method used for identifying the root causes of observed incidents or problems. It is based on the idea that effective management requires more than merely "putting out fires" when problems are detected, but also finding ways to prevent them. RCA is applied in a wide range of domains including IT systems, telecommunications, industry, transport accident analysis, medical diagnosis, etc. Thanks to RCA results, remediation actions/ reactions could be wisely taken to prevent or mitigate the damage of the recurrence of the problem.

Although there is a vast literature on RCA, we would restrict it to techniques that can be applied to IT systems. This condensed state-of-the-art will provide a general understanding of RCA techniques, for more in-detail explanations of each of the methods, other general surveys are available [129] [130]. For specific areas, specific surveys exist for computer networks [131, pp. 3–25], software [132, pp. 165–205], industrial systems [133, pp. 859–872], smart buildings [134, pp. 13–23], (regular) buildings [135, pp. 71–85], machinery [136, pp. 636–653], swarm systems [137], automatic control systems [138, pp. 41–64] [139, pp. 127–136], automotive systems [140] [141, pp. 213–219] and aerospace systems [142, pp. 11230–11243]. For the diagnostic, different machine learning approaches have been applied, namely Artificial Neural Networks, Fuzzy Set Theory, Rule Based Systems and Bayesian Networks. However, the actual analysis approaches remain a slow and manual (or partly manual) process often carried out by the operators' experts who are the main actors analyzing and correlating multiple data sources, such as network traces, alerts, logs, key performance indicators. The diagnostic delay is inevitable and the accuracy remains questionable because it depends hardly on the learning data set as well as the selection of network indicators.

In summary, there are three key ingredients to be able to perform a successful RCA:

- Domain Knowledge, the laws that govern the system (e.g., the laws of mechanics for a mechanical device). Except mathematical abstractions, anything would have to satisfy the laws of physics, but it is understood that domain knowledge refers to the rules that govern the

system at an abstraction level that makes these rules practically applicable (e.g., no one will use quantum mechanics equations to troubleshoot a printer, even though the printer must obey those rules, as an easier and more tractable set of rules is enough to model the device for that particular purpose). Domain Knowledge can be given as an input to the method (this encompasses model-based methods as they are sometimes called in the literature) or automatically derived using Machine Learning (ML) techniques. The latter techniques are sometimes called model-free, although automatically generated model methods would describe more precisely what they actually do. Note that the application of ML can yield models that include both the Domain Knowledge plus the System Knowledge. For instance, an assumption that is sometimes done when diagnosing a black-box system (or a system that people do not understand, i.e., they do not have a model for it), is that similar symptom patterns are related to similar root-causes. In this case, a potential way to proceed is to use a nearest neighbours classifier that maps the symptoms to an N-dimensional space and compares them to previously seen symptoms for which we know the root causes. In this case, the model consists of the labelled points, and the N-dimensional space both implicitly contains the information about the domain knowledge and the system knowledge.

- System Knowledge, the elements that comprise the system and their relationships (e.g., the different components of the printer and their relations, like relative position, elements in contact, groups of elements related to the same function, etc.). System Knowledge can be perfect or flawed. Assuming one or the other distinguishes between methods that can update (or suggest fixes) to current system knowledge and others that might simply give wrong answers in the presence of incorrect or missing facts. Methods that can handle inexact system knowledge are necessarily more complex and not as popular in research as the ones assuming perfect information, although in many fields this is a frequent source of problems. For instance, in large IT industrial environments, it is frequently assumed by administrators that the corresponding Configuration Management Database (CMDB) describing the infrastructure can contain a large number of errors or omissions.
- Observations, observable data coming from the system (e.g., printed pages from the printer, information from its internal monitoring sensors, history of previous failures, etc.). Observations in IT come usually from monitoring tools specialised in gathering them and providing unified access to the data. Collected data can be of a wide variety of types, from very structured, like a specific known set of metrics, to almost unstructured such as logs. These observations can be noisy for many factors, including low accuracy of sensors, bad conditions for readings, or simply because of the own particularities of the monitoring system setup: for instance, unsynchronized clocks, different granularities of collection, variable lags, etc., can alter the temporal order of event observation. This is a relevant issue because the temporal sequence is often used to restrict feasible casualties (i.e., a consequence cannot precede its cause), and causality discovery is essential for an informed root cause analysis. As a result, some techniques include the possibility to look for correlations between events both in the future and in the past [143, pp. 36–43].

These three elements allow making inferences on what could be happening in the unobserved parts of the system, predict future observations and potentially solve the RCA problem, among other things.

In addressing three aforementioned elements, the current and future trend are related to the following issues:

- Learning and diagnostic approaches: Machine learning approaches will be continuously integrated to automate and speed up the process, especially the Deep learning algorithms.
- "Good" datasets for the learning phase: The redundancy in the datasets is desirable to test different machine learning algorithms to determine a suitable or a combined one for a specific use case.
- Selection of the most relevant network indicators: The release of new tools, applications and even hardware devices are providing more relevant indicators which deserve to be taken into account. In the case there are too many indicators and the decision of the experts might be hardly difficult, PCA (Principal Component Analysis) or other feature selection approaches can be taken into consideration.

As an example, Figure 3.5.1 demonstrates an existing high-level architecture of RCA relying on machine learning algorithms to identify the most probable cause(s) of detected anomalies based on the knowledge of similar observed ones.

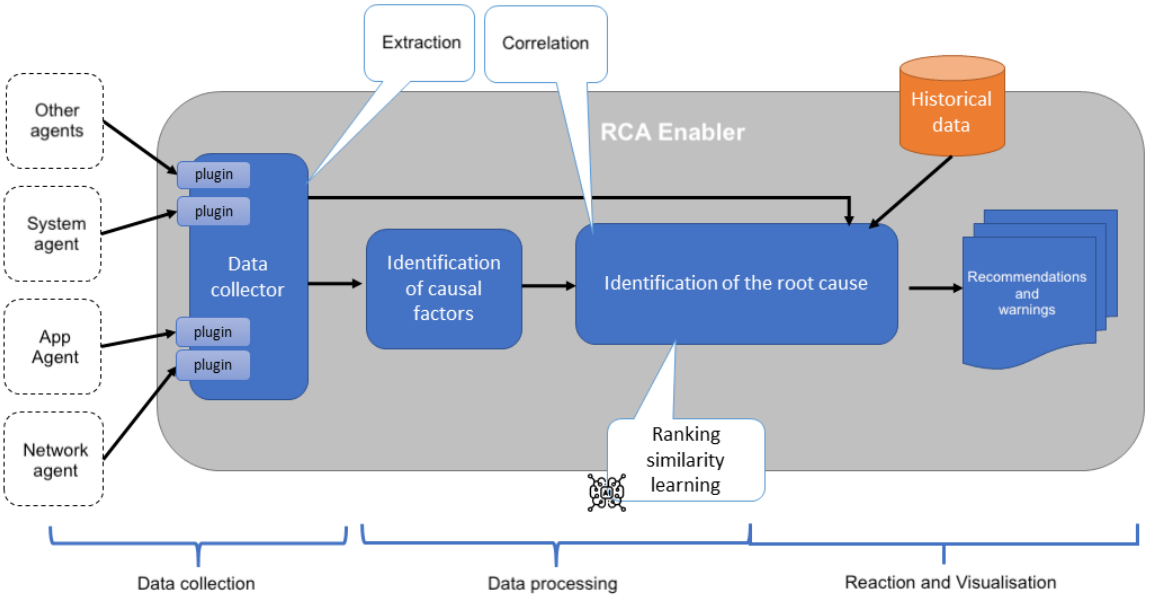

Figure 3.5.1 RCA high-level architecture example.

The data collector allows gathering information from different sources (e.g., network, application, system, hardware) by relying on dedicated monitoring agents. It has a plugin architecture that enhances its extension to new formats of data. The parsing of such data allows extracting numerous attributes values that can be relevant for the identification of the origin of detected incidents. The selection of the most relevant attributes is performed based on several machine learning algorithms in order to increase the accuracy of the analysis and reduce the data dimensions as well as the computations needed.

Historical data is a set of information used for learning purposes. They are labelled events collected over time to describe the origin cause of several incidents as well as the relative attributes values. This data is constructed by two means:

- Active learning: By actively performing different tests including the injection of known failures and attacks, the collected data can be easily labelled since we deal with a controlled system.
- Passive learning: Once an incident is detected without knowing its origin, thanks to the aid of the system experts, classical RCA is performed by debugging different logs and correlating various events to determine the corresponding root causes. The result of this task can be stored in the database with its relevant attributes values.

Based on these two inputs (collected new data and historical data), the idea is to determine when the system reaches a known undesirable state with a known cause. This involves using the concept of Similarity Learning and, more specifically, "Ranking Similarity Learning" [144]. That means the tool calculates the "similarity" of a new state with the known ones and then presents a list of the most similar states with the relative order of similarity. The final goal is to recognise the root origin of an incident and based on the set of known mitigation strategies based on the experience reflected in the historical data. In this way, the tool can recommend to the operator to perform the relevant countermeasures.

3.6. Reaction – remediation

As said before, a new generation of Industrial Control Systems (ICS) is providing advanced connectivity features, enabling new automation applications, services, and business models in the Industry 4.0 era. Nevertheless, due to the extended attack surface and an increasing number of cyber-attacks against industrial equipment, security concerns arise. Hence, these systems should provide enough protection and resilience against cyber-attacks throughout their entire lifespan, which, in the case of industrial systems, may last several decades. To face these threats, firstly continuous monitoring is needed, and secondly, the application of remediation and countermeasures effective at mitigating them.

Continuous monitoring techniques are already envisioned in VeriDevOps by means of vulnerability scanning, intrusion detection, runtime monitoring, and anomaly detection, described from section 3.1 to section 3.4. Furthermore, a root cause analysis (section 3.5) is proposed to determine any underlying causes of security weaknesses, vulnerabilities, or anomalies. The next natural step is the application of remediation techniques effective at mitigating potential detected threats.

Organizations should identify, review, and assess potential security incidents [145]. This process is of paramount importance so that an organization can determine which potential threat has a bigger impact on the compromise or attempt to compromise. For example, once a vulnerability or security flaw is discovered, for example, a buffer overflow in a software component, the developer should evaluate whether the library is used in the system, and, for instance, which functions use them, are they safety critical?, that is, the potential impact of the discovered vulnerability should be checked., also looking at possible remediation techniques.

Once this management process is performed, organizations should apply remediations. SP800-82 special publication published by NIST provides recommended security countermeasures to mitigate associated risks in industrial control systems [146], [147]. This publication also provides

recommendations, best practices, common security threats and vulnerabilities for industrial control systems, including, for example, SCADA (Supervisory Control And Data Acquisition) systems, DCS (Distributed Control System), and other control system configurations, such as PLCs (Programmable Logic Controllers). Unfortunately, this publication represents guidance about security controls to be implemented in industrial controls, such as account management, separation of duties, unsuccessful login attempts, and so on, that is, controls to be applied or implemented during the development or maintenance phases but not to be applied during operation.

Industrial standards IEC 63069 [148] and IEC 62443-4-1 [148], [149] address incidence response and they set these actions as possible remediation measures:

- Software patching
- Controlled power off
- Deactivation of certain functions or parts of the system
- System concept, architecture or defence in depth strategy change
- Implementation of organisational procedures and/or measures
- Use of compensating mechanisms, such as new security functions/capabilities.

In general, there is a lack of risk mitigation and remediation plans in organizations that develop or run industrial control systems. Companies are usually unprepared to respond accordingly in the event of a potential incident in IT environments and even more in OT ones. Accurate and timely information may help incident handlers reduce the number of infections, or address vulnerabilities before they are exploited, but this information has not always improved the situation for incident response teams [150].

Furthermore, current heterogeneous environments lack a body of knowledge for shared and common mitigations to be applied in ICS. MITRE ATT&CK for Industrial Control Systems (ICS) is a community-sourced framework for identifying malicious threat behaviors, including tactics and techniques of adversaries, in ICSs, and for each technique, general mitigations are given, but these are oriented again to network defense capabilities and configuration, mainly [151]. This MITRE ATT&CK framework can be a reference for an incident response if it is possible to associate potential anomalies or vulnerabilities with attack techniques. In VeriDevOps, needed actions for resilience will be analysed so that they can bring decision-support.

3.7. Gap Analysis

VeriDevOps will innovate by fully investigating diversity, prevention and tolerance combined with different risk management techniques that can be applied to industrial systems during their operation. Most of these techniques are applicable within a wide range of domains within cybersecurity.

The project addresses updating the risk management at run-time based on data that is continuously collected through monitoring, during the operation phase of systems. This is not addressed, in particular, by traditional current risk management techniques and notations. Moreover, in each stage of the risk management process, the project aims to propose new approaches beyond the state-of-art, and to test them in actual industrial systems, in order to generate reliable new techniques in this field.

In addition to the IDS, another fundamental piece of the puzzle will be the publicly-known vulnerability scanner that will use CPE-based asset inventory for hardware and software during system development to ensure that not only the operating system and installed applications are considered, but also open-source libraries, packages, use of cryptographic chips, and so on. This scanning will be run on a daily basis, every minute or every few minutes depending on the choice, and depending on the industrial component features with an agent or without.

VeriDevOps will also allow defining intelligent defense strategies by applying Root Cause Analysis algorithms based on probabilistic decision trees to perform an accurate diagnostic of a detected security incident and automatically deploy the relevant security mechanism (according to the ratio impact over cost) according to the application running environment. Several challenges have been described in this section. We can briefly summarize some of those here as follows:

- Continuous vulnerability scanning of known vulnerabilities in software and hardware industrial systems
- Signature-based intrusion detection for industrial systems
- Runtime network monitoring and detection using Genetic algorithms and machine learning based anomaly detection
- Root-cause analysis based on machine learning algorithms and big data sto identify most probable causes of detected anomalies
- Listing remediation techniques to mitigate threats in industrial systems

The combination of these tools will lead to bridge the gap of security in industrial systems during their operation. These tools will be validated in the two use cases in VeriDevOps to show effectiveness. Moreover, the matching between WP3 innovations and each step of the risk management process is depicted in Figure 3.7.1.

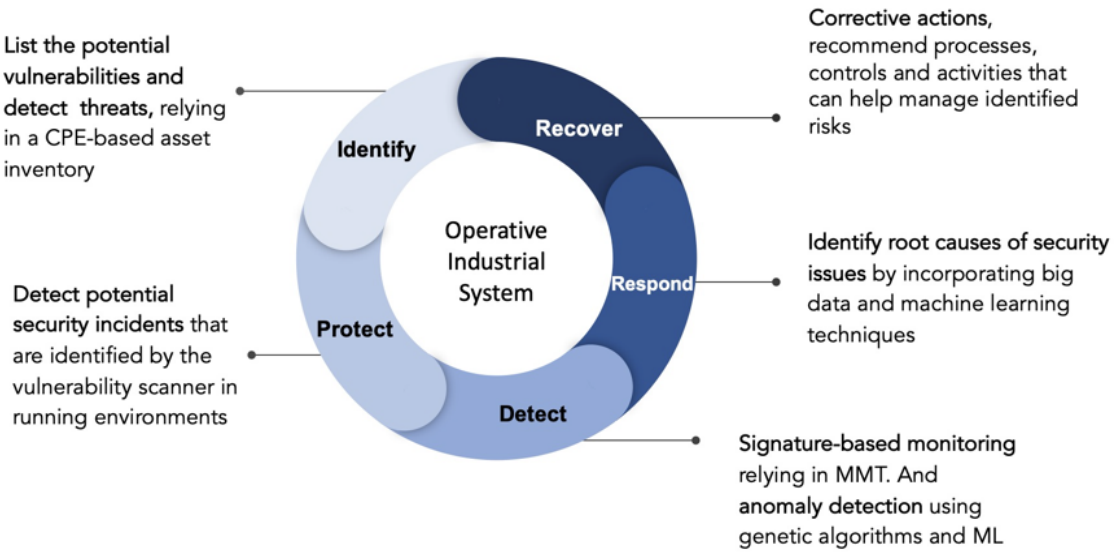

Figure 3.7.1 Summary of VeriDevOps WP3 contributions in each step of risk management process

4. Prevention at Development

Security is not a unique feature of software; it is an important aspect of the software that should be implemented at every major phase of the software development life cycle [152]. It is reported that there are more chances that a security issue would occur due to a bug in a typical part of the system (e.g., the interface to the database) than a given security feature like Secure Sockets Layer (SSL) for encrypting the communications [62]. There are several processes used to integrate the security at various stages of SDLC. Microsoft has published a software security process, called Security Development Lifecycle (SDL)[39] that introduces security and privacy early and throughout all phases of the development process. Another example is the Software Assurance Maturity Model (SAMM)[40] framework from Open Web Application Security Project (OWASP), comprising five phases: governance, design, implementation, verification, and operations as shown in Figure 4.1. We describe techniques used to enforce security aspects at development relevant for this project. Section 4.1 overviews techniques for formal analysis and verification for security requirements. In Section 4.2, we introduce security testing and discuss security testing methods (model-based security testing, penetration testing, mutation and fuzz testing approaches, and search-based security testing) in Section 4.3. We investigate the localization and debugging approaches in Section 4.4. Section 4.5 discusses the integration of security in the continuous delivery process.

