sensors

Review

Static Analysis of Information Systems for IoT Cyber Security: A Survey of Machine Learning Approaches

Igor Kotenko [1,*], Konstantin Izrailov [1,2] and Mikhail Buinevich [3]

[1] Computer Security Problems Laboratory, St. Petersburg Federal Research Center of the Russian Academy of Sciences, 199178 Saint-Petersburg, Russia; konstantin.izrailov@mail.ru
[2] Department of Secure Communication Systems, The Bonch-Bruevich Saint-Petersburg State University of Telecommunications, 193232 Saint-Petersburg, Russia
[3] Department of Applied Mathematics and Information Technologies, Saint-Petersburg University of State Fire Service of EMERCOM of Russia, 196105 Saint-Petersburg, Russia; bmv1958@yandex.ru
* Correspondence: ivkote@comsec.spb.ru

Abstract: Ensuring security for modern IoT systems requires the use of complex methods to analyze their software. One of the most in-demand methods that has repeatedly been proven to be effective is static analysis. However, the progressive complication of the connections in IoT systems, the increase in their scale, and the heterogeneity of elements requires the automation and intellectualization of manual experts' work. A hypothesis to this end is posed that assumes the applicability of machine-learning solutions for IoT system static analysis. A scheme of this research, which is aimed at confirming the hypothesis and reflecting the ontology of the study, is given. The main contributions to the work are as follows: systematization of static analysis stages for IoT systems and decisions of machine-learning problems in the form of formalized models; review of the entire subject area publications with analysis of the results; confirmation of the machine-learning instrumentaries applicability for each static analysis stage; and the proposal of an intelligent framework concept for the static analysis of IoT systems. The novelty of the results obtained is a consideration of the entire process of static analysis (from the beginning of IoT system research to the final delivery of the results), consideration of each stage from the entirely given set of machine-learning solutions perspective, as well as formalization of the stages and solutions in the form of "Form and Content" data transformations.

Keywords: IoT systems; cyber security; static analysis; machine learning; analytic model; survey model; formalization

check for updates

Citation: Kotenko, I.; Izrailov, K.; Buinevich, M. Static Analysis of Information Systems for IoT Cyber Security: A Survey of Machine Learning Approaches. *Sensors* **2022**, *22*, 1335. https://doi.org/10.3390/s22041335

Academic Editors: Giuseppe Maria Luigi Sarne, Jianhua Ma, Domenico Rosaci, Gautam Srivastava and Alessandra Rizzardi

Received: 19 December 2021
Accepted: 5 February 2022
Published: 10 February 2022

1. Introduction

A topical problem in today's world is the lack of IoT system (IoTS) security against the actions of intruders. IoTS security is significantly affected by software, where vulnerabilities lead to violations of confidentiality, integrity, and the availability of information [1,2]. While previously only individual files or software components were investigated to counter such breaches [3], currently a key requirement is the static (SA) and dynamic (DA) analysis of the entire information system.

IoTS consists of a great amount of data, documents and software of various types and purposes. Almost every one of these components can be a source of threats: executable code (e.g., exe files) may contain program bookmarks; Web-server code (e.g., PHP files) may contain backdoors; a hidden channel for leaking confidential and malicious information may be organized by means of stego attachments in images (e.g., jpeg files), and so on [4].

IoTS analysis is directly related to the study of its objects from different perspectives to gain new knowledge about the information security (IS) state of the system [5,6]. A great number of IoTS files, large volume of information in the files, great heterogeneity of information, container structure of documents, etc. do not allow for expert security analysis to be produced in an adequate time.

Classical automation with strict rules does not give the required results, because most of the manual work is spent on creating and debugging such rules. Therefore, an interesting and promising solution is the use of machine learning (ML), which features the transfer of human intellectual activity (clearly, resource-intensive) to artificial (software-implementable and therefore less resource-intensive) intelligence.

The application of DA is more justified for separate programs, which can be repeatedly executed in the same conditions. Full coverage of the code in the investigation process will require using all possible scenarios of the program functioning. As a result it will be possible to obtain error reports, execution logs or program results for further analysis (manually or automatically).

For large IoTS, however, this approach will be a significant problem, since the system components are in a constantly changing environment (which is extremely difficult to replace with a virtual one), and covering even part of their program code may require enormous time resources. Thus, the authors consider static analysis to be the most promising for the development of analysis applicable to modern IoTS.

Based on the above, we propose a hypothesis of the current research: ML is applicable to SA of IoTS systems for information security. This paper is devoted to its confirmation. The article proposes to limit the consideration of the SA task only, and the DA task is considered by the authors to be the direction of future research.

Despite the relative novelty of ML applications in the IS field, they have already shown relevance in this matter: for finding vulnerabilities in source [7] and machine code [8,9], attack detection in networks [10], predicting balances on components of large distributed systems [11,12], anomaly detection in real-time data [13], etc. Since a direct search for vulnerabilities is only one of IoTS analysis' subtasks (albeit most frequently reflected in scientific publications), it is necessary to consider an application of ML on the full cycle of system investigation, not just for executable files.

1.1. Novelty

The novelty of the current review is as follows. First, the predominant majority of existing reviews from the software IS field are devoted to various methods of analyzing software code (source, machine, and byte code) built on the basis of ML. For example, work [8] analyzes binary code exclusively, work [7] analyzes source code, and work [9] analyzes any of these code representations. While code analysis is a central part of the search for vulnerabilities in IoTS software, the processes of data collection and preparation that precede it, as well as the subsequent visualization of the results [14], are also essential parts of a full IoTS SA.

From this point of view, the current review differs from the previous ones in that it covers the entire SA process—from the beginning of investigation of an "unfamiliar" IoTS to the moment of providing all the results of work in the form required for further processing (human or automatic). Therefore, the novelty of the current work lies in the uniform study of the application of machine learning to the entire SA process, and not only to one of its stages.

Secondly, the existing reviews mainly aim at highlighting the entire variety of ways to solve a problem (typically, code analysis) from the point of view of many ML methods and models mentioned in the articles. Thus, papers [7–9] mention the ML problems considered in the current review as well, with various degrees of exhaustiveness; however, no complete systematization on such problems has been made.

The current review initially defines a certain set of problems (and, accordingly, their solutions by methods from the ML field) and performs the selection and description of articles based on their applicability for SA. From this position, the current review uses the task-oriented classification of ML methods, thus, carrying out a certain systematization of the subject area objects. Consequently, in the current work, for the first time, all possibilities (both theoretical and practical) of applying the entire variety of solutions in the field of ML at each stage of SA are considered.

Thirdly, in addition to the verbal description of SA, it is not only divided into stages, but, for the first time, it is proposed to represent their actions in a formalized form—as transformations of Form and Content of the studied data in IoTS. In contrast to this approach, the paper [8] only records a formalized form of the considered specific solutions, the paper [7] gives some formulas from the ML field, and the paper [9] does not contain any formulas at all.

As a consequence, using this mathematical apparatus, a formalized presentation of detailed SA techniques for a particular IoTS (taking into account the features of its elements, having non-linear and more complex conduct schemes, as well as having the possibility of automatic verification for correctness and optimization) is possible in the future. Consequently, a new mathematical apparatus for the formalization of SA stages is proposed.

Thus, there are no works on the topic of the article (SA with ML of IoTS), and the taxonomy of close surveys is inapplicable. The taxonomy proposed in the article is aimed at systematizing all SA stages and ML tasks for IoTS. Thus, the whole set of already existing or only proposed methods is covered.

1.2. Contributions

The main contribution of the current review to the theory is the systematization of the basic elements of the SA (stages) and ML (task solutions) fields, in the form of two models: the SA model and the ML application model for SA. Thus, these two models cover, from different positions, the subject area delineated by the SA stages from the position of the basic tasks and ML methods. The models have a matrix form, in which the columns represent the stages and the rows define the tasks. The SA model can be used to formally evaluate the performance and efficiency (as well as other characteristics) of the algorithms used to implement a given SA step with ML solutions.

The contribution of the current review to the practice is the substantiated possibility to create an intelligent framework, which contains a whole set of elements for the construction of methods of nonlinear variations of SA for complex IoTS, while being amenable to complete automation. Thus, in the simplest case, these elements of the Framework can correspond to the cells of the specified matrix models, and with further development, have more detail. The possibility of practical application for almost any of the ML solutions in any of the SA stages is substantiated. The SA model can serve as a theoretical basis for practical implementations of the corresponding algorithms.

1.3. Article Content

The article has the following content. Section 3 reviews publications with independent reviews of articles on the IoTS SA process produced using ML methods. The methodological scheme of the study is constructed, reflecting also its ontology. In Section 4, the main stages of SA are defined, and their formal representation is given in the form of transformations performed on the Form and Content of the IoTS data under study. Then, typical tasks solved with ML are considered, which are also written in a formalized form.

At the end of the chapter, the steps and problem solutions are systematized into a single matrix, the cells of which specify the application of one of the ML field solutions to the operation of one of the SA steps. This matrix corresponds to the first result of the study—the SA model.

Section 5 reviews articles with research results applicable to the performance of a stage of SA, and describes the ML solutions used in doing so. Section 6 systematizes the articles in the form of a matrix model—according to their relation to a certain SA stage and one of the tasks solved with the help of ML. This matrix corresponds to the second result of the study—a model of ML application to SA.

A conceptual model of a complex SA using ML, based on a hypothetical intelligent framework is described. Section 7 presents the main conclusions of the study and indicates the way forward.

2. Research Methodology

The specificity of using SA for IoTS based on ML is as follows.