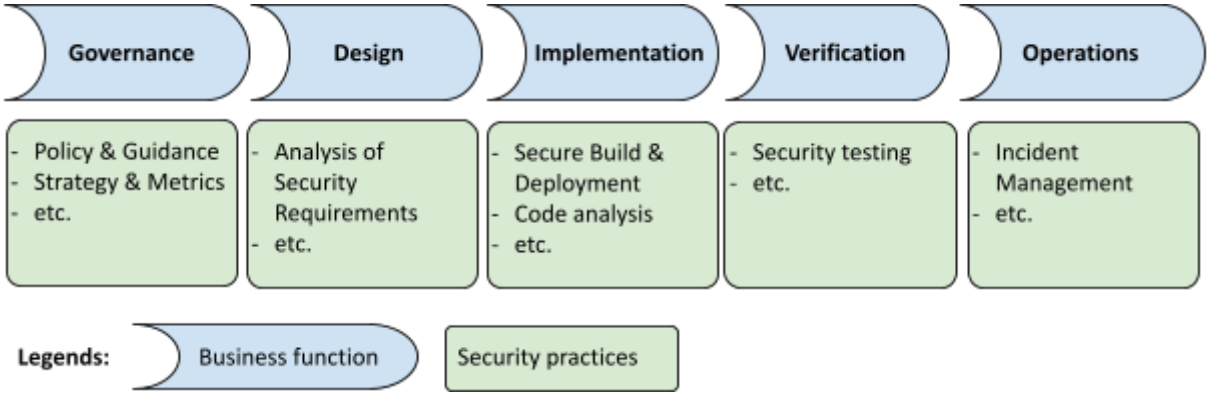

Figure 4.1 OWASP SAMM model overview

4.1. Formal analysis and verification for security

Chong et al. [153] presented a report on the results of the NSF workshop on formal methods for security held at the University of Maryland, College Park in November 2015. They discussed some benefits of using formal methods in software systems. Formal methods allow one to rule out various

[39] https://www.microsoft.com/en-us/securityengineering/sdl
[40] https://owaspsamm.org/model/

security attacks. They not only provide precise specifications and models of the system but also include approaches for reasoning about and mathematically proving certain properties of the system like e.g., liveliness. However, there is no single set of right security guarantees for software systems. The desired security requirements can vary from one application to another and may depend on the specifics of the application and system. They also highlighted several open research challenges in this area including (1) how to provide security guarantees at the system level (as opposed to individual component-level guarantees), (2) what abstractions to use to enable the use of formal methods for security guarantees, (3) how to standardize existing and new tools to allow compatibility of specifications, tools, and proofs, (4) how to support the use of formal methods through different stages of software development life cycle, and (5) how to enable industrial-scale use of formal methods for providing security guarantees. These challenges must be addressed to realize the benefits of formal methods for security.

Security issues can arise at the boundaries between components, even though individual components may be “secure” e.g., [154]. Security issues at these boundaries can be exacerbated when different individuals or organizations have responsibility for the various components. In addition to ensuring security guarantees when individually-secure components are composed together, “whole-system guarantees'' (also “cross-layer security” and sometimes “end-to-end guarantees”) refer to security guarantees that hold across abstraction boundaries. The incompatibility of formal method tools can hamper the integration of individually-secure components. Currently, there are a few standards for low-level tools such as Boolean Satisfiability (SAT) or Satisfiability Modulo Theories (SMT) solvers [155], [156], but there are no common standards for formal specifications and proofs, and existing tools can vary greatly in their representation of specifications and proofs. Thus, if different tools are used to formally validate the security of components, it may require significant effort to combine these formalizations. There have been several promising success stories (and in-progress stories) for whole-system security, including Ironclad [157], the HACMS DARPA program [157], and verification of a radiation therapy system [158].

Subburaj and Urban [159] presented a security framework for formally specifying security requirements of a multi-agent system in the Descartes - Agent formal specification language. The framework can be used to model authentication and authorization requirements for software agents. However, it lacks support for automated design, code, and test generation. Giorgini et al. [160] presented a formal framework called Secure Tropos, which allows modeling and analyzing functional and security requirements of agent-oriented software. They also presented a formal reasoning tool based on Datalog (a declarative logic programming language), which allows verifying correctness and consistency of functional and security requirements.

Rouland et al. [160], [161] presented a formal methods approach for specifying and verifying security requirements using first-order logic as a technology-independent formalism and Alloy as a tool-supported language for modeling and analysis of component-based systems. To validate their approach, they formalized and verified Confidentiality, Integrity and Availability (CIA) security properties of a component-based system by defining and verifying an appropriate security policy for each security property. Zhioua et al. [162] [163] presented a framework for integrating formal specification and verification of security guidelines by combining model checking analysis with information flow analysis. Their framework is based on an extension of Labeled Transition Systems

(LTS). It allows to formalize security guidelines for a program written in a natural language and to verify that the program adheres to the formalized security guidelines.

Yajima et al. [164] presented a support tool for formal verification of security specifications with ISO/IEC 15408 called FORVEST. The tool helps and guides human verifiers by providing information on Z notation, linear temporal logic, theorem proving, model checking, and ISO/IEC 15408. It can be used to check whether the system specifications satisfy the security criteria given in the ISO/IEC 15408 standard. It also provides a web-based environment for using model checking and theorem proving tools.

Vistbakka et al. [164], [165] presented a safety-security co-engineering approach for industrial control systems. They demonstrated how formal modeling can help analyze the effects of certain security solutions on the safety of the system and vice versa. Troubitsyna and Vistbakka [166] and Vistbakka and Troubitsyna [166], [167] presented an approach for deriving, formalizing, and verifying safety and security requirements for control systems using the Event-B framework on the Rodin platform. They integrated safety and security analysis and proposed to derive safety and security requirements by combining data flow analysis with a safety analysis technique called Hazard and Operability Analysis (HAZOP). The approach can be used to identify deviations from normal system behavior caused by accidental failures and security attacks and to reason about the impact of different security threats on the safety of the system. It also allows us to specify and verify system behaviour in the presence of accidental faults and security attacks and analyse interdependencies between safety and security requirements.

Dwyer et al. [39] have identified patterns in formal specifications for finite-state verification, and these patterns often take complex forms. The seamless object-oriented requirements approach [39], [168] makes it possible to codify such patterns in the form of template classes encapsulating the complicated verification logic. This approach makes the patterns reusable across systems and produces immediately verifiable requirements. Applying the object-oriented approach to requirements not only simplifies verification but also enables round-trip requirements engineering and traceability: the template classes are capable of automatically producing structured natural language representations of the source informal requirements.

Related to real-time systems, Kang et al. [169] and Farrell et al. [170] have specifically focused on formal verification of safety and security-related constraints using stochastic timed automata, PROMELA and Boogie. Wardell et al. [171] also proposed an approach for identifying security vulnerabilities of industrial control systems using PROMELA models.

4.2. Security Testing

Software security testing is the process of validating software requirements related to security properties such as confidentiality, integrity, availability, authentication, and authorization [172]. The purpose of security testing is to find weaknesses in software implementation, configuration, or deployment [173]. There are several literature reviews published on security testing; however, in this section, we discuss those that are relevant for this project.

Mahendra and Ahmad [174] presented a literature review on software security testing published between 2000 and 2015. They classified the existing works on software security testing into four categories namely frameworks, techniques, methodologies, and reviews. They studied 10 frameworks, 4 techniques, 5 methodologies, and 4 reviews. Their results show that the existing approaches for software security testing are implemented at various phases of the software development lifecycle. However, there is a lack of approaches that can be applied at the design phase.

Security testing has also been applied in the context of regression testing to evaluate that updates to the system do not introduce new security bugs. In this context, Felderer and Fourneret [172] presented a systematic classification of security regression testing approaches. They found and classified 17 approaches. Their classification is based on several criteria including abstraction level, security issue, regression testing techniques, tool support, and evaluation criteria (evaluated system, the maturity of the system, and evaluation measures). They also identified several potential directions for future research, such as the need for model-based techniques to increase abstraction and scalability, and improved tool support for automated generation and execution of security tests.

There are two main types of security testing: security functional testing and security vulnerability (or penetration) testing [175], [176]. In *security functional testing* we validate whether the specified security requirements are implemented correctly. Typically such requirements are expressed with a negative connotation, such as “something should not be allowed to happen” [177]. In security vulnerability testing we apply *penetration testing* to uncover the potential system vulnerabilities as an attacker. *Vulnerability* refers to the flaws in system design or implementation, for example, buffer overflow or SQL injection. It can be used to attack or compromise the security of a system.

Several authors have mentioned the need to complement security testing with other testing techniques. For instance, Zeng et al. [178] mention the importance of combining static code analysis and dynamic testing for the security of a power system. For dynamic testing, the procedures for operating the power system are simulated to generate test instructions and results. Tudela et al. [179] also analyze the merits of combining static, dynamic, and interactive analysis security testing tools.

Ali [180] proposes a four-stage process to ensure security in distributed control systems. The four stages are (1) risk management, (2) security management, (3) trust and reputation management, and (4) testing and evaluation. In the last stage of testing and evaluation, the author proposes two types of testing strategies: penetration testing and fuzzing.

4.3. Security Testing Techniques

In this section, we discuss different security testing techniques related to the project. Felderer et al. [152] presented an overview of security testing techniques. They discussed basic concepts and recent developments in different types of security testing techniques and classified the existing techniques into four different types: (1) model-based security testing (based on requirements and design models created during the analysis and design phase), (2) code-based testing and static analysis (based on source and byte code created during development), (3) penetration testing and dynamic analysis (based on running systems in a test or production environment), and (4) security regression testing (performed during maintenance).

4.3.1. Model-based Security Testing

Model-Based Testing (MBT) is a black-box testing technique that generates tests from abstract behavioral models [181]. It allows one to automate or semi-automate the entire testing process. Moreover, the testing activities can be left-shifted; and the SUT can be tested early in the software development life cycle. In Model-based Security Testing (MBST), we test the security requirements of the system under test using models [182]. MBST is a relatively new research field, where many approaches were published in recent years. In this section, we provide a comprehensive overview of the state of the art in MBST.

Lunkeit and Schieferdecker [183] described an approach that focuses on the use of the artefacts of previous phases of security engineering for security testing. This work is proposing a model-based definition of the security problem that supports finding an optimal test architecture and defining safety and security-oriented tests that go far beyond functional testing.

Felderer et al. [182] classify MBST based on the two dimensions: automated test generation and consideration of risk values. The automated test generation dimension describes how much of the system and formal models capture the security requirements. The risk dimension specifies whether the risk values are integrated into the model. Fully automated test generation is only possible with formal and complete models, which are typically not available. The results show that only a few model-based security testing approaches [184],[185] integrate complete or partial automated test generation and risk values**.**

Schieferdecker et al. [176] surveyed the area of MBST from specification and documentation of security test objectives to security test cases and test suites generation. They studied different types of model-based security testing techniques including security functional testing, model-based fuzzing, risk, and threat-oriented testing, and the usage of security test patterns. The authors classify MBST approaches based on three types of input models for security test generation: (1) architectural and functional models describe the system requirements regarding the general behaviour and setup of a software-based system. (2) The threat, fault, and risk models present what can go wrong. These models enable identifying multiple risk factors, describing their relationships, and relating them to occurrence probabilities and potential impacts. (3) Weakness and vulnerability models identify the causes and consequences of system failures, weaknesses, or vulnerabilities. The publicly available databases such as the National Vulnerability Database (NVD) and the Common Vulnerabilities and Exposures (CVE), which collect known vulnerabilities, are used to build these models. Felderer et al. [186] presented a taxonomy and systematic classification of MBST approaches from 1996 to 2013. The taxonomy is based on a comprehensive analysis of 119 systematically extracted publications on model-based security testing. It complements existing classification schemes by defining filter and evidence criteria. The filter criteria comprise the specification of the model of system security, the security model of the environment, and explicit test selection criteria. The evidence criteria comprise the maturity of the evaluated system, evidence measures (i.e., the qualitative or quantitative assessment criteria to evaluate an MBST approach in a specific application), and the evidence level that specifies whether the approach is evaluated on the level of non-executable abstract or executable concrete test cases. They also discussed some promising research directions with regard to security properties, coverage criteria,

and the feasibility and return on investment of model-based security testing. Although the taxonomy provides a starting point for further research on model-based security testing, the selected papers can also be classified into more fine-grained criteria, for example with regard to the types of systems or the types of vulnerabilities.

Yang et al. [187] proposed an adaptive MBT framework, called OAuthTester, to systematically evaluate the implementations of the OAuth protocol. They combine the protocol specification and the network trace observations, followed by iterative refinement of the model with implementation specifics. The framework discovered three previously unknown vulnerabilities, all of which can result in severe consequences, including large-scale resource theft and application account hijacking.

Peroli et al. [188] introduced MobSTer, a formal and flexible MBST Framework that supports a security analyst in conducting security testing of web applications. The main idea underlying MobSTer is a hybrid approach that takes advantage of model-checking techniques combined with the knowledge provided by penetration testing guidelines and checklists. This combination enables MobSTer to exploit the automatic search for possible vulnerable "entry points" without missing important checks.

Krichen et al. [189] proposed an MBST approach to check the security of Internet-of-Things (IoT) applications in the context of smart cities. The approach consists of specifying the desired IoT application in an abstract manner using an adequate formal specification language and then deriving test-suites from this specification to find security vulnerabilities in the application under test in a systematic manner. Additional work in the context IoT has been performed in the European project ARMOUR[41] - Large-scale Experiments of IoT security testing where test generation strategies for large scale IoT security testing are discussed in [190]. The proposed approaches are model-based and target security functional testing, vulnerability testing and security robustness testing using the CertifyIt tool and TTCN3 [191].

Mahmood et al. [192] presented an MBST approach designed for cybersecurity evaluation of the Over-The-Air update system for automobiles. The approach generates and executes test cases by using attack trees as input. Integrating threat modeling in the testing provides several benefits, including clear and systematic identification of different threats.

Regarding the use of formal models for security testing, Santos et al. [193] used Communicating Sequential Processes (CSP) to create architectural models of the systems, as well as an initial set of attacks against these systems. In another formal approach, Parizi et al. [194] used smart contracts to carry out an experimental assessment of current static smart contracts security testing tools for blockchain technologies. Jürjens [195] and Wimmel and Jürjens [196] address the problem of generating test sequences from abstract system specifications in order to detect possible vulnerabilities in security-critical systems. These approaches assume that the system specification, from which tests are generated, is formally defined in the language. Their results show that the combination of security modeling and test generation approaches is still a challenge in research and is of high interest for industrial applications.

[41] https://www.armour-project.eu/

4.3.2. Penetration Testing

Penetration Testing [197], also referred to as PenTest, is an efficient security testing technique used to identify security vulnerabilities in software systems. In penetration testing, the system under test is examined using multiple attacks to find vulnerabilities and security bugs in the implementation. Penetration testing has been applied to many application domains and several commercial and open-source tools are available, and several literature reviews have been performed.

For instance, Vats et al. [198] presented a literature review on penetration testing and its applications. They presented a comparative review of 30 research papers. They also studied 13 penetration testing tools with respect to their purpose or utility, some technical specifications, platform compatibility, and release date. An overview of vulnerability assessment and penetration testing techniques is presented in [199]. The authors regard penetration testing as a complementary phase to vulnerability assessment in which a real or simulated attacker attempts to exploit vulnerabilities of the target system. In another review, Bacudio et al., [200] presented an overview of penetration testing strategies, methodology and illustrated the penetration testing process on web applications. The methodology of the test phase is described as a step-wise process comprising information gathering, vulnerability analysis, and vulnerability exploits.

Denis et al. [201] investigated the attack methodologies, defense strategies, and the tools used for penetration testing in their study. They discuss four types of penetration testing: *external* - attack against externally-visible resources such as servers, firewalls, DNS; *internal* - an attack behind the firewall by an authorized user; *blind* - external attacker with limited information and *double-blind* - only a few persons in the organization are aware of the attack. The study discusses penetration testing for attacks such as Man-in-the-Middle attack, hacking WPA-protected Wifi, phones Bluetooth, remote PC via IP and open ports with advanced port scanner using Kali Linux, Metasploit, Ettercap, Aircrack, and Wireshark[42]. The cause of systems' vulnerabilities and mitigation techniques were presented to improve the security of systems. Shebli et al. [202] presented a study about the process, factors, components, and tools used for penetration testing. The methods of penetration testing, types of penetration, the phases of conducting the penetration testing, tools used, and the role of Information security Management System (ISMS), professional ethical and technical competency for performing the test were discussed in the study.

Al-Ahmad et al. [203] presented a systematic literature review on penetration testing for mobile cloud computing applications. They found 30 relevant papers and analyzed them to conclude that there is a lack of studies on mobile cloud computing and web-based vulnerabilities and the existing approaches do not consider the mobile cloud computing application penetration testing model. In addition, the model must consider offloading parameters, multiple input types from mobile devices and networks, and target APIs and apply random testing techniques.

In recent years, artificial intelligence methods have been also applied to penetration testing. McKinnel et al. [204] presented a systematic literature review on artificial intelligence in penetration testing and vulnerability assessment**.** They found 31 relevant papers and focused mainly on papers based on empirical studies. They also identified several potential research challenges and opportunities

[42] https://www.wireshark.org/

for future research, such as scalability of the penetration testing approach, the need for real-time identification of vulnerabilities, and the need for standardized systems that can be used to assess the effectiveness of penetration testing approaches in different application domains.

Shah et al. [199] presented an overview of vulnerability assessment and penetration testing methodologies, models, standards, and popular tools used for testing. The study classified vulnerability assessment techniques as (i) Manual testing, (ii) Automated testing, (iii) Static analysis, and (iv) Fuzzing. Penetration testing is described as a phase-wise process comprising (i) Planning and preparation, (ii) Detection and Penetration, (iii) Post-Exploitation and Data Ex Filtration, (iv) Reporting and clean up. They conducted a case study on a bank system to detect technical and logical vulnerabilities as a black box testing followed by a gray box approach to simulate both external and internal attacker's viewpoints. The experiment used the open-source tools mentioned in the study and successfully detected 4 vulnerabilities.

Stefinko et al. [205] presented a study on the benefits and drawbacks of manual and automated penetration testing for security evaluation. The manual penetration using Metasploit Framework and the testing effort and cost factors were compared with the automated penetration techniques such as Social engineering toolkit (SET). The factors considered for comparison were the effort for the testing process, vulnerability/attack database management, reporting, cleanup, and training. In the same line, the research and study on automated penetration testing claimed automated testing to be more efficient and low cost compared to a traditional manual approach. Xue Qiu et al. [206] presented an automated method named AEPT (automatic executing penetration testing). Shah et al. [207] introduced an automated vulnerability assessment and penetration testing tool Net-Nirikshak 1.0 to detect vulnerabilities in the banking domain. Shah et al. [208] conducted a study on vulnerability assessment and penetration testing (VAPT) techniques and the benefits of creating cybersecurity awareness. Goel et al. [209] presented the use of VAPT as a cyber defence technology and the VAPT life cycle, techniques, and widely used tools.

Antunes et al. [210] presented a penetration testing approach that uses attack signatures and interface monitoring and implemented a prototype tool to detect SQL Injection vulnerabilities in SOAP. The tool proved to provide higher detection coverage than commercial penetration testing tools. Mainka et al. [211] introduced an automated penetration testing tool called WS-Attacker for SOAP-based Web Services. The attack types used in the study were WS-Addressing spoofing and SOAPAction Spoofing and the resistance to these vulnerabilities were evaluated proving the approach to be beneficial in identifying vulnerabilities.