Internet of Things. IoTS security is of particular relevance, since the disruption of their functioning directly affects the real world—the operation of mechanisms, people's lives, etc. For example, errors in the operation of smart home security systems can lead to financial losses for owners, incorrect operation of the transport system [15] can lead to accidents and traffic congestion, and errors in the operation of medical IoT devices can lead to the death of patients.

Statical Analysis. The particular difficulty of conducting SA is manifested precisely for IoTS, which have different functional purposes. Thus, devices use various OS, distros and CPU architectures [16]. Each such choice, in particular, is selected based on the tasks of the devices. For example, some IoTS require high operating speed, others demand battery life, others lack ultra-precise complex calculations, etc. As a result, a more generalized and systematized set of SA methods and instruments is needed across the entire IoTS diversity.

The existing SA methods, based in particular on expert-based rules, are being developed for the some type of Desktop PC software. For example, the well-known Hex-Rays decompiler from IDA Pro supports only code for the x86, x64, ARM32, ARM64, and PowerPC processors. Wherein, threat detection based on suspicious network traffic (i.e., through dynamic analysis) will not always reveal a single attack exploiting a 0-day vulnerability. Thus, a security analysis is required even before the devices are actually launched.

System of IoT. The presence of a whole system of interacting IoTS imposes a number of restrictions on how to ensure security. Thus, in particular, dynamic analysis will be of little use, since it is rather difficult to emulate the environment of a device that continuously interacts with the outside world. At the same time, sometimes, security problems manifest themselves precisely in the case of a group of several interacted IoTS and not a single device.

Machine Learning. The large number of IoT manufacturers and the variety of solutions used do not allow the application of SA rules that are created by hand or are required to use manual labor of experts. Otherwise, SA will be extremely resource intensive. A large number of cyber-physical interfaces (sensors, sources of impact, etc.) significantly complicates the development of a test case manually. As a result, a partial replacement of a person's creative abilities with automation is required. This is what justifies the use of ML.

The first step in verifying applicability of any new set of methods to the tasks at hand should be a review of existing assumptions for such an application. In order to confirm the hypothesis, we formulate an ongoing research problem, compiled from two subtasks:

1. To justify that ML-based solutions can underlie the transformations carried out at each stage of IoTS SA for information security.
2. To make an overview of ML-based solutions, each applicable to each stage of IoTS SA for information security.

The process of solving this hypothesis and its solution is described in this article.

In addition, since the results of the review should give an answer regarding the formulated hypothesis and tasks, the main results of the study will be defined in the following form:

1. The SA model, systematizing in a formalized form the required transformations of IoTS SA stages and possible solutions of ML tasks.
2. A model of ML application for SA, systematizing the existing solutions linking IoTs SA and the possibilities of ML application for its implementation.

The obtained results will be useful both in creating a general methodology of intelligent static analysis and in selecting implementations of its specific solutions. In addition, the second result will assess the status and research trends of existing solutions. The research methodology in schematic form, reflecting also its ontology, is presented in Figure 1.

According to Figure 1, the ongoing study has the following layout, in which the actions are marked with the corresponding numbers:

1. Actualization of the research topic as IS of IoTS.

2. Selection of the subject area of the research, as SA from the ML position.
3. Proposal of the hypothesis for confirmation in the study.
4. Statement of the main research problem.
5. Division the main research problem into subtasks.
6. Analysis of publications with existing reviews in the subject area.
7. Allocation and formalization of the SA stages.
8. Allocation and formalization of tasks to be solved with the help of ML.
9. Systematization of SA stages and ML tasks.
10. Building the SA model.
11. Basic overview of the articles in the subject area.
12. Building an ML application model for SA.
13. Obtaining and analyzing publication metrics in the SA subject area.
14. Creation of the intelligent framework concept for SA.
15. Selection of future research directions.
16. Determination of future research perspectives: conducting a similar review for IoTS DA, development of the mathematical apparatus of ML application for SA, creation of a framework prototype.
17. Carrying out practical experiments with the framework prototype and its analysis.

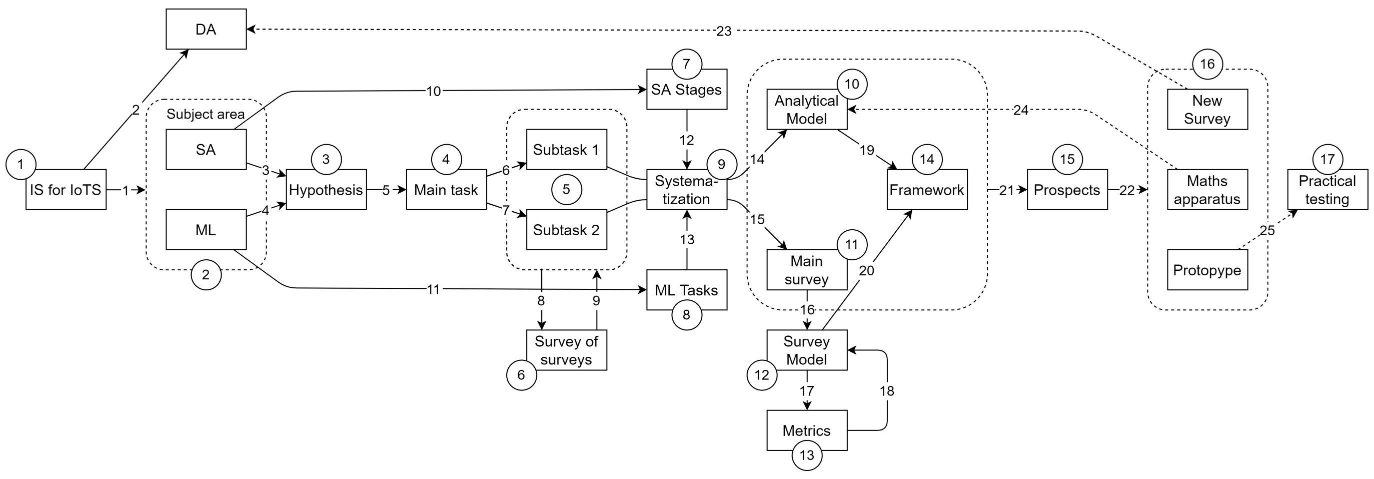

Figure 1. Research scheme.

The ontological part of the scheme in Figure 1 is defined by the elements presented on it (in rectangles) and their relationships, with the following values (according to the line numbers):

1. Has current solution paths.
2. Has solution paths not considered in the work.
3. Is the first basis of the hypothesis.
4. Is the second basis of the hypothesis.
5. Sets the main problem of the study.
6. Divides into subtask 1 of the study.
7. Divides into subtask 2 of the study.
8. Explores existing solutions.
9. Concludes that there are no suitable solutions.
10. SA steps are identified and formalized.
11. ML tasks are identified and formalized.
12. Used as the matrix columns in systematization.
13. Used as the matrix rows in systematization.
14. SA model is constructed.
15. Main review of publications is done.
16. ML application model is built for SA.
17. Publication metrics are calculated.

18. Justifies the ML application model.
19. Constructs the framework basis using the solution of subtask 1.
20. Constructs the framework basis using the solution of subtask 2.
21. Predicts continuation of the study.
22. Chooses directions for future research.
23. Analyzes the application of ML to DA.
24. Development of a mathematical apparatus to refine the model.
25. Development and testing of a smart framework prototype.

Note that the Main survey (action 11, Figure 1) is the collection and analysis of publications for each application of the ML tasks for SA stages (it is described in Section 4). This is necessary to collect statistical data on the research history of each application. Statistical data will reveal the main research trends and their possible problems. The general picture of the elaboration of the entire subject area will be visible. At the same time, a large number of publications will be able to confirm the research hypothesis.

The SA stages model (action 7, Figure 1) is needed to create a new model (action 9, Figure 1)—a model applying ML tasks (action 8, Figure 1) for SA stages. The last model is necessary for theoretical confirmation of the hypothesis. In fact, an attempt to create a methodological apparatus for theoretical confirmation of hypotheses of the form "Is it possible to use certain methods to solve requires problems" has been made.

3. Analysis of Existing Review Works

3.1. ML for SA of IoTs

There are many reviews devoted to collecting, systematizing, and comparing scientific publications on the topic of ML applications for information systems in general and IoTS in particular. However, only a small fraction of them relate to the field of IoTS security, and not, for example, network security. A brief analysis of interesting reviews is given below.

In [8], a comprehensive study of different ways to analyze binary code is presented. A taxonomy of the corresponding framework is given, which consists of four components: feature extraction, feature embedding, analysis techniques, and applications. Greater attention is specifically paid to the application of ML regarding the techniques of code analysis. Thus, there are many works (more than 100) in which ML is applied to the problem of classification, including viruses and their families, string tokens, cryptographic algorithms, and code clones.

Clustering is also described from the following positions, in addition to preparing data for other classifiers: authorship detection, detection of clones, determination of similarity of the program to a virus from the base, and identification of auxiliary features in the code (virus features and entry points in functions). The following works are described concerning the application of ML for binary code analysis: classify malware [17–20], identify function entry points [21], and recognize functions [22].

Work [7] is devoted to the application of ML exclusively to source code for the following tasks: Recommender Systems—help systems for software developers (code autocompletion, etc.); Inferring Coding Conventions—support of coding style conventions (formatting, variable naming, etc.; Code Defects—detection of anomalies in the code, which can tell you about vulnerabilities; Code Translation, Copying, and Clones—code conversion between different programming languages and search for similar code fragments; Code to Text and Text to Code—conversion between the program code and natural language code; Documentation, Traceability and Information Retrieval—code documentation and searching; and Program Synthesis—mechanisms of "smart" program code generation.