Stepien et al. [212] demonstrated an approach to use TTCN-3 as a modeling language for web penetration testing. Using TTCN-3 provides the advantage of specifying and executing test suites at an abstract level thereby reducing efforts of modeling and test results analysis.

Falkenberg et al. [213] presented an automated plugin-tool for penetration testing of web service specific Denial of Service attacks. The Denial of service plugin has a DoS Attack class to provide attack-specific implementation details and an MVC DoS Extension to provide the DoS attack-specific functionality. The study was conducted on Web service frameworks Axis2 Java, Apache CXF, ASP.NET and Metro, and the results showed that except ASP.NET latest versions of other frameworks are vulnerable to DoS attacks.

Vibhandik et al. [214] presented an approach for vulnerability assessment of web applications by using a combination of W3AF and Nikto tools. The results from the study proved that combination of these tools identified vulnerabilities such as Cross-site request forgery, Sensitive information disclosure, Cross-site tracing, potential risk of attack migration thru nodes, DDoS in shared hosting, Obsolete or Insecure Software version, Security misconfiguration, Unauthorized access to sensitive information and Information leakage.

Nagpure et al. [215] presented a comparative study and analysis of penetration testing methods and tools for vulnerability assessment of web applications. The study showed that manual penetration testing proved to be more accurate in detecting vulnerabilities such as Cross-site scripting, SQL injection, Clickjacking, File upload, Browser cache weakness, Directory traversal, Authentication bypass, and Cross-site request forgery and recommended a combination of manual and automated testing approach for improved accuracy for vulnerability assessment.

Chu et al. [216] proposed the use of penetration testing to evaluate the security problems of IoT. They analysed the security issues in IoT, discussed the automation of penetration testing based on the belief-desire-intention (BDI) model and validated the work by a simulated experiment in Jason.

A systematic mapping study on penetration testing has been performed by Bertoglio and Zorzo [217] overviewed among other things the challenges of penetration testing. They identified the efficacy in the process of vulnerability assessment, creating generic models and tools that can be easily deployed to specific target scenarios and automation of the PenTest activities.

Researchers have also investigated the use of black-box scanners for detecting unknown vulnerabilities. For instance, Bau et al. [218] conducted a study on automated black-box scanners for unknown vulnerabilities to find (i) type of vulnerabilities tested by scanners, (ii) effectiveness of the scanners, (iii) to evaluate the relevance of the target vulnerabilities to vulnerabilities found in the wild. The study found vulnerabilities such as Cross-site scripting, SQL Injection, Cross-Channel Scripting, and Information Disclosure to be the most prevalent ones. Even though the vulnerability detection rate was less than 50% as per the study results, black box scanners are suggested to be beneficial in security auditing programs. In addition, Seng et al. [106] conducted a systematic review and analysis of methodologies to assess the quality of web application security scanners, that is, computer programs that assess web application security with penetration testing techniques. They studied 108 relevant papers and published conference proceedings and quantified 93 web application security scanners. The measurement parameters used for the survey were vulnerability detection rate and the number of vulnerabilities detected under the same category. They concluded that the measurement metrics to analyze a web application security scanner's attack coverage, test coverage, vulnerability detection rate, and scanning efficiency need to be elaborated based on further studies and research.

As a summary, it is generally agreed that penetration testing is a necessary part of the vulnerability assessment process, but it currently requires custom solutions for different application domains and individual systems. In change, we would like common methods and tools that can be easily customized and deployed to specific applications. In addition, there is a lack of research for methods and tools in rapidly advancing domains such as web, mobile, and cloud domains.

4.3.3. Mutation and fuzz testing approaches

One category of testing techniques that can be applied to security testing is to generate invalid inputs to the SUT in order to detect possible faults. We can split them into largely two types: functional mutation and fuzz testing.

4.3.3.1. Functional Mutation

Büchler et al. [219] and Dadeau et al. [220] presented mutation testing approaches for testing security properties in network security protocols modeled in HLPSL (High-Level Security Protocol Language). They defined and used security-specific mutation operators to generate model-level mutants. In these approaches, the original model is modified in such a way that a certain security property is violated in the implementation. The mutated models are analysed by the AVISPA tool, which produces counter-examples leading to the security flaws and allows to produce attack traces for the detected security flaws. Siavashi et al. [221] presented a model-based mutation testing approach for vulnerability assessment of web services. They modeled the web service under test and its security requirements in UPPAAL timed automata and evaluated the authentication and authorization of the web service. They used a model-based mutation testing tool called µUTA and introduced several new mutation operators to generate additional mutants. Papadakis et al. [222] presented a survey of mutation testing techniques including mutation-based security testing techniques. They analyzed several approaches for testing security protocols, regression testing of security policies, and transforming functional tests into security tests. Dadeau et al. [220] and Buchler et al. [219] have investigated the creation of security-relevant mutants at the model level and how to use these mutations for security testing.

Buchler et al. [223] presented an approach for testing web applications from a secure model. They used mutation operators generated from the set of known vulnerabilities and injected these to the secure model. To generate the models they used ASLAN++ language used for security protocols and used model checker CL-AtSe model-checker to generate attack traces. The evaluation was done on an insecure web application WebGoat maintained by OWASP and diagnosis of attack traces revealed the vulnerability attacks Role-Based Access Control (RBAC) and Cross-Site Scripting (XSS).

Henard et al. [224] presented the approach to test software product lines using mutation based on feature models. The study investigated the application of mutation operators and mutation analysis on feature models to demonstrate higher mutant detection ability of dissimilar test suites than similar ones.

4.3.3.2. Fuzz testing

Fuzz testing or Fuzzing is a dynamic testing technique used for testing the robustness and security of software systems. In this approach, the system under test is tested with invalid input data to find any security vulnerabilities that can crash or give access to the system. Felderer et al. [152] presented a survey on security testing where fuzz testing approaches are classified as random fuzzing and advanced fuzzing techniques such as mutation-based fuzzing, generation-based fuzzing.

Liang et al. [225] presented a survey on fuzzing in the area of software testing and security by reviewing 171 papers published between 1990 to 2017. They presented the fuzzing process,

classifications, tools used, and the evolution of fuzzing techniques and the growing trend of fuzzing as a popular research area.

In *mutation-based fuzzing*, the testing starts with a valid input which is then mutated in subsequent valid and invalid inputs. Its advantages are that it is easy to start with, but the inputs may diverge easily from the expected format and not be efficient in terms of input coverage. In *generation-based fuzzing*, inputs are generated in a more systematic way from the specifications of the input format such as grammars and XML schemas. Thus input coverage is easier to control and improve, but it requires that the input format is known and its specification created.

Also, the fuzzing approaches can be classified based on the "visibility" they have into the source code of the SUT. As such, black-box fuzzing generates input against the SUT without knowledge of the internal structure of the code, whereas in white-box fuzzing the test generation can be guided for instance to increase the level of coverage of the code.

Schieferdecker et al. [226] presented the model-based fuzz testing as a smarter approach for determining the tests to reveal the security risks. Unlike the random fuzzing method where the input data is totally invalid, model-based fuzzers have protocol knowledge based on the model of the system under test. Based on the protocol knowledge, model-based fuzzers generate input data containing invalid data among valid data. Model-based fuzzing outperforms the randomized approach in terms of its ability to reveal complex bugs in the system. Behavioral fuzzing focus on finding design level flaws and system vulnerabilities by submitting invalid message sequences to a system under test. Behavioral fuzzing approach can be used for the generation of test cases from models specified as finite state machines and UML sequence diagrams.

Schneider et al. [227] presented an approach to apply fuzzing operators to modify the existing UML sequence diagrams to generate invalid message sequences to test the system. UML sequence diagram provides an advantage of allowing the reuse of functional test cases for security testing using behavioral fuzzing. The case study of a banknote processing system from the DIAMONDS research project was investigated to identify the fuzzing operators and classify them for test case selection and prioritization.

Schneider et al. [228] proposed an approach to reduce test execution time and make behavioral fuzz testing more efficient by generating test cases at runtime, taking into account results from previous test executions, and focusing on message subsequences from a risk analysis carried out earlier. Online model-based behavioral fuzzing is applied to the case study of a banknote processing system from the ITEA-2 research project DIAMONDS by generating a model of the functional test cases using UML sequence diagrams. During the online test generation, fuzzing operators are applied to UML sequence diagrams to determine the message sequences to be sent to the system under test.

Duchene et al. [229] presented a black box Cross-Site Scripting fuzzer for web applications named KameleonFuzz. The fuzzer generates fuzzed inputs using an attack grammar and the control and taint flow model. The study conducted proved KameleonFuzz detected most XSS vulnerabilities, outperformed several other black box security scanners and had no false positives.

Zhang et al. [230] presented a survey on machine learning for software testing covering 138 papers on testing properties such as correctness, robustness, and security, etc. The study covered the

research related to the application of machine learning techniques for fuzzing, eg. TensorFuzz [230], [231], DLFuzz [232].

Godefroid et al. [233] demonstrated the automated generation of input grammar for fuzzing using machine learning techniques. They used a learn&fuzz algorithm where learning captures the structure of well-formed inputs and fuzzing tries to break the structure to find vulnerabilities.

Mutation-based fuzzing tools use random data mutations in the input. However, according to Wang et al. [234] such random changes are not effective if the running program uses a checksum mechanism to check the integrity of inputs. Thus the authors proposed TaintScope, a checksum-aware directed fuzzing system. TaintScope uses the taint propagation information during program execution to detect and overcome checksum-based integrity checks to generate test cases that have higher chances of detecting vulnerabilities. TaintScope works on both Linux and Windows binary executables. BuzzFuzz [235] is another taint-analysis based, white-box directed fuzzing tool that uses an instrumented application to identify input data that influences the system calls. The instrumentation of the application's source code remains a limitation of this approach. According to Munea et al. [236] most common fuzzing types are application, file format, and protocol fuzzing. In protocol fuzzing, counterfeit packets are sent to the target system or sometimes a protocol fuzzer acts as a proxy server. The authors classify network protocol fuzzing based on different factors such as intelligence level, invalid input creation, and the manner of detecting vulnerabilities, among others. Tao et al. [237] studies existing hybrid fuzzing techniques that combine symbolic execution to typical fuzz testing. Symbolic execution represents a program's inputs with symbols instead of values. The program input is then generated based on program analysis, making sure that the input executes the program in the desired location. However, symbolic execution is known to have path explosion problems, thus the hybrid fuzzing techniques aim to use low-overhead fuzzing with symbolic execution to address incomplete traversal. PANGOLIN is one example tool based on hybrid fuzzing proposed by Huang et al. [238]. They identify the uncovered branches in fuzzing, construct a summary of these uncovered branches to describe a search space of the feasible inputs with respect to the path constraints. This is then used to guide the mutation of the seeds.

Another approach to deal with the path explosion problem of symbolic execution is to use concolic execution (aka dynamic symbolic execution), i.e., some variables are kept as symbolic, but concrete values of other variables are used at runtime. The tool by Godefroid et al. [239] [240], SAGE (Scalable Automated Guided Execution) is the first tool to perform dynamic symbolic execution at the x86 binary level. Ghimis et al. [241] present a different classification of fuzz testing: white-box random fuzzing, black-box random fuzzing, and grammar-based fuzzing. In white-box random fuzzing, fuzz testing is directed to interesting areas of the program, with the goal to improve code coverage. BuzzFuzz and SAGE (discussed above) are example tools for white-box random fuzzing. In black-box random fuzzing or fuzz testing in general, sample inputs are mutated by the fuzzer with the goal to exercise overlooked corner cases.

One example black-box random fuzzing approach is proposed by Paduraru et al. [242] where a parallel implementation of a genetic algorithm in Apache Spark guides the test generation towards the uncovered area of the program. In grammar-based fuzzing, a grammar describing the input is given to the program. An example tool based on such grammar is QuickFuzz [243] that uses a grammar-based

and mutation-based fuzzing for more than a dozen common file formats. Further details on fuzz testing are available in recent review studies on the subject [225] [244]. Recently, Microsoft released publicly a REST API fuzzing tool called RESTler[43] [245] for automatically testing and finding security and reliability bugs in cloud/web services through their REST APIs. The tool generates HTTP requests based on the interface specification and in addition, it infers dependencies between different requests.

4.3.4. Search-based Security Testing at Design and Development

The paper by McMinn [246] on metaheuristic search-based software test data generation surveyed different areas of the test data generation like (1) the coverage of program structure, or white-box approaches; (2) reject the grey-box properties in the software; (3) verify non-functional properties; and (4) check the program features as per the specification.

McMinn et al. [247] research on search-based test input generation for string data types using the results of web queries uses examples of string inputs from the Internet and formulated into web queries, and the resulting URLs are splitted into tokens, which are used as seeds for search-based test data generation. The paper also presents an empirical studying using 20 Java classes from 10 open source projects.

Antoniol [248] showcased a paper on search-based software testing for software security. The paper presents the promising aspects and challenges of search-based testing methods to detect software vulnerabilities. The paper discusses recent achievements in SBSE like Buffer overflow, SQL injection, Privilege Escalation Testing, Robustness, and Penetration Testing, and Intrusion Detection Systems to mitigate the vulnerabilities. It also discusses challenges like: (1) What to break? (2) System Wide Test (3) Data Types (4) Aborting at fixed time (5) Exception raising and Singularities search (6) Good search Strategy (7) Best environment.

There are different approaches to provide security to web applications. Avancini et al. [249] investigated on security testing of web applications: a search-based approach for Cross-Site Scripting Vulnerabilities. The purpose of the paper is to implement a search-based approach to prevent exploitation from hackers which may leak crucial information like credentials and credit card numbers. The implemented technique uses Genetic Algorithm to search vulnerabilities in the input values, then collect symbolic constraints at run time and pass it to a solver to fix the security issues. The method does not automatically detect and eliminate the issues from the system but rather provides detailed information by generating test cases for developers.

Similar to the approach mentioned above, Thome et al. [250] presented a paper on search-based security testing of web applications. The paper presents BIOFUZZ, a security testing web application that uses evolutionary black-box testing to detect vulnerabilities in SQL injections. The prototype uses search-based testing to populate the inputs towards an expected target. The approach works in two phases, firstly the application tracks web applications and creates inputs that affect the interaction between the web server and database on the SQL interaction. Compared to white-box testing, BIOFUZZ requires no code access, which makes it applicable for a wider range of situations. Secondly, BIOFUZZ generates SQLI attacks to the identified inputs in the first phase.

[43] https://github.com/microsoft/restler-fuzzer

Alshahwan et al. [251] implemented an automated web application testing approach using Search Based Software Engineering. The algorithm is based on Hill Climbing using Korel's [252] Alternating Variable Method (AVM) but with constant seeding and Dynamically Mined Values (DMV). In this approach, a SWAT (Search-based Web Application Tester) tool is created in which the inputs that reached the branch but failed are kept track of and used as seeds. The tool passes constantly gathered seeds to the search space and measures the fitness for strings.

Liu et al. [253] used a different approach to that of BIOFUZZZ for detecting SQL Injection Vulnerabilities using a search-based approach. They present a novel fitness function called Similarity Matching Distance (SMD), which evaluates the similarities between the SQL statements produced by providing inputs to the system under test and a known threat SQL statement. The experiment is conducted in 19 diverse configurations of different SQL statements and attacks. It also capitalizes Differential Evolution (DE), which serves as a search engine to detect SQL injection vulnerabilities.

XML Injection is similar to SQL injection but in a different environment. Out of a few, Jan et al. [254] showcased a search-based testing approach for XML Injection Vulnerabilities in web applications. The paper proposes a novel search based approach that uses Genetic Algorithm to generate the test data to provide malicious XML messages to the web service and to detect the XML attacks. It uses black-box approach to match the output from Software Under Test and Test Objectives. The method is applied to an industrial web application with millions of users.

Apart from testing the core applications, it is also wise to implement a test case on the testing itself. For that purpose, Souza et al. [255] proposed using automated search-based technique with Hill Climbing for mutation testing. The paper proposes an efficient and cost-effective approach by removing the manual effort to strongly kill mutants, focusing on the mutations' propagation. The proposed tool is an extended version of AUSTIN [256] tool integrated with Proteum [257]. The proposed approach uses a novel fitness function that helps the process to generate test cases to reach, infect and kill the discovered mutants.

Several papers investigated security related problems, but what about Service Level Agreement? Penta et al. [258] focused on search-based testing on Service Level Agreements. The paper presents methods to analyze if Service Level Agreement (SLA) negotiation between provider and consumer is fulfilled and also to predict the Quality of Service (QoS) that can be guaranteed for the consumers. The technique used is Genetic Algorithm to generate inputs for service-oriented systems which causes SLA violations. The generated tool is applied to an audio processing workflow and a chart generation service. More precisely, the tool uses black-box approach where violated constraints are captured by the fitness and a white box approach which combines the obtained fitness with code coverage provided by Wegener et al. [259], which considers a starting point for searching for QoS violations.

The increasing use of computerized systems and software applications in our society creates a high demand for security and dependability. The traditional methods of testing involved greater human interactions and instructions and slowly being replaced by AI and different automated techniques. A huge number of software is made or modified on a daily basis, which requires a huge amount of testing to maintain their efficiency and security. AI through SBST helps to expose and prevent such software vulnerabilities and provides high security to the system. There are different algorithms inside SBST itself that can be chosen depending upon the case in hand.

4.4. Localization and debugging

Software fault localization, the act of identifying the locations of faults in a program, is widely recognized as one of the most tedious, time-consuming, and expensive activities in program debugging. Over the years, researchers have proposed a plethora of automated fault localization techniques and tools to reduce time and effort [260]–[263].

Zou et al. [264] have classified fault localization techniques into the following types:

1. Spectrum-based fault localization (SBFL): utilizing test coverage information
2. Mutation-based fault localization (MBFL): utilizing test results collected from mutating the program
3. (Dynamic) program slicing: utilizing the dynamic program dependency
4. Stack trace analysis: utilizing error messages
5. Predicate switching: utilizing test results from mutating the results of conditional expressions
6. Information retrieval-based fault localization (IR-based FL): utilizing bug report information
7. History-based fault localization: utilizing the development history

Spectrum-based fault localization techniques rank each program element (e.g., code statement) with respect to a suspiciousness score, whereas other techniques, such as dynamic program slicing, highlight a collection of elements as suspicious that cause or correlate with the failure in order to help the developer in fixing the fault.

Despite the existence of many software fault localization techniques, we found only a few studies directed to fault localization in security testing to the best of our knowledge. Concurrency-related bugs have caused many web security issues in the past. Zhenyuan Jiang [265] presented a new automated edge-labeled communication graph-based locating technique, called LUCON to identify concurrency bugs. It can find buggy memory access pairs, present buggy patterns, and build bug triggering scenarios. The buggy patterns give the essence of the bugs, and the bug triggering scenarios show how the bugs happen, which can help programmers understand the bugs.

Simos et al. [266] proposed a security testing approach for Cross-Site Scripting (XSS). The authors derive combinatorial XSS attack vectors with the help of grammar. Subsequently, the test cases are executed against a web application. Afterward, a fault-localization approach, called BEN [267], checks the structure of attack vectors that detected a vulnerability and tries to track its critical combination. BEN is a spectrum-based fault localization technique. It utilizes the results of the combinatorial test set and generates the ranking of statements in terms of their likelihood of being faulty. BEN consists of two main steps: (1) It identifies a combination that is very likely to be a failure-inducing combination. (2) It takes the failure-inducing combination identified in step 1 and then produces a ranking of statements in the source code by analyzing the spectra of the small group of tests. In this work, the authors only applied the first phase of BEN because they are not interested in the ranking of statements in the source code since they are following a black-box security testing approach.

Ji et al. [268] formulated a program error propagation-based model (PEP) to find security bugs. They proposed a spectrum-based fault-localization technique to locate faults at the code level. DeMott et al. [269] developed a distributed fuzzing technique that identifies bugs and subsequently localizes the bug's root cause. This work focuses on memory corruption errors, which usually have software

security implications. The authors localize faults using a code-based coverage algorithm, called Tarantula [270] combined with input data tainting and tracking techniques.

Lu et al. [271] proposed a fault localization framework, CRAXfault, based on single-path concolic execution, which exploits the execution path of one failed test case to automatically generate numerous test cases. These test cases are used in a fault localization technique (e.g., Tarantula [270], Ochiai [272], Crosstab [273], and DStar [274]) to rank program components, such as statements or predicates, with top-ranked components more likely to be faulty.

Firewalls are the mainstay of enterprise security and the most widely adopted technology for protecting private networks. A firewall highly depends on the quality of its firewall policy in terms of packet filtering. Firewall errors are mostly caused by faults (i.e., misconfigurations) in rules of firewall policies. Hwang et al. [275] proposed a fault localization approach to reduce the number of rules for inspection based on information collected during evaluating failed tests. The approach ranks the reduced rules to decide which rules should be inspected first.

4.5. Continuous integration and security

As we have discussed previously, there are many methods for secure software development such as SAMM and SDL [152]. Some of them involve testing or analysis of various kinds using automated tools. However, none of these approaches explicitly promote doing security testing continuously. Nevertheless, researchers have proposed some methods for adopting security testing into CI systems and DevOps workflows. Hilton et al. [276] outline the concept of a security tradeoff between increased security measures, and the ability to access and modify the CI system as needed. They report that developers encounter increased complexity, increased time costs, and new security concerns when working with Continuous Integration (CI).

Mårtensson et al. [277] identified several factors that must be taken into account when applying continuous integration to software-intensive embedded systems related to security. Security aspects can be an impediment, such as when developers are hindered from communicating freely regarding the exact content of the functions they have built. This further increases the difficulty of a common understanding of the product, which also becomes an impediment related to testing the product end-to-end.

Another problem identified in the literature is that in delivery features continuous releases, traditional security methods are not suited. In recent years the new terms DevOpsSec or SecDevOps [278] were formed which address exactly this problem. Regarding DevOpsSec, there is a lack of methodologies for handling direct security requirements and how to extract these requirements. In addition, the adoption of DevOps in industrial fields with high requirements for security, traceability and safety, such as industrial embedded systems, is scarcely reported [279].

Automated security testing is an essential component of Continuous Integration and Continuous Delivery (CICD), such as scheduling automated test sessions on overnight builds. That allows stakeholders to execute entire test suites and achieve exhaustive test coverage. On the other hand, developers also need test feedback from CI servers when pushing changes, even if not all test cases are executed. As an example, Neto et al. [280] proposed similarity-based test case selection on

integration-level tests executed on continuous integration pipelines. Related to test generation and selection, STAMP[44] is a European research project on software testing amplification for DevOps teams by reusing existing assets in order to generate more test cases and test configurations each time the application is updated.

4.6. Gap Analysis

Mahendra and Ahmad [174] reported a lack of security testing approaches that can be applied at the design phase. Felderer et al. [186] reviewed 119 publications on model-based security testing. They identified several challenges related to the kind of security properties that are tested, the use of coverage criteria, and the feasibility and return on investment (ROI) of security testing. Specifically, there is room for research that explicitly addresses vulnerabilities and directly targets system-wide security properties. In addition, there is a need to understand which variants of coverage criteria can yield effective and efficient security tests in the specific context of security testing. Felderer and Fourneret [172] highlighted the need for model-based techniques to increase abstraction and scalability and improved tool support for automated generation and execution of security tests. Jürjens [195] and Wimmel and Jürjens [196] reported that the combination of security modeling and test generation approaches is still a challenge in research and is of high interest for industrial applications. In VeriDevOps, we aim to develop model-based test generation and selection methods for security testing using security models based on formal specifications. These models can be built that can be reused in different contexts and stages of the software development life cycle, and to empirically understand the return on investment. We will develop new approaches for model-based mutation testing and fuzz testing by defining new model-level mutation operators suitable for testing security aspects. We plan to validate the novel techniques on the systems provided by ABB. Building on standards and practices such as IEC 62443 [281], VeriDevOps will develop a consolidated cybersecurity methodology incorporating the roles played in cybersecurity by people, processes, and software technologies.

McKinnel et al. [204] identified several potential research challenges and opportunities for future research, such as scalability of the penetration testing approach using artificial intelligence, the need for real-time identification of vulnerabilities, and the need for standardized systems that can be used to assess the effectiveness of penetration testing approaches in different application domains. Furthermore, Al-Ahmad et al. [203] found a lack of studies on mobile cloud computing domains such as IoT and web-based vulnerabilities and the existing penetration testing approaches do not consider the mobile cloud computing application. In VeriDevOps, we will collaborate with Fagor (one of the project's industrial partners in the project) that offers a cloud-based IoT solution to develop new penetration testing approaches to test their IoT solution for vulnerabilities using machine learning and heuristic algorithms.

To avoid ripple effects from failures and to ensure that vulnerabilities are identified as fast as possible, it is paramount for the industry to ensure that the systems satisfy their functional safety

[44] https://www.stamp-project.eu/view/main/

requirements throughout the development and operation process. We plan to improve upon the current state of the art by developing an organizational and process model blueprint that is used to create a domain-driven formal model of security to tackle the non-existing ability in many approaches to define, model, execute and test security requirements from a system point-of-view. In addition, VeriDevOps is focusing on overcoming the obstacles to the adoption of DevOps by examining its role in specific standards, such as IEC/ISO standards and how to provide fast feedback on checking security requirements. The main goal of the VeriDevops techniques is to reduce the number of exploitable vulnerabilities and cost of security protection.

5. Summary

The objective of this document is to extend the literature review conducted in the project proposal by collecting and classifying the new approaches and technologies related to automated generation of security requirements, prevention at development and reactive protection at operations that appeared since the project proposal was submitted.

Chapter 2 discussed security requirement engineering approaches using NLP. We identified that, in most cases, an approach was designed to solve a specific problem using a corresponding dataset depending on the type of requirement context. Furthermore, there is a need for a dataset to train an NLP model in the context of security requirements. In VeriDevOps, we aim to proceed from natural language requirements to their verification seamlessly. Also, we plan to construct a dataset for security requirement patterns by manually analyzing the requirement documents and security verification procedures resulting from these documents.

Chapter 3 investigated several studies related to risk analysis and vulnerability scanning of industrial control systems employed in many domains such as energy, automotive, healthcare, avionics, and telecommunications. In ICS, the cyber-physical infrastructures must protect the physical components, such as sensors, controllers, actuators, and the cyber components such as computing nodes and communication channels against security threats. We noticed that traditional current risk assessment techniques do not update the risk assessment at run-time based on continuously collected data through monitoring. In VeriDevOps, we aim to develop different risk-analysis and assessment techniques that can be applied to industrial systems during their operation. In the chapter, we reviewed several anomaly-based intrusion detection approaches. We found out that these approaches are not mature enough to give a confident level of alerts or explain the source of the anomalies on the system. In VeriDevOps, we plan to extend the Montimage Monitoring Tool (MMT) to identify, predict and forecast attacks and anomalies in distributed environments using heuristic and machine learning algorithms. Further, VeriDevOps will define intelligent defense strategies to accurately diagnose a detected security incident and automatically deploy the relevant security mechanism.

Chapter 4 surveyed different techniques for formal analysis and verification for security and different security testing methods such as model-based security testing and penetration testing. We have highlighted several challenges and limitations concerning security properties for testing, coverage criteria, test selection and generation, scalability of penetration and model-based testing techniques for industrial applications. In VeriDevOps, we plan to develop specific test generation and selection methods based on formal specifications that can be used for model-based security testing given generic security models. These models can be built to be reused in different contexts and stages of the software development life cycle. We will develop new approaches for model-based mutation testing and fuzz testing by defining new model-level mutation operators suitable for testing security aspects. In addition, we will develop new penetration testing approaches to test a cloud-based IoT solution of Fagor (one of the project's industrial partners in the project) for vulnerabilities using machine learning and heuristic algorithms. Lastly, we discuss methods for adopting security testing into Continuous Integration (CI) systems and DevOps workflows. It indicated that traditional security methods are not

compatible with DevOps workflows. Further, generally, developers cannot get valuable test feedback from CI systems (executing the security test cases) to debug the security issue. In VeriDevOps, we aim to improve the current state of the art by developing an organizational and process model blueprint to overcome the non-existing ability in many approaches to define, model, execute and test security requirements from a system-point-of-view. Moreover, we plan to provide fast feedback to the developers on testing the security requirements by adopting DevOps and its role in specific standards, such as IEC/ISO standards.

6. References

[1] P. Loucopoulos and V. Karakostas, *System Requirements Engineering*. 1995, p. 160.

[2] C. Ma, H. Zheng, P. Xie, C. Li, L. Li, and L. Si, "DM_NLP at SemEval-2018 Task 8: neural sequence labeling with linguistic features," *Proceedings of The 12th International Workshop on Semantic Evaluation*. 2018 [Online]. Available: http://dx.doi.org/10.18653/v1/s18-1114

[3] K. S. Hoo, "Tangible ROI through secure software engineering," *Security Business Quarterly*, 2001.

[4] T. Li and Z. Chen, "An ontology-based learning approach for automatically classifying security requirements," *J. Syst. Softw.*, p. 110566, 2020.

[5] E. Knauss, S. Houmb, K. Schneider, S. Islam, and J. Jürjens, "Supporting requirements engineers in recognising security issues," in *International Working Conference on Requirements Engineering: Foundation for Software Quality*, 2011, pp. 4–18.

[6] C. B. Haley, R. Laney, J. D. Moffett, and B. Nuseibeh, "Security requirements engineering: A framework for representation and analysis," *IEEE Trans. Software Eng.*, vol. 34, no. 1, pp. 133–153, Jan. 2008, doi: 10.1109/tse.2007.70754. [Online]. Available: http://ieeexplore.ieee.org/document/4359475/

[7] M. Kassab, C. Neill, and P. Laplante, "State of Practice in Requirements Engineering: Contemporary Data," *Innovations in Systems and Software Engineering: A NASA Journal*, Dec. 2014, doi: 10.1007/s11334-014-0232-4. [Online]. Available: http://dx.doi.org/10.1007/s11334-014-0232-4

[8] L. Mich, M. Franch, and P. L. Novi Inverardi, "Market Research for Requirements Analysis Using Linguistic Tools," *Requirements Engineering*, vol. 9, pp. 40–56, Jan. 2004, doi: 10.1007/s00766-003-0179-8. [Online]. Available: http://dx.doi.org/10.1007/s00766-003-0179-8

[9] P. Sawyer, P. Rayson, and K. Cosh, "Shallow knowledge as an aid to deep understanding in early phase requirements engineering," *IEEE Trans. Software Eng.*, vol. 31, no. 11, pp. 969–981, Nov. 2005, doi: 10.1109/TSE.2005.129. [Online]. Available: http://dx.doi.org/10.1109/TSE.2005.129

[10] D. Jurafsky and C. Manning, "Natural language processing," *Instructor*, vol. 212, no. 998, p. 3482, 2012.

[11] E. D. Liddy, "Natural language processing," 2001 [Online]. Available: http://surface.syr.edu/cgi/viewcontent.cgi?article=1019&context=cnlp

[12] L. Zhao *et al.*, "Natural Language Processing for Requirements Engineering: A Systematic Mapping Study," Apr. 2020.

[13] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.

[14] S. Kommrusch, "Artificial Intelligence Techniques for Security Vulnerability Prevention," *arXiv preprint arXiv:1912. 06796*, 2019.

[15] A. Tharwat, "Classification assessment methods," *Applied Computing and Informatics*, 2020.

[16] Z. S. H. Abad, O. Karras, P. Ghazi, M. Glinz, G. Ruhe, and K. Schneider, "What works better? A study of classifying requirements," in *2017 IEEE 25th International Requirements Engineering Conference (RE)*, Lisbon, Portugal, 2017, doi: 10.1109/re.2017.36 [Online]. Available: http://ieeexplore.ieee.org/document/8049172/

[17] C. D. Manning, M. Surdeanu, J. Bauer, J. R. Finkel, S. Bethard, and D. McClosky, "The Stanford CoreNLP natural language processing toolkit," in *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, 2014, pp. 55–60.

[18] E. Boutkova and F. Houdek, "Semi-automatic identification of features in requirement specifications," in *2011 IEEE 19th International Requirements Engineering Conference*, 2011, pp. 313–318.

[19] R. Malhotra, A. Chug, A. Hayrapetian, and R. Raje, “Analyzing and evaluating security features in software requirements,” in *2016 International Conference on Innovation and Challenges in Cyber Security (ICICCS-INBUSH)*, 2016, pp. 26–30.
[20] N. F. Noy *et al.*, “Protégé-2000: an open-source ontology-development and knowledge-acquisition environment,” in *AMIA... Annual Symposium proceedings. AMIA Symposium*, 2003, pp. 953–953.
[21] A. Hayrapetian and R. Raje, “Empirically Analyzing and Evaluating Security Features in Software Requirements,” in *Proceedings of the 11th Innovations in Software Engineering Conference*, 2018, pp. 1–11.
[22] B. Magnini *et al.*, “The excitement open platform for textual inferences,” in *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, 2014, pp. 43–48.
[23] W. Wang, K. R. Mahakala, A. Gupta, N. Hussein, and Y. Wang, “A linear classifier based approach for identifying security requirements in open source software development,” *Journal of Industrial Information Integration*, vol. 14, pp. 34–40, 2019.
[24] Z. Kurtanović and W. Maalej, “Automatically classifying functional and non-functional requirements using supervised machine learning,” in *2017 IEEE 25th International Requirements Engineering Conference (RE)*, 2017, pp. 490–495.
[25] J. Cleland-Huang, S. Mazrouee, H. Liguo, and D. Port, “nfr,” *Certification Commission for Health Information Technology*. Zenodo, Mar-2007 [Online]. Available: https://doi.org/10.5281/zenodo.268542
[26] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015, doi: 10.1038/nature14539. [Online]. Available: http://www.nature.com/articles/nature14539
[27] Y. LeCun *et al.*, “Backpropagation applied to handwritten zip code recognition,” *Neural Comput.*, vol. 1, no. 4, pp. 541–551, 1989.
[28] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient estimation of word representations in vector space,” *arXiv preprint arXiv:1301. 3781*, 2013.
[29] J. Winkler and A. Vogelsang, “Automatic classification of requirements based on convolutional neural networks,” in *2016 IEEE 24th International Requirements Engineering Conference Workshops (REW)*, 2016, pp. 39–45.
[30] T. Hey, J. Keim, A. Koziolek, and W. F. Tichy, “NoRBERT: Transfer learning for requirements classification,” in *2020 IEEE 28th International Requirements Engineering Conference (RE)*, Zurich, Switzerland, 2020, doi: 10.1109/re48521.2020.00028 [Online]. Available: https://ieeexplore.ieee.org/document/9218141/
[31] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” *arXiv [cs.CL]*, 11-Oct-2018 [Online]. Available: http://arxiv.org/abs/1810.04805
[32] M. Bozorgi, L. K. Saul, S. Savage, and G. M. Voelker, “Beyond heuristics: learning to classify vulnerabilities and predict exploits,” in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2010, pp. 105–114.
[33] R. Pandita, X. Xiao, W. Yang, W. Enck, and T. Xie, “${WHYPER}$: Towards automating risk assessment of mobile applications,” in *22nd ${USENIX} Security Symposium ({USENIX}$ Security 13)*, 2013, pp. 527–542.
[34] A. Ferrari, G. O. Spagnolo, and S. Gnesi, “PURE: A Dataset of Public Requirements Documents,” in *2017 IEEE 25th International Requirements Engineering Conference (RE)*, Lisbon, Portugal, 2017, doi: 10.1109/re.2017.29 [Online]. Available: http://ieeexplore.ieee.org/document/8049173/
[35] J. Cleland-Huang, R. Settimi, X. Zou, and P. Solc, “Automated classification of non-functional