The works [23–29] describe concerning the application of ML to analyze the source code for anomalies, i.e., code defects. Thus, the review covers only a narrow set of security subdomains.

In the review [9] on software vulnerability analysis and detection, all reviewed works are divided into four categories (with corresponding papers): Vulnerability Prediction Models based on Software Metrics, Anomaly Detection Approaches, Vulnerable Code

Pattern Recognition, and miscellaneous Approaches. The first category is devoted to prediction of possible vulnerabilities on the basis of different source metrics [30–38] and binary code [37,39], as well as code commits (which goes beyond SA).

The second category is devoted to detecting anomalies in the code, which may be a consequence of the vulnerabilities in it. This is done by looking for deviations in API calls from applicable patterns [40–46] and missing typical checks [45,47,48]. The third category is devoted to the direct detection of vulnerabilities in code by pretrained patterns. For this purpose, the works [49–58] apply classification and in [52,56] also clustering. The fourth category contains descriptions of approaches not included in the previous three categories.

The category mentions the following works and their ideas: fuzzy testing using genetic algorithms [59]; bug report analysis using classification to identify hidden vulnerabilities [60,61]; searching for memory corruption vulnerabilities by source code using genetic algorithms and Fish School Search [62]; classification and prediction of false positives in SA vulnerabilities in Web application code and automatic fixing them [63]; and improving SA efficiency and scalability for software repositories [64]. Thus, although the research in the fourth category is related to SA, it is rather of an auxiliary nature and will not be discussed further.

Work [65] focuses on the security of software interactions in complex information systems through visualization. In particular, to increase the intellectualization of the analysis of interactions, it is proposed to use classification, anomaly detection, clustering, regression, and dimensionality reduction.

As we can clearly see, despite the vast number of works, most of them are applicable only to the source code and have far from always satisfactory efficiency. The review concludes that the field of detection of vulnerabilities in code using ML is insufficiently developed. In spite of the rather extensive reviews (by the number of publications reviewed—from 100 to 200), all the works considered in them can be attributed only to direct software security testing.

However, the processes of collecting such a code, its preparation and presentation to an expert are of a partial nature, not being part of a unified static analysis methodology. As a result, it is not always possible to build a static analysis process out of several coordinated steps. It is necessary to pay attention to these questions, which will be done in this article.

3.2. ML for IoT Security

We also analyze the latest works devoted to other reviews on the topic of using ML to enhance IoT security.

The work [66] is devoted to the problem of identifying IoT devices. The article provides an overview of identification methods based on ML. Identification mechanisms are separated into four groups: pattern recognition, deep learning identification, unsupervised learning identification, and anomalous device detection. Thus, this work is partially related to the current subject area. However, the proposed solutions can only be attributed to SA Stage 1.

In [67], the security trends for IoT devices are assessed. A total of 20 surveys are reviewed, and ML is considered in only 25% of them. The main taxonometries, however, relate to the identification of groups of attacks and not how to defend against them. The use of ML to improve security is mainly considered in the context of identifying the dynamic attacks, and not static weaknesses in the implementation of devices (i.e., in program code, algorithms, architecture, etc.).

The work [16] emphasizes the relevance and underdevelopment of solutions for the search for malicious IoT software. Particular attention is paid to the executed Unux-like programs in the ELF format. The statistics show that the OS for IoT is 70% Linux (file format—ELF), and Windows—22% (file format—PE); the frequency of using other operating systems is less than 20%.

The main fields of the ELF header are considered. A feature of the survey is the consideration of approaches to searching for malicious code from the standpoint of their independence from the architecture of the program execution. A taxonomy of features obtained from ELF programs is created, which can be used for static and dynamic analysis to search for malicious code in IoT devices.

The taxonomy separates features into the following: for SA—metrics, graphs and trees, sequences, and dependencies; for DA—logs and resource usage. Despite some closeness of this survey to the current research, its results can be partially attributed to Stage 3 only (with partial consideration of Stage 1 and Stage 2), i.e., not to the full SA process. At the same time, only such ML problems as classification (and its special case—detection) are highlighted.

In [68], the IoT threats as well as ML-based IoT security models are outlined. A layered IoT model is proposed. A Q work flow of threat detection is described based on this model. However, all countermeasures involve analyzing the network activity of devices at various network levels, i.e., they belong to dynamic analysis. At the same time, there is no direct analysis of IoTS as a software system. From ML techniques, only classification, clustering and regression are indicated. Techniques, such as rule-based systems and reinforcement learning, are highlighted separately.

In [69], the application of Federated Learning (FT) for IoT is considered. As a result of using this technique, there will be no need to transfer data for training. The FL-IoT system is presented, describing local training, FT-server and exchange processes between them. The possibility of using FT also for detecting malicious software in a static way is indicated. The main applicability of ML is in classification, regression, and anomaly detection. The main countermeasures for IoT threats are through dynamic device analysis. At the same time, the survey is devoted specifically to the implementation of FT in IoT and not to systematize SA. Note that the use of FT can increase efficiency and safety when conducting the SA (at all stages and tasks of ML).