requirements," *Requirements Engineering*, vol. 12, no. 2, pp. 103–120, 2007.
[36] A. Rashwan, O. Ormandjieva, and R. Witte, "Ontology-Based Classification of Non-Functional Requirements in Software Specifications: A new Corpus and SVM-Based Classifier," in *The 37th Annual International Computer Software & Applications Conference (COMPSAC 2013)*, 2013, p. 381\textendash386, doi: 10.1109/COMPSAC.2013.64 [Online]. Available: http://dx.doi.org/10.1109/COMPSAC.2013.64
[37] S. Konrad, B. H. C. Cheng, L. A. Campbell, and R. Wassermann, "Using security patterns to model and analyze security requirements," *Requirements Engineering for High Assurance Systems (RHAS'03)*, vol. 11, 2003 [Online]. Available: https://documentcloud.adobe.com/link/review?uri=urn:aaid:scds:US:df310295-34de-492a-b389-5359146eed19
[38] E. Gamma, R. Helm, R. Johnson, J. Vlissides, and D. Patterns, "Elements of Reusable Object-Oriented Software," *Design Patterns. massachusetts: Addison-Wesley Publishing Company*, 1995.
[39] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett, "Patterns in property specifications for finite-state verification," *Proceedings of the 21st international conference on Software engineering - ICSE '99*. 1999 [Online]. Available: http://dx.doi.org/10.1145/302405.302672
[40] R. Wassermann and B. H. C. Cheng, "Security patterns," in *Michigan State University, PLoP Conf. Citeseer*, 2003 [Online]. Available: https://documentcloud.adobe.com/link/review?uri=urn:aaid:scds:US:562417be-c44d-47d7-a9c5-bc366f207ca9
[41] N. Yoshioka, H. Washizaki, and K. Maruyama, "A survey on security patterns," *Progress in informatics*, vol. 5, no. 5, pp. 35–47, 2008 [Online]. Available: https://documentcloud.adobe.com/link/review?uri=urn:aaid:scds:US:4c50be1b-1da2-40d6-812b-ce345f334208
[42] J. Jürjens, G. Popp, and G. Wimmel, "Towards using security patterns in model-based system development," 2002.
[43] J. Jürjens, "UMLsec: Extending UML for secure systems development," in *International Conference on The Unified Modeling Language*, 2002, pp. 412–425.
[44] S. Ouchani and M. Debbabi, "Specification, verification, and quantification of security in model-based systems," *Computing*, vol. 97, no. 7, pp. 691–711, Jul. 2015, doi: 10.1007/s00607-015-0445-x. [Online]. Available: http://link.springer.com/10.1007/s00607-015-0445-x
[45] I. Siveroni, A. Zisman, and G. Spanoudakis, "Property specification and static verification of UML models," in *2008 Third International Conference on Availability, Reliability and Security*, 2008, doi: 10.1109/ares.2008.194 [Online]. Available: http://ieeexplore.ieee.org/document/4529326/
[46] I. Siveroni, A. Zisman, and G. Spanoudakis, "A UML-based static verification framework for security," *Requirements engineering*, vol. 15, no. 1, pp. 95–118, 2010 [Online]. Available: https://documentcloud.adobe.com/link/review?uri=urn:aaid:scds:US:fb5db61a-df01-4a99-beb1-a5affd89def1
[47] A. Zisman, "A static verification framework for secure peer-to-peer applications," in *Second International Conference on Internet and Web Applications and Services (ICIW'07)*, 2007, pp. 8–8 [Online]. Available: https://documentcloud.adobe.com/link/review?uri=urn:aaid:scds:US:c142f282-81ef-4185-afd6-72eff2323f1e
[48] J. Dong, T. Peng, and Y. Zhao, "Automated verification of security pattern compositions," *Information and Software Technology*, vol. 52, no. 3, pp. 274–295, 2010 [Online]. Available:

https://documentcloud.adobe.com/link/review?uri=urn:aaid:scds:US:9f79cd8f-663c-420c-936b-544e078857a3

[49] S. Ouchani, O. A. Mohamed, and M. Debbabi, “A security risk assessment framework for SysML activity diagrams,” in *2013 IEEE 7th International Conference on Software Security and Reliability*, Gaithersburg, MD, USA, 2013, doi: 10.1109/sere.2013.11 [Online]. Available: https://ieeexplore.ieee.org/document/6571713

[50] S. Ouchani, O. A. Mohamed, M. Debbabi, and M. Pourzandi, “Verification of the correctness in composed UML behavioural diagrams,” in *Software Engineering Research, Management and Applications 2010*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 163–177 [Online]. Available: http://link.springer.com/10.1007/978-3-642-13273-5_11

[51] S. Ouchani, Y. Jarraya, and O. A. Mohamed, “Model-based systems security quantification,” in *2011 Ninth Annual International Conference on Privacy, Security and Trust*, 2011, pp. 142–149 [Online]. Available: https://documentcloud.adobe.com/link/review?uri=urn:aaid:scds:US:2f653129-e94b-46e0-a9e5-55614133ab3b

[52] M. Kwiatkowska, G. Norman, and D. Parker, “PRISM: Probabilistic symbolic model checker,” in *International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, 2002, pp. 200–204.

[53] I. Kotenko and E. Doynikova, “The CAPEC based generator of attack scenarios for network security evaluation,” in *2015 IEEE 8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, Warsaw, Poland, 2015, doi: 10.1109/idaacs.2015.7340774 [Online]. Available: http://ieeexplore.ieee.org/document/7340774/

[54] K. Kanakogi *et al.*, “Tracing CAPEC Attack Patterns from CVE Vulnerability Information using Natural Language Processing Technique,” in *Proceedings of the 54th Hawaii International Conference on System Sciences*, 2021, p. 6996 [Online]. Available: https://documentcloud.adobe.com/link/review?uri=urn:aaid:scds:US:00a8e50d-b031-407b-bcd7-964e32f5ec33

[55] Q. Le and T. Mikolov, “Distributed representations of sentences and documents,” in *International conference on machine learning*, 2014, pp. 1188–1196.

[56] X. Yuan, E. B. Nuakoh, J. S. Beal, and H. Yu, “Retrieving relevant CAPEC attack patterns for secure software development,” in *Proceedings of the 9th Annual Cyber and Information Security Research Conference*, 2014, pp. 33–36 [Online]. Available: https://documentcloud.adobe.com/link/review?uri=urn:aaid:scds:US:717194d1-5823-4c79-a130-3f4ad8537ab6

[57] A. Shostack, “Experiences Threat Modeling at Microsoft,” *MODSEC@ MoDELS*, vol. 2008, 2008.

[58] D. R. H. Miller, T. Leek, and R. M. Schwartz, “A hidden Markov model information retrieval system,” in *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval - SIGIR ’99*, Berkeley, California, United States, 1999, doi: 10.1145/312624.312680 [Online]. Available: http://portal.acm.org/citation.cfm?doid=312624.312680

[59] H. Kaiya *et al.*, “Security requirements analysis using knowledge in CAPEC,” in *International conference on advanced information systems engineering*, 2014, pp. 343–348 [Online]. Available: https://documentcloud.adobe.com/link/review?uri=urn:aaid:scds:US:081d63c7-ceaf-4c7a-8f59-bff759ded9cb

[60] I. Williams, “An ontology based collaborative recommender system for security requirements elicitation,” in *2018 IEEE 26th International Requirements Engineering Conference (RE)*, Banff, AB, 2018, doi: 10.1109/re.2018.00060 [Online]. Available:

https://ieeexplore.ieee.org/document/8491167/
[61] I. Williams and X. Yuan, “Creating abuse cases based on attack patterns: A user study,” in *2017 IEEE Cybersecurity Development (SecDev)*, Cambridge, MA, USA, 2017, doi: 10.1109/secdev.2017.27 [Online]. Available: http://ieeexplore.ieee.org/document/8077812/
[62] G. McGraw, “Software security,” *IEEE Secur. Priv.*, vol. 2, no. 2, pp. 80–83, 2004.
[63] A. Sudhodanan, A. Armando, R. Carbone, L. Compagna, and Others, “Attack Patterns for Black-Box Security Testing of Multi-Party Web Applications,” in *NDSS*, 2016 [Online]. Available: https://documentcloud.adobe.com/link/review?uri=urn:aaid:scds:US:6057a4db-921a-43f3-a273-01bc59670632
[64] J. Bozic, D. E. Simos, and F. Wotawa, “Attack pattern-based combinatorial testing,” in *Proceedings of the 9th International Workshop on Automation of Software Test - AST 2014*, Hyderabad, India, 2014, doi: 10.1145/2593501.2593502 [Online]. Available: http://dl.acm.org/citation.cfm?doid=2593501.2593502
[65] L. Yu, Y. Lei, R. N. Kacker, and D. R. Kuhn, “ACTS: A Combinatorial Test Generation Tool,” in *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*, Luxembourg, Luxembourg, 2013, doi: 10.1109/icst.2013.52 [Online]. Available: http://ieeexplore.ieee.org/document/6569749/
[66] B. Smith and L. Williams, “On the effective use of security test patterns,” in *2012 IEEE Sixth International Conference on Software Security and Reliability*, 2012, pp. 108–117 [Online]. Available: https://documentcloud.adobe.com/link/review?uri=urn:aaid:scds:US:4a205482-ab00-49c1-b729-4e496738b369
[67] M. Felderer, M. Büchler, M. Johns, A. D. Brucker, R. Breu, and A. Pretschner, “Security testing: A survey,” in *Advances in Computers*, vol. 101, Elsevier, 2016, pp. 1–51 [Online]. Available: https://documentcloud.adobe.com/link/review?uri=urn:aaid:scds:US:8493d713-4d8c-4b16-b380-721e9a581325
[68] J. Großmann, M. Schneider, J. Viehmann, and M.-F. Wendland, “Combining risk analysis and security testing,” in *International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*, 2014, pp. 322–336 [Online]. Available: https://documentcloud.adobe.com/link/review?uri=urn:aaid:scds:US:dc5262bc-68ec-46b2-9dfb-ce441abb02ae
[69] M. S. Lund, B. Solhaug, and K. Stølen, *Model-driven risk analysis: the CORAS approach*. Springer Science & Business Media, 2010.
[70] F. Bouquet, C. Grandpierre, B. Legeard, and F. Peureux, “A test generation solution to automate software testing,” in *Proceedings of the 3rd international workshop on Automation of software test*, 2008, pp. 45–48.
[71] J. Botella, B. Legeard, F. Peureux, and A. Vernotte, “Risk-Based Vulnerability Testing Using Security Test Patterns,” *Leveraging Applications of Formal Methods, Verification and Validation. Specialized Techniques and Applications*. pp. 337–352, 2014 [Online]. Available: http://dx.doi.org/10.1007/978-3-662-45231-8_24
[72] A. G. Vouffo Feudjio, “Initial Security Test Pattern Catalog. Public Deliverable D3. WP4. T1, Diamonds Project, Berlin, Germany (June 2012).” 2014 [Online]. Available: https://documentcloud.adobe.com/link/review?uri=urn:aaid:scds:US:1ea59932-35c4-4ed0-9094-47d37db8e0b3
[73] A.-G. V. Feudjio, I. Schieferdecker, and A. Vouffo, “Test Automation Design Patterns for Reactive Software Systems,” *ceur-ws. org*, vol. 566, p. E6_BlackBoxTesting, 2009 [Online]. Available: https://documentcloud.adobe.com/link/review?uri=urn:aaid:scds:US:89e33581-7576-432d-a3ad-

29a46b8e79af
[74] F. Wotawa and J. Bozic, “Plan it! automated security testing based on planning,” in *IFIP International Conference on Testing Software and Systems*, 2014, pp. 48–62 [Online]. Available: https://documentcloud.adobe.com/link/review?uri=urn:aaid:scds:US:a706f532-07d2-4a59-8a8c-880c4f8bd01a
[75] A. Blome, M. Ochoa, K. Li, M. Peroli, and M. T. Dashti, “Vera: A flexible model-based vulnerability testing tool,” in *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*, 2013, pp. 471–478.
[76] W. Herzner, S. Sieverding, O. Kacimi, E. Bode, T. Bauer, and B. Nielsen, “Expressing best practices in (risk) analysis and testing of safety-critical systems using patterns,” in *2014 IEEE International Symposium on Software Reliability Engineering Workshops*, Naples, Italy, 2014, doi: 10.1109/issrew.2014.24 [Online]. Available: https://ieeexplore.ieee.org/document/6983857
[77] D. Dghaym *et al.*, “Systematic Verification and Testing,” in *Validation and Verification of Automated Systems*, Springer, 2020, pp. 89–104 [Online]. Available: https://documentcloud.adobe.com/link/review?uri=urn:aaid:scds:US:fd5d01f8-9306-48c4-bd76-822702af0173
[78] A. Vernotte, “A pattern-driven and model-based vulnerability testing for Web applications,” Université de Franche-Comté, 2015 [Online]. Available: https://documentcloud.adobe.com/link/review?uri=urn:aaid:scds:US:9331271b-d98c-4561-adc0-7bbe94821210
[79] A. Naumchev, “Seamless Object-Oriented Requirements,” in *2019 International Multi-Conference on Engineering, Computer and Information Sciences (SIBIRCON)*, 2019, pp. 0743–0748.
[80] N. Tillmann and W. Schulte, “Parameterized unit tests,” *ACM SIGSOFT Software Engineering Notes*, vol. 30, no. 5, pp. 253–262, 2005.
[81] J. Botella, B. Legeard, F. Peureux, and A. Vernotte, “Risk-based vulnerability testing using security test patterns,” in *International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*, 2014, pp. 337–352 [Online]. Available: https://documentcloud.adobe.com/link/review?uri=urn:aaid:scds:US:decdf192-2e7b-45cf-b6fe-0c517eb8b764
[82] J. Bozic and F. Wotawa, “Security testing based on attack patterns,” in *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops*, OH, USA, 2014, doi: 10.1109/icstw.2014.58 [Online]. Available: http://ieeexplore.ieee.org/document/6825631/
[83] D. Prince, “Cybersecurity: The Security and Protection Challenges of Our Digital World,” *Computer*, vol. 51, no. 4. pp. 16–19, 2018 [Online]. Available: http://dx.doi.org/10.1109/mc.2018.2141025
[84] P. Marwedel, *Embedded System Design: Embedded Systems Foundations of Cyber-Physical Systems, and the Internet of Things*. Springer, 2017 [Online]. Available: https://play.google.com/store/books/details?id=6JkuDwAAQBAJ
[85] R. Langner, “Stuxnet: Dissecting a Cyberwarfare Weapon,” *IEEE Security Privacy*, vol. 9, no. 3, pp. 49–51, May 2011, doi: 10.1109/MSP.2011.67. [Online]. Available: http://dx.doi.org/10.1109/MSP.2011.67
[86] I. C. L. I. Toolkit, “Steel mill in Germany (2014) --- International cyber law: interactive toolkit,.” 2020 [Online]. Available: https://cyberlaw.ccdcoe.org/w/index.php?title=Steel_mill_in_Germany_(2014)&oldid=2282
[87] C. Thompson and D. Wagner, “A Large-Scale Study of Modern Code Review and Security in Open Source Projects,” *Proceedings of the 13th International Conference on Predictive Models and Data Analytics in Software Engineering - PROMISE*. 2017 [Online]. Available:

http://dx.doi.org/10.1145/3127005.3127014
[88] X. Franch and A. Susi, “Risk assessment in open source systems,” *Proceedings of the 38th International Conference on Software Engineering Companion - ICSE ’16*. 2016 [Online]. Available: http://dx.doi.org/10.1145/2889160.2891052
[89] A. Salamai, O. Hussain, and M. Saberi, “Decision Support System for Risk Assessment Using Fuzzy Inference in Supply Chain Big Data,” *2019 International Conference on High Performance Big Data and Intelligent Systems (HPBD&IS)*. 2019 [Online]. Available: http://dx.doi.org/10.1109/hpbdis.2019.8735465
[90] National Standards Authority of Ireland, *Risk Management: Risk Assessment Techniques (IEC/ISO 31010:2009 (EQV)*. 2013 [Online]. Available: https://books.google.com/books/about/Risk_Management.html?hl=&id=mqsXogEACAAJ
[91] V. Mahajan, H. A. Linstone, and M. Turoff, “The Delphi Method: Techniques and Applications,” *Journal of Marketing Research*, vol. 13, no. 3. p. 317, 1976 [Online]. Available: http://dx.doi.org/10.2307/3150755
[92] “Hazard and operability studies (HAZOP studies). Application guide.” [Online]. Available: http://dx.doi.org/10.3403/02337615u
[93] J. Dalton, “SWOT Analysis (Strengths, Weaknesses, Opportunities, Threats),” *Great Big Agile*. pp. 249–252, 2019 [Online]. Available: http://dx.doi.org/10.1007/978-1-4842-4206-3_62
[94] A. Bouti and D. A. Kadi, “A STATE-OF-THE-ART REVIEW OF FMEA/FMECA,” *International Journal of Reliability, Quality and Safety Engineering*, vol. 01, no. 04. pp. 515–543, 1994 [Online]. Available: http://dx.doi.org/10.1142/s0218539394000362
[95] R. M. Robinson and G. Francis, *Risk and Reliability: An Introductory Text*. 2007 [Online]. Available: https://books.google.com/books/about/Risk_and_Reliability.html?hl=&id=v2jIGAAACAAJ
[96] I. Ben-Gal, “Bayesian Networks,” *Encyclopedia of Statistics in Quality and Reliability*. 2008 [Online]. Available: http://dx.doi.org/10.1002/9780470061572.eqr089
[97] S. Mannan, *Lees’ Loss Prevention in the Process Industries: Hazard Identification, Assessment and Control*. Butterworth-Heinemann, 2004 [Online]. Available: https://books.google.com/books/about/Lees_Loss_Prevention_in_the_Process_Indu.html?hl=&id=UDAwZQO8ZGUC
[98] M. S. Lund, B. Solhaug, and K. Stølen, “Model-Driven Risk Analysis.” 2011 [Online]. Available: http://dx.doi.org/10.1007/978-3-642-12323-8
[99] M. Kravchik and A. Shabtai, “Detecting Cyber Attacks in Industrial Control Systems Using Convolutional Neural Networks,” *Proceedings of the 2018 Workshop on Cyber-Physical Systems Security and PrivaCy*. 2018 [Online]. Available: http://dx.doi.org/10.1145/3264888.3264896
[100] K. Stouffer, J. Falco, and K. Scarfone, “Guide to Industrial Control Systems (ICS) Security - Supervisory Control and Data Acquisition (SCADA) systems, Distributed Control Systems (DCS), and other control system configurations such as Programmable Logic Controllers (PLC).” 2011 [Online]. Available: http://dx.doi.org/10.6028/nist.sp.800.82
[101] “Communication network dependencies for ICS/SCADA Systems,” 19-Dec-2016. [Online]. Available: https://www.enisa.europa.eu/publications/ics-scada-dependencies. [Accessed: 26-Jan-2021]
[102] S. Qayyum, S. Ashraf, M. Shafique, and S. Waheed, “Hardware devices security, their vulnerabilities and solutions,” in *2018 15th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*, 2018, pp. 439–445, doi: 10.1109/IBCAST.2018.8312261 [Online]. Available: http://dx.doi.org/10.1109/IBCAST.2018.8312261
[103] O. Andreeva *et al.*, “Industrial control systems vulnerabilities statistics,” *Kaspersky Lab, Report*, 2016 [Online]. Available:

https://www.researchgate.net/profile/Sergey_Gordeychik/publication/337732465_INDUSTRIAL_CONTROL_SYSTEMS_VULNERABILITIES_STATISTICS/links/5de7842e92851c8364600e7e/INDUSTRIAL-CONTROL-SYSTEMS-VULNERABILITIES-STATISTICS.pdf

[104] B. Schneier and L. Secrets, "Digital security in a networked world," *John Wiley and Sons Inc*, vol. 5, no. 3, pp. 300–315, 2000.