The work [70] is devoted to IoT security in 5G networks, characterized by a huge number of devices and high data transfer rates. The existing authentication schemes at the physical layer are analyzed. Schemes are considered from the perspective of applying ML. ML technologies, such as autoregressive random process, Kalman filtering prediction, reinforcement learning (specifically Q-learning and Dyna-Q), AdaBoost classifier, and kernel machine, are investigated.

The work [71] emphasizes the relevance of IoTS security due to their wireless transmission network, large coverage area, heterogeneity of services provided, and cyber-physicality. The application of ML methods for creating attack detection models is analyzed. It mainly describes the application of deep-learning-based classification.

The work [72] focuses on securing the IoT using ML. Four types of defenses are considered: device authentication, DoS and DDoS attacks defense, intrusion detection, and malware detection (relevant in the context of current research). The possibilities of using ML based on the following models are indicated: convolutional neural networks, SVM, BP neural networks, and ensemble learning.

The work [73] provides an overview of using honeypots to collect information about malware that attacks PE files, data streams, and IoT. A feature of the considered honeypots is the use of ML for data collection and analysis. Based on a “raw” review of articles in the IEEE Xplore and ACM databases, it is concluded that there is an increasing trend in the use of such smart honeypots. However, there is no specificity regarding the details of the use of ML.

The work [74] provides an overview of ways to detect malware for Android systems (including those in the IoT field) using ML. Experiments were carried out using the Naive Bayes, J48, Random Forest, Multiclass Classifier, and Multi class perceptron algorithms. The main application of ML here is to solve a classification problem.

Based on this review of other surveys on the application of ML for IoT security, the following conclusions can be drawn.

First, the considered solutions mainly belong to the field of dynamic analysis, since they process external effects of malicious devices during their operation. In contrast, our review focuses to SA. Such an analysis can be carried out both without the direct launch of IoTS, and only at a certain point in time.

Second, the solutions reviewed mainly seek to introduce taxonometry for systematizing devices, offering countermeasures based in part on ML. Our review introduces the systematization of all SA stages using basic ML tasks. We investigate not a problem of identifying threats, but a problem of identifying ways to counter threats during SA.

Third, the considered solutions mainly conducted systematization according to existing threats and solutions. This can lead to the fact that hypothetically some of the elements will be lost (for example, if such solutions have not yet been proposed). We initially synthesize the entire set of solutions—as a 4×5 matrix (or most of it). Then, we prove that every such decision has a right to exist. Thus, our approach is more methodologically correct.

Fourth, none of the reviews considered the entire set of main tasks solved with the help of ML—classification, anomaly detection, regression, clustering, and generalization. As a rule, IoT security applies only the solution of the first two problems. SA of a binary code is done basically only with the use of classifiers. Thus, new applications of more rare solutions in ML for the IoT field can be "missed".

Fifth, the considered solutions are mainly aimed at studying individual IoT devices or their interactions. We position the review as an analysis of the whole IoTS. In the interests of this, the concept of Intelligent Framework is proposed.

4. SA Model

To create models of the subject area, we consider in more detail the SA area from the position of application of ML in it. To do this, we use both known theoretical preconditions and practical experience in the published studies. First, the whole SA is heterogeneous (and often not linear), which means that it can be divided into certain time segments or stages (with a certain purpose). Secondly, the solutions must use ML. Consequently, it is possible to distinguish the tasks for which ML is applicable. As a result, it is possible to systematize in a single model both the stages of SA and the ML solutions applicable to them.

For the stages, classical and generalized time segments were used (excluding, for example, balancing and augmentation).

The purpose of the Model is as follows. First, its formation itself formally confirms the possibility of applying various solutions to ML tasks at each stage of SA. At the same time, such a proof will be built on a single basis (using Form and Content, described below). Thus, the module serves as a theoretical proof of the Hypothesis. Secondly, the theoretical model can be used for practical implementations of algorithms for solving problems for stages.

4.1. Stages of Static Analysis

Since SA is a complex process consisting of an entire set of tasks to be solved and using qualitatively different data, it is usually divided into several stages. For convenience, we use the following formalization to assign each of the activities to the stages of SA. Any data is represented by the information it contains—Content (C), and appearance—Form (F). For example, a line-by-line log of execution errors consists of: Content—errors and Form—text strings.

Then, certain data, both stored in the IoTS and transformed in the SA process, can be written as a tuple $< F \mid C >$. From this position, we consider each stage as a Content and Form transformation of incoming data. Then, the formalized version of the transformations of the stages can be written as:

$$< F_{i+t} \mid C_{i+1} >= Stage_i(< F_i \mid C_i >), \tag{1}$$

where F_i and F_{i+1} is Form before and after i-th Stage, C_i, and C_{i+1} is Content before and after i-th Stage, $Stage_i()$ is a transformation of i-th Stage. Summarizing the described approaches and tools for SA software [75–78], we can combine them based on the method of data transformation. With a sufficient degree of abstraction, without violating the correctness, we can distinguish the following four stages of IoTS analysis.

4.1.1. Stage 1. Data Collection

This is the initial stage and is intended to isolate from the IoTS that subsystem ("vertical component") or level ("horizontal component"), which will be further processed. This step is necessary because the IoTS, like any complex and multifaceted object, can be considered in terms of different aspects. A specific aspect must be chosen at this stage. The input of the stage is the information about the IoTS itself. The output of the stage receives only part of the information about the IoTS selected for processing.

The formal notation of Stage 1 is as follows:

$$\begin{cases} < F_2 \mid C_2 >= Stage_1(< F_1 \mid C_1 >) \\ F_2 = F_1 \\ \mid C_2 \mid < \mid C_1 \mid \end{cases}, \quad (2)$$

where operator $\mid X \mid$ is a set power calculation, which together with the third equation means that after Stage 2 the Content size has decreased. This is explained by the fact that at this stage only part of IoTS information (C_2) is selected, and not information about the entire IoTS (C_1). At the same time, the IoTS Data Form does not change at this stage, which is shown in Equation (2).

This step may not be explicitly related to IoTS tasks, although it is necessary. Typical actions for the stage are the following: selecting the direction and level of research; unpacking data containers (archives); typing and attributing files; structuring executable code architectures, etc. For example, at this stage, only executable files (PE format for Windows or ELF format for Linux) will be selected from all IoTS files.

4.1.2. Stage 2. Data Preparation

This stage is designed to convert the data about a part of the IoTS to a form suitable for the processing methods. Without this step, all processing methods would have to adapt to the form of the data in the IoTS, which is naturally not productive. The input of the stage is the IoTS piece of information selected in Stage 1. The output of the stage creates part of the IoTS information converted into a suitable form. Naturally, if there is more than one processing method, this step must convert the IoTS information into a form suitable for each method. Thus, the step may separate the Form and Content population for each method.

The formal notation of Stage 2 is as follows:

$$\begin{cases} < F_3 \mid C_3 >^j = Stage_2^j(< F_2 \mid C_2 >) \\ F_3 \neq F_2 \\ \bigcup\limits_{j=1..N} C_3^j \equiv C_2 \end{cases}, \quad (3)$$

where $< F_3 \mid C_3 >^j$ is the Form and Content after Stage 2 suitable for the j-th processing method; $Stage_2^j()$ is the part of Stage 2 where the data after Stage 1 is converted to the form suitable for each j-th method; and operator "$\equiv$" is identity, which together with the second equation means that, after Stage 3, the total Content has not changed (N is the number of processing methods); $\bigcup\limits_j C_j$—merging of all Content. The second equation is explained by the fact that this stage "tweaked" only individual Form parts of Content, without changing the total information in Content. The fact of the change of the Form in the step is reflected in the second equation.

This phase does not solve specific IS problems but converts the data into a form that best represents the features associated with such problems. Typical actions for this stage are: creating a project and environment; separating IoTS modules; obtaining binary code sections; disassembling; separating subprograms in the code; building an Abstract Syntax Tree, Intermediate Representation Tree, Call Tree, Control Flow, Data Flow, etc. For example, at this stage, for each IoTS file will be built a graph of the execution of instructions of its functions—Control Flow, as well as calls of other functions from them—Call Flow.

4.1.3. Stage 3. Data Processing

This stage is the main and most difficult both from a theoretical and practical point of view. At this stage, all the methods of data processing are performed in relation to IS (vulnerabilities, malwares, stego, bugs, backdoors, etc.). The results of this stage are determined by the combined results of all its methods. At the input of the stage comes the data, prepared for its work with the help of Stage 2. Each method, as a rule, consists of many complex algorithms, often creating qualitatively new data sets. Consequently, at the output of the stage, we can obtain data with completely different Form and Content. The formal notation of Step 3 is as follows (for one method):

$$\begin{cases} < F_4 \mid C_4 >= Stage_3(< F_3 \mid C_3 >) \\ F_4 \neq F_3 \\ C_4 \neq C_3 \end{cases}, \tag{4}$$

where $Stage_3()$ is the part of Stage 3 corresponding to one of the methods for which data were generated in Stage 2. As indicated, the resulting Form and Content can have significant variations, as reflected in Equations (2) and (3).

At this stage, IS tasks are solved directly: vulnerabilities are discovered, virus clones are found, code security metrics are evaluated, code vulnerabilities are neutralized, predictions are made regarding future threats, etc. Typical actions at this stage are: creating and updating an infiltrator model; searching for and predicting vulnerabilities; searching for clones; converting code to a human-oriented form; authorship determination, etc. For example, at the stage by the aggregate of function calls (received by Control Flow and Call Flow) this function can be detected as malicious. The exact match between the function instructions and the malware from the database is not necessary.