[105] P. Foreman, *Vulnerability Management*. CRC Press, 2019 [Online]. Available: https://play.google.com/store/books/details?id=owadDwAAQBAJ

[106] L. K. Seng, N. Ithnin, and S. Z. M. Said, "The approaches to quantify web application security scanners quality: a review," *International Journal of Advanced Computer Research*, vol. 8, no. 38. pp. 285–312, 2018 [Online]. Available: http://dx.doi.org/10.19101/ijacr.2018.838012

[107] N. Uddin Sheikh, H. Rahman, S. Vikram, and H. AlQahtani, "A Lightweight Signature-Based IDS for IoT Environment," *arXiv e-prints*, p. arXiv:1811.04582, Nov. 2018 [Online]. Available: http://arxiv.org/abs/1811.04582

[108] M. Masdari and H. Khezri, "A survey and taxonomy of the fuzzy signature-based Intrusion Detection Systems," *Appl. Soft Comput.*, vol. 92, p. 106301, 2020, doi: 10.1016/j.asoc.2020.106301. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1568494620302416

[109] A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman, "Survey of intrusion detection systems: techniques, datasets and challenges," *Cybersecurity*, vol. 2, no. 1. 2019 [Online]. Available: http://dx.doi.org/10.1186/s42400-019-0038-7

[110] Yu Wang, Weizhi Meng, Wenjuan Li, Jin Li, Wai-Xi Liu, Yang Xiang, "A fog-based privacy-preserving approach for distributed signature-based intrusion detection," *J. Parallel Distrib. Comput.*, vol. 122, pp. 26–35, 2018, doi: 10.1016/j.jpdc.2018.07.013. [Online]. Available: https://doi.org/10.1016/j.jpdc.2018.07.013.

[111] D. J. Richey, *Leveraging PLC Ladder Logic for Signature Based IDS Rule Generation*. 2016 [Online]. Available: https://books.google.com/books/about/Leveraging_PLC_Ladder_Logic_for_Signatur.html?hl=&id=dM4vswEACAAJ

[112] R. H. Gong, M. Zulkernine, and P. Abolmaesumi, "A Software Implementation of a Genetic Algorithm Based Approach to Network Intrusion Detection," *Sixth International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing and First ACIS International Workshop on Self-Assembling Wireless Networks (SNPD/SAWN'05)*. [Online]. Available: http://dx.doi.org/10.1109/snpd-sawn.2005.9

[113] ECLab: evolutionary Computation laboratory, "A Java-based Evolutionary Computation (ECJ) and Genetic Programming Research System." [Online]. Available: https://cs.gmu.edu/~eclab/projects/ecj/. [Accessed: 15-Dec-2020]

[114] "1998 DARPA Intrusion Detection Evaluation Dataset," Mar-1998. [Online]. Available: https://www.ll.mit.edu/r-d/datasets/1998-darpa-intrusion-detection-evaluation-dataset. [Accessed: 15-Dec-2020]

[115] C. D. Grosso, C. Del Grosso, G. Antoniol, M. Di Penta, P. Galinier, and E. Merlo, "Improving network applications security," *Proceedings of the 2005 conference on Genetic and evolutionary computation - GECCO '05*. 2005 [Online]. Available: http://dx.doi.org/10.1145/1068009.1068185

[116] C. D. Grosso, C. Del Grosso, G. Antoniol, E. Merlo, and P. Galinier, "Detecting buffer overflow via automatic test input data generation," *Computers & Operations Research*, vol. 35, no. 10. pp. 3125–3143, 2008 [Online]. Available: http://dx.doi.org/10.1016/j.cor.2007.01.013

[117] G. Dozier, D. Brown, H. Hou, and J. Hurley, "Vulnerability analysis of immunity-based intrusion detection systems using genetic and evolutionary hackers," *Applied Soft Computing*, vol. 7, no. 2.

pp. 547–553, 2007 [Online]. Available: http://dx.doi.org/10.1016/j.asoc.2006.05.001

[118] H. Hou and G. Dozier, “Immunity-based intrusion detection system design, vulnerability analysis, and GENERTIA’s genetic arms race,” *Proceedings of the 2005 ACM symposium on Applied computing - SAC ’05*. 2005 [Online]. Available: http://dx.doi.org/10.1145/1066677.1066895

[119] J. Campos, R. Abreu, G. Fraser, and M. d’Amorim, “Entropy-based test generation for improved fault localization,” *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 2013 [Online]. Available: http://dx.doi.org/10.1109/ase.2013.6693085

[120] G. Fraser and A. Arcuri, “EvoSuite,” *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering - SIGSOFT/FSE ’11*. 2011 [Online]. Available: http://dx.doi.org/10.1145/2025113.2025179

[121] J. P. Galeotti, G. Fraser, and A. Arcuri, “Improving search-based test suite generation with dynamic symbolic execution,” *2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE)*. 2013 [Online]. Available: http://dx.doi.org/10.1109/issre.2013.6698889

[122] N. Tracey, J. Clark, J. McDermid, and K. Mander, “A Search-Based Automated Test-Data Generation Framework for Safety-Critical Systems,” *Systems Engineering for Business Process Change: New Directions*. pp. 174–213, 2002 [Online]. Available: http://dx.doi.org/10.1007/978-1-4471-0135-2_12

[123] W. Afzal, R. Torkar, and R. Feldt, “A systematic review of search-based testing for non-functional system properties,” *Information and Software Technology*, vol. 51, no. 6. pp. 957–976, 2009 [Online]. Available: http://dx.doi.org/10.1016/j.infsof.2008.12.005

[124] D. Bhamare, M. Zolanvari, A. Erbad, R. Jain, K. Khan, and N. Meskin, “Cybersecurity for Industrial Control Systems: A Survey,” *arXiv [cs.CR]*. 2020 [Online]. Available: http://arxiv.org/abs/2002.04124

[125] E. Hodo, X. Bellekens, A. Hamilton, C. Tachtatzis, and R. Atkinson, “Shallow and Deep Networks Intrusion Detection System: A Taxonomy and Survey,” *arXiv [cs.CR]*. 2017 [Online]. Available: http://arxiv.org/abs/1701.02145

[126] H. Liu and B. Lang, “Machine Learning and Deep Learning Methods for Intrusion Detection Systems: A Survey,” *NATO Adv. Sci. Inst. Ser. E Appl. Sci.*, vol. 9, no. 20, p. 4396, Oct. 2019, doi: 10.3390/app9204396. [Online]. Available: https://www.mdpi.com/2076-3417/9/20/4396. [Accessed: 11-Feb-2021]

[127] K. A. P. da Costa, J. P. Papa, C. O. Lisboa, R. Munoz, and V. H. C. de Albuquerque, “Internet of Things: A survey on machine learning-based intrusion detection approaches,” *Computer Networks*, vol. 151, pp. 147–157, Mar. 2019, doi: 10.1016/j.comnet.2019.01.023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128618308739

[128] N. Sultana, N. Chilamkurti, W. Peng, and R. Alhadad, “Survey on SDN based network intrusion detection system using machine learning approaches,” *Peer-to-Peer Networking and Applications*, vol. 12, no. 2, pp. 493–501, Mar. 2019, doi: 10.1007/s12083-017-0630-0. [Online]. Available: https://doi.org/10.1007/s12083-017-0630-0

[129] V. Chandola, A. Banerjee, and V. Kumar, “Anomaly detection,” *ACM Computing Surveys*, vol. 41, no. 3. pp. 1–58, 2009 [Online]. Available: http://dx.doi.org/10.1145/1541880.1541882

[130] S. Lazarova-Molnar, H. R. Shaker, N. Mohamed, and B. N. J⊘rgensen, “Fault detection and diagnosis for smart buildings: State of the art, trends and challenges,” in *2016 3rd MEC International Conference on Big Data and Smart City (ICBDSC)*, 2016, pp. 1–7, doi: 10.1109/ICBDSC.2016.7460392 [Online]. Available: http://dx.doi.org/10.1109/ICBDSC.2016.7460392

[131] S. Katipamula and M. R. Brambley, “Review Article: Methods for Fault Detection, Diagnostics, and Prognostics for Building Systems—A Review, Part I,” *HVAC&R Research*, vol. 11, no. 1, pp.

3–25, Jan. 2005, doi: 10.1080/10789669.2005.10391123. [Online]. Available: https://www.tandfonline.com/doi/abs/10.1080/10789669.2005.10391123

[132] Z. Feng, M. Liang, and F. Chu, “Recent advances in time–frequency analysis methods for machinery fault diagnosis: A review with application examples,” *Mech. Syst. Signal Process.*, vol. 38, no. 1, pp. 165–205, Jul. 2013, doi: 10.1016/j.ymssp.2013.01.017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S088832701300071X

[133] N. Delgado, A. Q. Gates, and S. Roach, “A taxonomy and catalog of runtime software-fault monitoring tools,” *IEEE Trans. Software Eng.*, vol. 30, no. 12, pp. 859–872, Dec. 2004, doi: 10.1109/TSE.2004.91. [Online]. Available: http://dx.doi.org/10.1109/TSE.2004.91

[134] L. Qin, X. He, and D. H. Zhou, “A survey of fault diagnosis for swarm systems,” *Systems Science & Control Engineering*, vol. 2, no. 1, pp. 13–23, Dec. 2014, doi: 10.1080/21642583.2013.873745. [Online]. Available: https://doi.org/10.1080/21642583.2013.873745

[135] R. Isermann, “Model-based fault-detection and diagnosis – status and applications,” *Annual Reviews in Control*, vol. 29, no. 1. pp. 71–85, 2005 [Online]. Available: http://dx.doi.org/10.1016/j.arcontrol.2004.12.002

[136] I. Hwang, S. Kim, Y. Kim, and C. E. Seah, “A Survey of Fault Detection, Isolation, and Reconfiguration Methods,” *IEEE Trans. Control Syst. Technol.*, vol. 18, no. 3, pp. 636–653, May 2010, doi: 10.1109/TCST.2009.2026285. [Online]. Available: http://dx.doi.org/10.1109/TCST.2009.2026285

[137] P. E. Lanigan, S. Kavulya, P. Narasimhan, T. E. Fuhrman, and M. A. Salman, “Diagnosis in automotive systems: A survey,” *Last accessed Sept*, vol. 10, p. 2011, 2011 [Online]. Available: https://www.academia.edu/download/30795812/CMU-PDL-11-110.pdf

[138] J. Mohammadpour, M. Franchek, and K. Grigoriadis, “A survey on diagnostic methods for automotive engines,” *Int. J. Engine Res.*, vol. 13, no. 1, pp. 41–64, Feb. 2012, doi: 10.1177/1468087411422851. [Online]. Available: https://doi.org/10.1177/1468087411422851

[139] R. J. Patton, “Fault detection and diagnosis in aerospace systems using analytical redundancy,” *Computing & Control Engineering Journal*, vol. 2, no. 3, pp. 127–136, May 1991, doi: 10.1049/cce:19910031. [Online]. Available: https://digital-library.theiet.org/content/journals/10.1049/cce_19910031. [Accessed: 07-Feb-2021]

[140] Buchanan, G. Bruce, Shortliffe, and E. H. (editors), “Rule- Based Expert Systems : The MYCIN Experiments of the Stanford Heuristic Programming Project,” 1984 [Online]. Available: http://papers.cumincad.org/cgi-bin/works/paper/ec87. [Accessed: 07-Feb-2021]

[141] Silvianita, D. S. Mahandeka, and D. M. Rosyid, “Fault Tree Analysis for Investigation on the Causes of Project Problems,” *Procedia Earth and Planetary Science*, vol. 14, pp. 213–219, Jan. 2015, doi: 10.1016/j.proeps.2015.07.104. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1878522015002556

[142] A. Alaeddini and I. Dogan, “Using Bayesian networks for root cause analysis in statistical process control,” *Expert Syst. Appl.*, vol. 38, no. 9, pp. 11230–11243, Sep. 2011, doi: 10.1016/j.eswa.2011.02.171. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417411003952

[143] M. Chen, A. X. Zheng, J. Lloyd, M. I. Jordan, and E. Brewer, “Failure diagnosis using decision trees,” in *International Conference on Autonomic Computing, 2004. Proceedings.*, 2004, pp. 36–43, doi: 10.1109/ICAC.2004.1301345 [Online]. Available: http://dx.doi.org/10.1109/ICAC.2004.1301345

[144] M. Pelillo, Ed., *Similarity-Based Pattern Analysis and Recognition*. Springer, London, 2013 [Online]. Available: https://link.springer.com/10.1007/978-1-4471-5628-4

[145] I. Mugarza, J. L. Flores, and J. L. Montero, “Security Issues and Software Updates Management in the Industrial Internet of Things (IIoT) Era,” *Sensors* , vol. 20, no. 24, Dec. 2020, doi: 10.3390/s20247160. [Online]. Available: http://dx.doi.org/10.3390/s20247160

[146] Keith Stouffer, Suzanne Lightman, Victoria Pillitteri, Marshall Abrams, Adam Hahn, “Guide to Industrial Control Systems (ICS) Security,” NIST Special Publication, 800-82, May 2015.

[147] A. A. Jillepalli, F. T. Sheldon, D. C. de Leon, M. Haney, and R. K. Abercrombie, “Security management of cyber physical control systems using NIST SP 800-82r2,” *2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC)*. 2017 [Online]. Available: http://dx.doi.org/10.1109/iwcmc.2017.7986568

[148] International Electrotechnical Commission, “IEC TR 63069: Industrial-Process Measurement, Control and Automation—Framework for Functional Safety and Security,” IEC Central Office, Geneva, Switzerland, 2019.

[149] International Electrotechnical Commission, “IEC 62443: Security for Industrial Automation and Control Systems—Part 4-1: Secure Product Development Lifecycle Requirements,” IEC Central Office, Geneva, Switzerland, 2018.

[150] ENISA, “Actionable Information for Security Incident Response,” European Union Agency for Network and Information Security, Nov. 2014 [Online]. Available: https://www.enisa.europa.eu/publications/actionable-information-for-security

[151] MITRE, “ATT&CK® for Industrial Control Systems.” [Online]. Available: https://collaborate.mitre.org/attackics/index.php/Main_Page. [Accessed: 03-Apr-2021]

[152] M. Felderer, M. Büchler, M. Johns, A. D. Brucker, R. Breu, and A. Pretschner, “Security Testing,” *Advances in Computers*. pp. 1–51, 2016 [Online]. Available: http://dx.doi.org/10.1016/bs.adcom.2015.11.003

[153] S. Chong *et al.*, “Report on the NSF Workshop on Formal Methods for Security,” *arXiv [cs.CR]*, 02-Aug-2016 [Online]. Available: http://arxiv.org/abs/1608.00678

[154] A. Datta, J. Franklin, D. Garg, L. Jia, and D. Kaynar, “On Adversary Models and Compositional Security,” *IEEE Security & Privacy Magazine*, vol. 9, no. 3. pp. 26–32, 2011 [Online]. Available: http://dx.doi.org/10.1109/msp.2010.203

[155] Clark Barrett, Pascal Fontaine, and Cesare Tinelli, “The SMT-LIB Standard: Version 2.5,” Department of Computer Science, The University of Iowa, Technical report, 2015 [Online]. Available: www.SMT-LIB.org. [Accessed: 11-Mar-2021]

[156] A. Stump, G. Sutcliffe, and C. Tinelli, “StarExec: A Cross-Community Infrastructure for Logic Solving,” *Automated Reasoning*. pp. 367–373, 2014 [Online]. Available: http://dx.doi.org/10.1007/978-3-319-08587-6_28

[157] C. Hawblitzel *et al.*, “IronFleet,” *Proceedings of the 25th Symposium on Operating Systems Principles*. 2015 [Online]. Available: http://dx.doi.org/10.1145/2815400.2815428

[158] S. Pernsteiner *et al.*, “Investigating Safety of a Radiotherapy Machine Using System Models with Pluggable Checkers,” *Computer Aided Verification*. pp. 23–41, 2016 [Online]. Available: http://dx.doi.org/10.1007/978-3-319-41540-6_2

[159] V. H. Subburaj and J. E. Urban, “Applying Formal Methods to Specify Security Requirements in Multi–Agent Systems,” *Proceedings of the 2018 Federated Conference on Computer Science and Information Systems*. 2018 [Online]. Available: http://dx.doi.org/10.15439/2018f262

[160] P. Giorgini, F. Massacci, J. Mylopoulos, and N. Zannone, “Modeling security requirements through ownership, permission and delegation,” *13th IEEE International Conference on Requirements Engineering (RE’05)*. 2005 [Online]. Available: http://dx.doi.org/10.1109/re.2005.43

[161] Q. Rouland, B. Hamid, J.-P. Bodeveix, and M. Filali, “A Formal Methods Approach to Security Requirements Specification and Verification,” *2019 24th International Conference on Engineering*

of Complex Computer Systems (ICECCS). 2019 [Online]. Available: http://dx.doi.org/10.1109/iceccs.2019.00033

[162] Z. Zhioua, Y. Roudier, and R. B. Ameur, "Formal Specification and Verification of Security Guidelines," *2017 IEEE 22nd Pacific Rim International Symposium on Dependable Computing (PRDC)*. 2017 [Online]. Available: http://dx.doi.org/10.1109/prdc.2017.51

[163] Z. Zhioua, R. Ameur-Boulifa, and Y. Roudier, "Framework for the Formal Specification and Verification of Security Guidelines," *Advances in Science, Technology and Engineering Systems Journal*, vol. 3, no. 1. pp. 38–48, 2018 [Online]. Available: http://dx.doi.org/10.25046/aj030106

[164] K. Yajima, S. Morimoto, D. Horie, N. S. Azreen, Y. Goto, and J. Cheng, "FORVEST: A Support Tool for Formal Verification of Security Specifications with ISO/IEC 15408," *2009 International Conference on Availability, Reliability and Security*. 2009 [Online]. Available: http://dx.doi.org/10.1109/ares.2009.74

[165] I. Vistbakka, E. Troubitsyna, T. Kuismin, and T. Latvala, "Co-engineering Safety and Security in Industrial Control Systems: A Formal Outlook," *Lecture Notes in Computer Science*. pp. 96–114, 2017 [Online]. Available: http://dx.doi.org/10.1007/978-3-319-65948-0_7

[166] E. Troubitsyna and I. Vistbakka, "Deriving and Formalising Safety and Security Requirements for Control Systems," *Developments in Language Theory*. pp. 107–122, 2018 [Online]. Available: http://dx.doi.org/10.1007/978-3-319-99130-6_8

[167] I. Vistbakka and E. Troubitsyna, "Towards a Formal Approach to Analysing Security of Safety-Critical Systems," *2018 14th European Dependable Computing Conference (EDCC)*. 2018 [Online]. Available: http://dx.doi.org/10.1109/edcc.2018.00040

[168] A. Naumchev, "Object-Oriented Requirements: Reusable, Understandable, Verifiable," *Software Technology: Methods and Tools*. pp. 150–162, 2019 [Online]. Available: http://dx.doi.org/10.1007/978-3-030-29852-4_12

[169] L. Huang and E.-Y. Kang, "Formal Verification of Safety & Security Related Timing Constraints for a Cooperative Automotive System," *Fundamental Approaches to Software Engineering*. pp. 210–227, 2019 [Online]. Available: http://dx.doi.org/10.1007/978-3-030-16722-6_12

[170] M. Farrell *et al.*, "Using Threat Analysis Techniques to Guide Formal Verification: A Case Study of Cooperative Awareness Messages," *Software Engineering and Formal Methods*. pp. 471–490, 2019 [Online]. Available: http://dx.doi.org/10.1007/978-3-030-30446-1_25

[171] D. C. Wardell, R. F. Mills, G. L. Peterson, and M. E. Oxley, "A Method for Revealing and Addressing Security Vulnerabilities in Cyber-physical Systems by Modeling Malicious Agent Interactions with Formal Verification," *Procedia Computer Science*, vol. 95. pp. 24–31, 2016 [Online]. Available: http://dx.doi.org/10.1016/j.procs.2016.09.289

[172] M. Felderer and E. Fourneret, "A systematic classification of security regression testing approaches," *International Journal on Software Tools for Technology Transfer*, vol. 17, no. 3. pp. 305–319, 2015 [Online]. Available: http://dx.doi.org/10.1007/s10009-015-0365-2

[173] "Methods for Testing and Specification (MTS); Security Testing; Basic Terminology," ETSI, TR 101 583 - V1.1.1, Mar. 2015.

[174] N. Mahendra and S. Ahmad, "A Categorized Review on Software Security Testing," *International Journal of Computer Applications*, vol. 154, no. 1. pp. 21–25, 2016 [Online]. Available: http://dx.doi.org/10.5120/ijca2016912023

[175] G. Tian-yang, S. Yin-Sheng, and F. You-yuan, "Research on software security testing," *Proc. World Acad. of Sci. Eng. Technol.*, vol. 70, pp. 647–651, 2010 [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.190.4771&rep=rep1&type=pdf

[176] I. Schieferdecker, J. Grossmann, and M. Schneider, "Model-Based Security Testing," *Electronic Proceedings in Theoretical Computer Science*, vol. 80. pp. 1–12, 2012 [Online]. Available:

http://dx.doi.org/10.4204/eptcs.80.1
[177] G. Fink and M. Bishop, “Property-based testing: a new approach to testing for assurance,” *SIGSOFT Softw. Eng. Notes*, vol. 22, no. 4, pp. 74–80, Jul. 1997, doi: 10.1145/263244.263267. [Online]. Available: https://doi.org/10.1145/263244.263267
[178] J. Zeng, C. Yang, W. Shen, and J. Zhang, “Application of Software Testing Technology in Security and Protection of Power System,” *Advances in Intelligent Systems and Computing*. pp. 1132–1137, 2021 [Online]. Available: http://dx.doi.org/10.1007/978-981-33-4572-0_162
[179] F. Mateo Tudela, J.-R. Bermejo Higuera, J. Bermejo Higuera, J.-A. Sicilia Montalvo, and M. I. Argyros, “On Combining Static, Dynamic and Interactive Analysis Security Testing Tools to Improve OWASP Top Ten Security Vulnerability Detection in Web Applications,” *NATO Adv. Sci. Inst. Ser. E Appl. Sci.*, vol. 10, no. 24, p. 9119, Dec. 2020, doi: 10.3390/app10249119. [Online]. Available: https://www.mdpi.com/2076-3417/10/24/9119. [Accessed: 12-Feb-2021]
[180] S. Ali, “Cybersecurity management for distributed control system: systematic approach,” *Journal of Ambient Intelligence and Humanized Computing*. 2021 [Online]. Available: http://dx.doi.org/10.1007/s12652-020-02775-5
[181] M. Utting, A. Pretschner, and B. Legeard, “A taxonomy of model-based testing approaches,” *Software Testing, Verification and Reliability*, vol. 22, no. 5. pp. 297–312, 2012 [Online]. Available: http://dx.doi.org/10.1002/stvr.456
[182] M. Felderer, B. Agreiter, P. Zech, and R. Breu, “A classification for model-based security testing,” *Advances in System Testing and Validation Lifecycle (VALID 2011)*, pp. 109–114, 2011 [Online]. Available: https://www.researchgate.net/profile/Michael_Felderer/publication/267561985_A_Classification_for_Model-Based_Security_Testing/links/5453b1a40cf2cf51647c20d0.pdf
[183] A. Lunkeit and I. Schieferdecker, “Model-Based Security Testing - Deriving Test Models from Artefacts of Security Engineering,” *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. 2018 [Online]. Available: http://dx.doi.org/10.1109/icstw.2018.00056
[184] H. Stallbaum, A. Metzger, and K. Pohl, “An automated technique for risk-based test case generation and prioritization,” *Proceedings of the 3rd international workshop on Automation of software test - AST ’08*. 2008 [Online]. Available: http://dx.doi.org/10.1145/1370042.1370057
[185] P. Zech, “Risk-Based Security Testing in Cloud Computing Environments,” *2011 Fourth IEEE International Conference on Software Testing, Verification and Validation*. 2011 [Online]. Available: http://dx.doi.org/10.1109/icst.2011.23
[186] M. Felderer, P. Zech, R. Breu, M. Büchler, and A. Pretschner, “Model-based security testing: a taxonomy and systematic classification,” *Software Testing, Verification and Reliability*, vol. 26, no. 2. pp. 119–148, 2016 [Online]. Available: http://dx.doi.org/10.1002/stvr.1580
[187] R. Yang, G. Li, W. C. Lau, K. Zhang, and P. Hu, “Model-based Security Testing: An Empirical Study on OAuth 2.0 Implementations,” in *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, Xi’an, China, 2016, pp. 651–662, doi: 10.1145/2897845.2897874 [Online]. Available: https://doi.org/10.1145/2897845.2897874. [Accessed: 25-Jan-2021]
[188] M. Peroli, F. De Meo, L. Viganò, and D. Guardini, “MobSTer: A model-based security testing framework for web applications,” *Software Testing, Verification and Reliability*, vol. 28, no. 8. p. e1685, 2018 [Online]. Available: http://dx.doi.org/10.1002/stvr.1685
[189] M. Krichen, O. Cheikhrouhou, M. Lahami, R. Alroobaea, and A. J. Maâlej, “Towards a Model-Based Testing Framework for the Security of Internet of Things for Smart City Applications,” *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*. pp. 360–365, 2018 [Online]. Available:

http://dx.doi.org/10.1007/978-3-319-94180-6_34
[190] ARMOUR project, “Deliverable D2.2 Test generation strategies for large-scale IoT security testing - v1,” Aug. 2016.
[191] C. Willcock, T. Deiß, S. Tobies, S. Keil, F. Engler, and S. Schulz, *An Introduction to TTCN-3*. John Wiley & Sons, 2005 [Online]. Available: https://play.google.com/store/books/details?id=cT9dqyhcQ0QC
[192] S. Mahmood, A. Fouillade, H. N. Nguyen, and S. A. Shaikh, “A Model-Based Security Testing Approach for Automotive Over-The-Air Updates,” in *2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, 2020, pp. 6–13, doi: 10.1109/ICSTW50294.2020.00019 [Online]. Available: http://dx.doi.org/10.1109/ICSTW50294.2020.00019
[193] E. dos Santos, E. dos Santos, A. Simpson, and D. Schoop, “A Formal Model to Facilitate Security Testing in Modern Automotive Systems,” *Electronic Proceedings in Theoretical Computer Science*, vol. 271. pp. 95–104, 2018 [Online]. Available: http://dx.doi.org/10.4204/eptcs.271.7
[194] R. M. Parizi, A. Dehghantanha, K.-K. R. Choo, and A. Singh, “Empirical Vulnerability Analysis of Automated Smart Contracts Security Testing on Blockchains,” *arXiv [cs.CR]*, 07-Sep-2018 [Online]. Available: http://arxiv.org/abs/1809.02702
[195] J. Jürjens, “UMLsec: Extending UML for Secure Systems Development,” *≪UML≫ 2002 — The Unified Modeling Language*. pp. 412–425, 2002 [Online]. Available: http://dx.doi.org/10.1007/3-540-45800-x_32
[196] G. Wimmel and J. Jürjens, “Specification-Based Test Generation for Security-Critical Systems Using Mutations,” *Formal Methods and Software Engineering*. pp. 471–482, 2002 [Online]. Available: http://dx.doi.org/10.1007/3-540-36103-0_48
[197] B. Arkin, S. Stender, and G. McGraw, “Software penetration testing,” *IEEE Security Privacy*, vol. 3, no. 1, pp. 84–87, Jan. 2005, doi: 10.1109/MSP.2005.23. [Online]. Available: http://dx.doi.org/10.1109/MSP.2005.23
[198] P. Vats, M. Mandot, and A. Gosain, “A Comprehensive Literature Review of Penetration Testing & Its Applications,” in *8th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*, 2020, pp. 674–680, doi: 10.2139/ssrn.3470687 [Online]. Available: http://dx.doi.org/10.2139/ssrn.3470687
[199] S. Shah and B. M. Mehtre, “An overview of vulnerability assessment and penetration testing techniques,” *Journal of Computer Virology and Hacking Techniques*, vol. 11, no. 1, pp. 27–49, Feb. 2015, doi: 10.1007/s11416-014-0231-x. [Online]. Available: https://doi.org/10.1007/s11416-014-0231-x
[200] A. G. Bacudio, X. Yuan, B.-T. B. Chu, and M. Jones, “An overview of penetration testing,” *International Journal of Network Security & Its Applications*, vol. 3, no. 6, p. 19, 2011 [Online]. Available: https://search.proquest.com/openview/911a51c6546eb7400e083f17edca89c9/1.pdf?pq-origsite=gscholar&cbl=646392
[201] M. Denis, C. Zena, and T. Hayajneh, “Penetration testing: Concepts, attack methods, and defense strategies,” in *2016 IEEE Long Island Systems, Applications and Technology Conference (LISAT)*, 2016, pp. 1–6, doi: 10.1109/LISAT.2016.7494156 [Online]. Available: http://dx.doi.org/10.1109/LISAT.2016.7494156
[202] H. M. Z. A. Shebli and B. D. Beheshti, “A study on penetration testing process and tools,” in *2018 IEEE Long Island Systems, Applications and Technology Conference (LISAT)*, 2018, pp. 1–7, doi: 10.1109/LISAT.2018.8378035 [Online]. Available: http://dx.doi.org/10.1109/LISAT.2018.8378035

[203] A. S. Al-Ahmad, H. Kahtan, F. Hujainah, and H. A. Jalab, “Systematic Literature Review on Penetration Testing for Mobile Cloud Computing Applications,” *IEEE Access*, vol. 7. pp. 173524–173540, 2019 [Online]. Available: http://dx.doi.org/10.1109/access.2019.2956770

[204] D. R. McKinnel, T. Dargahi, A. Dehghantanha, and K.-K. R. Choo, “A systematic literature review and meta-analysis on artificial intelligence in penetration testing and vulnerability assessment,” *Computers & Electrical Engineering*, vol. 75. pp. 175–188, 2019 [Online]. Available: http://dx.doi.org/10.1016/j.compeleceng.2019.02.022

[205] Y. Stefinko, A. Piskozub, and R. Banakh, “Manual and automated penetration testing. Benefits and drawbacks. Modern tendency,” in *2016 13th International Conference on Modern Problems of Radio Engineering, Telecommunications and Computer Science (TCSET)*, 2016, pp. 488–491, doi: 10.1109/TCSET.2016.7452095 [Online]. Available: http://dx.doi.org/10.1109/TCSET.2016.7452095

[206] Xue Qiu, Shuguang Wang, Qiong Jia, Chunhe Xia, and Qingxin Xia, “An automated method of penetration testing,” in *2014 IEEE Computers, Communications and IT Applications Conference*, 2014, pp. 211–216, doi: 10.1109/ComComAp.2014.7017198 [Online]. Available: http://dx.doi.org/10.1109/ComComAp.2014.7017198

[207] S. Shah and B. M. Mehtre, “An automated approach to Vulnerability Assessment and Penetration Testing using Net-Nirikshak 1.0,” in *2014 IEEE International Conference on Advanced Communications, Control and Computing Technologies*, 2014, pp. 707–712, doi: 10.1109/ICACCCT.2014.7019182 [Online]. Available: http://dx.doi.org/10.1109/ICACCCT.2014.7019182

[208] S. Shah and B. M. Mehtre, “A modern approach to cyber security analysis using vulnerability assessment and penetration testing,” *Int J Electron Commun Comput Eng*, vol. 4, no. 6, pp. 47–52, 2013 [Online]. Available: https://www.ijecce.org/Download/conference/NCRTCST-2/11NCRTCST-13018.pdf

[209] J. N. Goel and B. M. Mehtre, “Vulnerability Assessment & Penetration Testing as a Cyber Defence Technology,” *Procedia Comput. Sci.*, vol. 57, pp. 710–715, Jan. 2015, doi: 10.1016/j.procs.2015.07.458. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050915019870

[210] N. Antunes and M. Vieira, “Enhancing Penetration Testing with Attack Signatures and Interface Monitoring for the Detection of Injection Vulnerabilities in Web Services,” in *2011 IEEE International Conference on Services Computing*, 2011, pp. 104–111, doi: 10.1109/SCC.2011.67 [Online]. Available: http://dx.doi.org/10.1109/SCC.2011.67

[211] C. Mainka, J. Somorovsky, and J. Schwenk, “Penetration Testing Tool for Web Services Security,” in *2012 IEEE Eighth World Congress on Services*, 2012, pp. 163–170, doi: 10.1109/SERVICES.2012.7 [Online]. Available: http://dx.doi.org/10.1109/SERVICES.2012.7

[212] B. Stepien, L. Peyton, and P. Xiong, “Using TTCN-3 as a modeling language for web penetration testing,” in *2012 IEEE International Conference on Industrial Technology*, 2012, pp. 674–681, doi: 10.1109/ICIT.2012.6210016 [Online]. Available: http://dx.doi.org/10.1109/ICIT.2012.6210016

[213] A. Falkenberg, C. Mainka, J. Somorovsky, and J. Schwenk, “A New Approach towards DoS Penetration Testing on Web Services,” in *2013 IEEE 20th International Conference on Web Services*, 2013, pp. 491–498, doi: 10.1109/ICWS.2013.72 [Online]. Available: http://dx.doi.org/10.1109/ICWS.2013.72

[214] R. Vibhandik and A. K. Bose, “Vulnerability assessment of web applications - a testing approach,” *2015 Forth International Conference on e-Technologies and Networks for Development (ICeND)*. 2015 [Online]. Available: http://dx.doi.org/10.1109/icend.2015.7328531

[215] S. Nagpure and S. Kurkure, “Vulnerability Assessment and Penetration Testing of Web Application,” in *2017 International Conference on Computing, Communication, Control and*

Automation (ICCUBEA), 2017, pp. 1–6, doi: 10.1109/ICCUBEA.2017.8463920 [Online]. Available: http://dx.doi.org/10.1109/ICCUBEA.2017.8463920

[216] G. Chu and A. Lisitsa, "Penetration Testing for Internet of Things and Its Automation," in *2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, 2018, pp. 1479–1484, doi: 10.1109/HPCC/SmartCity/DSS.2018.00244 [Online]. Available: http://dx.doi.org/10.1109/HPCC/SmartCity/DSS.2018.00244

[217] D. Dalalana Bertoglio and A. F. Zorzo, "Overview and open issues on penetration test," *J. Braz. Comput. Soc.*, vol. 23, no. 1, p. 2, Feb. 2017, doi: 10.1186/s13173-017-0051-1. [Online]. Available: https://doi.org/10.1186/s13173-017-0051-1

[218] J. Bau, E. Bursztein, D. Gupta, and J. Mitchell, "State of the Art: Automated Black-Box Web Application Vulnerability Testing," *2010 IEEE Symposium on Security and Privacy*. 2010 [Online]. Available: http://dx.doi.org/10.1109/sp.2010.27

[219] M. Büchler, J. Oudinet, and A. Pretschner, "Security Mutants for Property-Based Testing," *Tests and Proofs*. pp. 69–77, 2011 [Online]. Available: http://dx.doi.org/10.1007/978-3-642-21768-5_6

[220] F. Dadeau, P.-C. Héam, and R. Kheddam, "Mutation-Based Test Generation from Security Protocols in HLPSL," *2011 Fourth IEEE International Conference on Software Testing, Verification and Validation*. 2011 [Online]. Available: http://dx.doi.org/10.1109/icst.2011.42

[221] F. Siavashi, D. Truscan, and J. Vain, "Vulnerability Assessment of Web Services with Model-Based Mutation Testing," *2018 IEEE International Conference on Software Quality, Reliability and Security (QRS)*. 2018 [Online]. Available: http://dx.doi.org/10.1109/qrs.2018.00043

[222] M. Papadakis, M. Kintis, J. Zhang, Y. Jia, Y. Le Traon, and M. Harman, "Mutation Testing Advances: An Analysis and Survey," *Advances in Computers*. pp. 275–378, 2019 [Online]. Available: http://dx.doi.org/10.1016/bs.adcom.2018.03.015

[223] M. Büchler, J. Oudinet, and A. Pretschner, "Semi-Automatic Security Testing of Web Applications from a Secure Model," in *2012 IEEE Sixth International Conference on Software Security and Reliability*, 2012, pp. 253–262, doi: 10.1109/SERE.2012.38 [Online]. Available: http://dx.doi.org/10.1109/SERE.2012.38

[224] C. Henard, M. Papadakis, G. Perrouin, J. Klein, and Y. Le Traon, "Assessing Software Product Line Testing Via Model-Based Mutation: An Application to Similarity Testing," *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops*. 2013 [Online]. Available: http://dx.doi.org/10.1109/icstw.2013.30

[225] H. Liang, X. Pei, X. Jia, W. Shen, and J. Zhang, "Fuzzing: State of the Art," *IEEE Trans. Reliab.*, vol. 67, no. 3, pp. 1199–1218, Sep. 2018, doi: 10.1109/TR.2018.2834476. [Online]. Available: http://dx.doi.org/10.1109/TR.2018.2834476

[226] I. Schieferdecker, "Model-Based Fuzz Testing," in *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*, 2012, pp. 814–814, doi: 10.1109/ICST.2012.180 [Online]. Available: http://dx.doi.org/10.1109/ICST.2012.180

[227] M. Schneider, J. Großmann, N. Tcholtchev, I. Schieferdecker, and A. Pietschker, "Behavioral Fuzzing Operators for UML Sequence Diagrams," in *System Analysis and Modeling: Theory and Practice*, 2013, pp. 88–104, doi: 10.1007/978-3-642-36757-1_6 [Online]. Available: http://dx.doi.org/10.1007/978-3-642-36757-1_6

[228] M. Schneider, J. Großmann, I. Schieferdecker, and A. Pietschker, "Online Model-Based Behavioral Fuzzing," in *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops*, 2013, pp. 469–475, doi: 10.1109/ICSTW.2013.61 [Online]. Available: http://dx.doi.org/10.1109/ICSTW.2013.61

[229] F. Duchene, S. Rawat, J.-L. Richier, and R. Groz, “KameleonFuzz: evolutionary fuzzing for black-box XSS detection,” in *Proceedings of the 4th ACM conference on Data and application security and privacy*, San Antonio, Texas, USA, 2014, pp. 37–48, doi: 10.1145/2557547.2557550 [Online]. Available: https://doi.org/10.1145/2557547.2557550. [Accessed: 02-Feb-2021]

[230] J. M. Zhang, M. Harman, L. Ma, and Y. Liu, “Machine Learning Testing: Survey, Landscapes and Horizons,” *IEEE Trans. Software Eng.*, pp. 1–1, 2020, doi: 10.1109/TSE.2019.2962027. [Online]. Available: http://dx.doi.org/10.1109/TSE.2019.2962027

[231] A. Odena, C. Olsson, D. Andersen, and I. Goodfellow, “Tensorfuzz: Debugging neural networks with coverage-guided fuzzing,” in *International Conference on Machine Learning*, 2019, pp. 4901–4911 [Online]. Available: http://proceedings.mlr.press/v97/odena19a.html

[232] J. Guo, Y. Jiang, Y. Zhao, Q. Chen, and J. Sun, “DLFuzz: differential fuzzing testing of deep learning systems,” in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, Lake Buena Vista, FL, USA, 2018, pp. 739–743, doi: 10.1145/3236024.3264835 [Online]. Available: https://doi.org/10.1145/3236024.3264835. [Accessed: 03-Feb-2021]

[233] P. Godefroid, H. Peleg, and R. Singh, “Learn&fuzz: Machine learning for input fuzzing,” in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2017, pp. 50–59 [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8115618/

[234] T. Wang, T. Wei, G. Gu, and W. Zou, “TaintScope: A checksum-aware directed fuzzing tool for automatic software vulnerability detection,” in *2010 IEEE Symposium on Security and Privacy*, Oakland, CA, USA, 2010, doi: 10.1109/sp.2010.37 [Online]. Available: http://ieeexplore.ieee.org/document/5504701/

[235] V. Ganesh, T. Leek, and M. Rinard, “Taint-based directed whitebox fuzzing,” in *2009 IEEE 31st International Conference on Software Engineering*, 2009, pp. 474–484, doi: 10.1109/ICSE.2009.5070546 [Online]. Available: http://dx.doi.org/10.1109/ICSE.2009.5070546

[236] T. L. Munea, H. Lim, and T. Shon, “Network protocol fuzz testing for information systems and applications: a survey and taxonomy,” *Multimedia Tools and Applications*, vol. 75, no. 22. pp. 14745–14757, 2016 [Online]. Available: http://dx.doi.org/10.1007/s11042-015-2763-6

[237] T. Zhang, Y. Jiang, R. Guo, X. Zheng, and H. Lu, “A Survey of Hybrid Fuzzing based on Symbolic Execution,” *Proceedings of the 2020 International Conference on Cyberspace Innovation of Advanced Technologies*. 2020 [Online]. Available: http://dx.doi.org/10.1145/3444370.3444570

[238] H. Huang, P. Yao, R. Wu, Q. Shi, and C. Zhang, “Pangolin: Incremental Hybrid Fuzzing with Polyhedral Path Abstraction,” in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 1613–1627, doi: 10.1109/SP40000.2020.00063 [Online]. Available: http://dx.doi.org/10.1109/SP40000.2020.00063

[239] P. Godefroid, M. Y. Levin, and D. Molnar, “AASAGE: whitebox fuzzing for security testing,” *Commun. ACM*, vol. 55, no. 3, pp. 40–44, 2012.

[240] P. Godefroid, M. Y. Levin, and D. Molnar, “SAGE: Whitebox Fuzzing for Security Testing,” *Queue*, vol. 10, no. 1. pp. 20–27, 2012 [Online]. Available: http://dx.doi.org/10.1145/2090147.2094081

[241] B. Ghimis, M. Paduraru, and A. Stefanescu, “RIVER 2.0: an open-source testing framework using AI techniques,” in *Proceedings of the 1st ACM SIGSOFT International Workshop on Languages and Tools for Next-Generation Testing*, Virtual, USA, 2020, pp. 13–18, doi: 10.1145/3416504.3424335 [Online]. Available: https://doi.org/10.1145/3416504.3424335. [Accessed: 05-Feb-2021]

[242] C. Paduraru, M.-C. Melemciuc, and A. Stefanescu, “A distributed implementation using apache spark of a genetic algorithm applied to test data generation,” *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. 2017 [Online]. Available:

http://dx.doi.org/10.1145/3067695.3084219
[243] G. Grieco, M. Ceresa, and P. Buiras, “QuickFuzz: an automatic random fuzzer for common file formats,” *Proceedings of the 9th International Symposium on Haskell*. 2016 [Online]. Available: http://dx.doi.org/10.1145/2976002.2976017
[244] V. J. M. Manes *et al.*, “The Art, Science, and Engineering of Fuzzing: A Survey,” *IEEE Transactions on Software Engineering*. pp. 1–1, 2019 [Online]. Available: http://dx.doi.org/10.1109/tse.2019.2946563
[245] V. Atlidakis, P. Godefroid, and M. Polishchuk, “RESTler: Stateful REST API Fuzzing,” *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. 2019 [Online]. Available: http://dx.doi.org/10.1109/icse.2019.00083
[246] P. McMinn, “Search-based software test data generation: a survey,” *Software Testing, Verification and Reliability*, vol. 14, no. 2. pp. 105–156, 2004 [Online]. Available: http://dx.doi.org/10.1002/stvr.294
[247] P. McMinn, M. Shahbaz, and M. Stevenson, “Search-Based Test Input Generation for String Data Types Using the Results of Web Queries,” *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*. 2012 [Online]. Available: http://dx.doi.org/10.1109/icst.2012.94
[248] G. Antoniol, “Keynote Paper: Search Based Software Testing for Software Security: Breaking Code to Make it Safer,” *2009 International Conference on Software Testing, Verification, and Validation Workshops*. 2009 [Online]. Available: http://dx.doi.org/10.1109/icstw.2009.12
[249] A. Avancini and M. Ceccato, “Security Testing of Web Applications: A Search-Based Approach for Cross-Site Scripting Vulnerabilities,” *2011 IEEE 11th International Working Conference on Source Code Analysis and Manipulation*. 2011 [Online]. Available: http://dx.doi.org/10.1109/scam.2011.7
[250] J. Thomé, A. Gorla, and A. Zeller, “Search-based security testing of web applications,” *Proceedings of the 7th International Workshop on Search-Based Software Testing - SBST 2014*. 2014 [Online]. Available: http://dx.doi.org/10.1145/2593833.2593835
[251] N. Alshahwan and M. Harman, “Automated web application testing using search based software engineering,” *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*. 2011 [Online]. Available: http://dx.doi.org/10.1109/ase.2011.6100082
[252] B. Korel, “Automated software test data generation,” *IEEE Transactions on Software Engineering*, vol. 16, no. 8. pp. 870–879, 1990 [Online]. Available: http://dx.doi.org/10.1109/32.57624
[253] M. Liu, K. Li, and T. Chen, “Security testing of web applications,” *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. 2019 [Online]. Available: http://dx.doi.org/10.1145/3319619.3322026
[254] S. Jan, C. D. Nguyen, A. Arcuri, and L. Briand, “A Search-Based Testing Approach for XML Injection Vulnerabilities in Web Applications,” *2017 IEEE International Conference on Software Testing, Verification and Validation (ICST)*. 2017 [Online]. Available: http://dx.doi.org/10.1109/icst.2017.39
[255] F. C. M. Souza, M. Papadakis, Y. Le Traon, and M. E. Delamaro, “Strong mutation-based test data generation using hill climbing,” *Proceedings of the 9th International Workshop on Search-Based Software Testing - SBST ’16*. 2016 [Online]. Available: http://dx.doi.org/10.1145/2897010.2897012
[256] K. Lakhotia, M. Harman, and H. Gross, “AUSTIN: A Tool for Search Based Software Testing for the C Language and Its Evaluation on Deployed Automotive Systems,” *2nd International Symposium on Search Based Software Engineering*. 2010 [Online]. Available:

http://dx.doi.org/10.1109/ssbse.2010.21
[257] M. E. Delamaro, J. C. Maldonado, and A. M. R. Vincenzi, “Proteum/IM 2.0: An Integrated Mutation Testing Environment,” *Mutation Testing for the New Century*. pp. 91–101, 2001 [Online]. Available: http://dx.doi.org/10.1007/978-1-4757-5939-6_17
[258] M. D. Penta, M. Di Penta, G. Canfora, G. Esposito, V. Mazza, and M. Bruno, “Search-based testing of service level agreements,” *Proceedings of the 9th annual conference on Genetic and evolutionary computation - GECCO ’07*. 2007 [Online]. Available: http://dx.doi.org/10.1145/1276958.1277174
[259] J. Wegener, A. Baresel, and H. Sthamer, “Evolutionary test environment for automatic structural testing,” *Information and Software Technology*, vol. 43, no. 14. pp. 841–854, 2001 [Online]. Available: http://dx.doi.org/10.1016/s0950-5849(01)00190-2
[260] W. E. Wong, W. Eric Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa, “A Survey on Software Fault Localization,” *IEEE Transactions on Software Engineering*, vol. 42, no. 8. pp. 707–740, 2016 [Online]. Available: http://dx.doi.org/10.1109/tse.2016.2521368
[261] P. Agarwal and A. P. Agrawal, “Fault-localization techniques for software systems,” *ACM SIGSOFT Software Engineering Notes*, vol. 39, no. 5. pp. 1–8, 2014 [Online]. Available: http://dx.doi.org/10.1145/2659118.2659125
[262] A. Zakari, S. P. Lee, R. Abreu, B. H. Ahmed, and R. A. Rasheed, “Multiple fault localization of software programs: A systematic literature review,” *Information and Software Technology*, vol. 124. p. 106312, 2020 [Online]. Available: http://dx.doi.org/10.1016/j.infsof.2020.106312
[263] W. Masri, “Automated Fault Localization,” *Advances in Computers*. pp. 103–156, 2015 [Online]. Available: http://dx.doi.org/10.1016/bs.adcom.2015.05.001
[264] D. Zou, J. Liang, Y. Xiong, M. D. Ernst, and L. Zhang, “An Empirical Study of Fault Localization Families and Their Combinations,” *IEEE Transactions on Software Engineering*. pp. 1–1, 2019 [Online]. Available: http://dx.doi.org/10.1109/tse.2019.2892102
[265] Z. Jiang, “Fault Localization of Concurrency Bugs and Its Application in Web Security,” 2014, pp. 618–630, doi: 10.1007/978-3-319-11194-0_55 [Online]. Available: https://link.springer.com/chapter/10.1007%2F978-3-319-11194-0_55
[266] D. E. Simos, K. Kleine, L. S. G. Ghandehari, B. Garn, and Y. Lei, “A Combinatorial Approach to Analyzing Cross-Site Scripting (XSS) Vulnerabilities in Web Application Security Testing,” *Testing Software and Systems*. pp. 70–85, 2016 [Online]. Available: http://dx.doi.org/10.1007/978-3-319-47443-4_5
[267] L. S. Ghandehari, J. Chandrasekaran, Y. Lei, R. Kacker, and D. R. Kuhn, “BEN: A combinatorial testing-based fault localization tool,” in *2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, 2015, pp. 1–4, doi: 10.1109/ICSTW.2015.7107446 [Online]. Available: http://dx.doi.org/10.1109/ICSTW.2015.7107446
[268] M. Ji, S. Huang, and Z. Hui, “Spectrum-based Security Bug Localization by Analyzing Error Propagation,” *International Journal of Performability Engineering*, vol. 16, no. 8, 2020 [Online]. Available: https://qrs20.techconf.org/download/QRS-IJPE/11_Spectrum-based%20Security%20Bug%20Localization%20by%20Analyzing%20Error%20Propagation.pdf
[269] J. D. DeMott, R. J. Enbody, and W. F. Punch, “Systematic bug finding and fault localization enhanced with input data tracking,” *Comput. Secur.*, vol. 32, pp. 130–157, Feb. 2013, doi: 10.1016/j.cose.2012.09.015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S016740481200168X
[270] J. A. Jones and M. J. Harrold, “Empirical evaluation of the tarantula automatic fault-localization

technique," *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering - ASE '05*. 2005 [Online]. Available: http://dx.doi.org/10.1145/1101908.1101949

[271] H. Lu, R. Gao, S. Huang, and W. E. Wong, "Spectrum-Base Fault Localization by Exploiting the Failure Path," in *2016 International Computer Symposium (ICS)*, 2016, pp. 252–257, doi: 10.1109/ICS.2016.0058 [Online]. Available: http://dx.doi.org/10.1109/ICS.2016.0058

[272] R. Abreu, P. Zoeteweij, R. Golsteijn, and A. J. C. van Gemund, "A practical evaluation of spectrum-based fault localization," *Journal of Systems and Software*, vol. 82, no. 11. pp. 1780–1792, 2009 [Online]. Available: http://dx.doi.org/10.1016/j.jss.2009.06.035

[273] W. E. Wong, V. Debroy, and D. Xu, "Towards Better Fault Localization: A Crosstab-Based Statistical Approach," *IEEE Trans. Syst. Man Cybern. C Appl. Rev.*, vol. 42, no. 3, pp. 378–396, May 2012, doi: 10.1109/TSMCC.2011.2118751. [Online]. Available: http://dx.doi.org/10.1109/TSMCC.2011.2118751

[274] W. E. Wong, V. Debroy, R. Gao, and Y. Li, "The DStar Method for Effective Software Fault Localization," *IEEE Trans. Reliab.*, vol. 63, no. 1, pp. 290–308, Mar. 2014, doi: 10.1109/TR.2013.2285319. [Online]. Available: http://dx.doi.org/10.1109/TR.2013.2285319

[275] J. Hwang, T. Xie, F. Chen, and A. X. Liu, "Fault Localization for Firewall Policies," *2009 28th IEEE International Symposium on Reliable Distributed Systems*. 2009 [Online]. Available: http://dx.doi.org/10.1109/srds.2009.38

[276] M. Hilton, N. Nelson, T. Tunnell, D. Marinov, and D. Dig, "Trade-offs in continuous integration: assurance, security, and flexibility," *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2017*. 2017 [Online]. Available: http://dx.doi.org/10.1145/3106237.3106270

[277] T. Mårtensson, D. Ståhl, and J. Bosch, "Continuous Integration Applied to Software-Intensive Embedded Systems – Problems and Experiences," *Product-Focused Software Process Improvement*. pp. 448–457, 2016 [Online]. Available: http://dx.doi.org/10.1007/978-3-319-49094-6_30

[278] J. Bird, *DevOpsSec: Securing Software Through Continuous Delivery*. 2016 [Online]. Available: https://books.google.com/books/about/DevOpsSec.html?hl=&id=5v25AQAACAAJ

[279] T. Laukkarinen, K. Kuusinen, and T. Mikkonen, "DevOps in Regulated Software Development: Case Medical Devices," *2017 IEEE/ACM 39th International Conference on Software Engineering: New Ideas and Emerging Technologies Results Track (ICSE-NIER)*. 2017 [Online]. Available: http://dx.doi.org/10.1109/icse-nier.2017.20

[280] F. G. de O. Neto, F. G. de Oliveira Neto, A. Ahmad, O. Leifler, K. Sandahl, and E. Enoiu, "Improving continuous integration with similarity-based test case selection," *Proceedings of the 13th International Workshop on Automation of Software Test - AST '18*. 2018 [Online]. Available: http://dx.doi.org/10.1145/3194733.3194744

[281] International Electrotechnical Commission, "IEC 62443 Security for Industrial Automation and Control Systems," Geneva, CH, IEC 62443, 2018.