4.1.4. Stage 4. Result Formation

Result formation is the final stage, providing all the results of the IoTS analysis. At this stage, the data obtained during processing at Stage 3 are converted into a single form (less often, into their set), ready for further analysis—both human and software. At the input of the stage are the results of all methods of data processing. The output of the stage can be considered problem-oriented, depending on the purpose of the analysis application.

The formal notation of Stage 4 is as follows:

$$\begin{cases} < F_5 \mid C_5 >= Stage_4\left(\bigcup\limits_{j=1..M} < F_4 \mid C_4 >^j\right) \\ F_5 \neq F_4 \\ C_5 = \bigcup\limits_{j=1..M} C_4^j \end{cases}, \tag{5}$$

where $Stage_4()$ is the transformation of Stage 4, taking as input all the set of methods results (by number M). The form of data is likely to change after applying stage (although, theoretically, it can remain the same for trivial problems). The Content after the stage will consist of all the Content obtained by the methods at the previous stage.

4.1.5. Form and Content Transformation

The stage adapts results of IS tasks to the required view given in the specific SA goal. Typical actions for the stage are as follows: systematization of parameters and characteristics of IoTS objects; classification of viruses and vulnerabilities; displaying the list of found viruses and vulnerabilities; displaying security metrics and statistics; selection of IS recommendations, etc. For example, at the stage on the obtained list of malicious functions, the output of this stage may consist of the list itself as infection statistics can. For an easier presentation of stages analyzing the IoTS, we present their Form and Content transformations in Table 1.

Table 1. Form and Content transformations in the process of static analysis of the information system.

Transformations	**Stage 1. Data Collection**	**Stage 2. Data Preparation**	**Stage 3. Data Processing**	**Stage 4. Result Formation**
Form and Content	$< F_2 \mid C_2 >= Stage_1(< F_1 \mid C_1 >)$	$< F_3 \mid C_3 >^j = Stage_2^j(< F_2 \mid C_2 >)$	$< F_4 \mid C_4 >= Stage_3(< F_3 \mid C_3 >)$	$< F_5 \mid C_5 >= Stage_4\left(\bigcup\limits_{j=1..M} < F_4 \mid C_4 >^j\right)$
Form	$F_2 = F_1$	$F_3 \neq F_2$	$F_4 \neq F_3$	$F_5 \neq F_4$
Content	$\mid C_2 \mid < \mid C_1 \mid$	$\bigcup\limits_{j=1..N} C_3^j \equiv C_2$	$C_4 \neq C_3$	$C_5 = \bigcup\limits_{j=1..M} C_4^j$

The formalization from Table 1 will be used implicitly when assigning each of the following activities to the SA stages.

4.2. Tasks Solved by ML

Now, we consider the problems usually solved with ML. According to both general theory and a large number of scientific papers and their reviews ([79,80]), all ML problems can be divided into the following.

Classification is assigning objects to a given class. The application requires teacher-led training regarding the features of each class. One of the most common applications in IS is the division of program code into safe and malware, using the features of the latter.

The formal notation of a solution to the classification problem is as follows:

$$\begin{cases} \{O\} \rightarrow \{O\}^{T_1} ... \{O\}^{T_N} \\ i \neq j : \{O\}^{T_i} \cap \{O\}^{T_j} = \varnothing, \\ Given : \{O^i_{Train} \in \{O\}^{T_k}\} \end{cases} \tag{6}$$

where $\{O\}$ is the set of classified objects; $\{O\}^{T_i}$ is the set of objects belonging only to class T_i (following the second equation of the formula, the intersection of the set of objects of different classes equals an empty set "$\varnothing$"); *Given* is the indication of given solution parameters as a set of predefined training objects O^i_{Train}, assigned to the corresponding classes T_k.

Anomaly detection is the assignment of objects to a new unknown class, thus, distinguishing the task from classification. Applications are possible both with and without training with the teacher—on normal and anomalous data—on deviations from normal data. One of the most frequent applications in the field of IS (besides the typical detection of network attacks) can be the detection of differences in the operation of the program from normal (function calls, disk operation, processor power consumption, etc.), which often indicates the presence of malware in it.

The formal notation of a solution to the anomaly detection problem is as follows:

$$\begin{cases} \{O\} \rightarrow \{O\}^N, \{O\}^A \\ O^i \in \{O\}^N, O^j \in \{O\}^A, O^i \not\approx O^j \\ Given : \varnothing \parallel \{O^i_{Train} \in \{O\}^N\}, \{O^i_{Train} \in \{O\}^A\} \end{cases}, \tag{7}$$

where $\{O\}$ is a set of objects to detect anomalies; $\{O\}^N$ is a set of normal objects; $\{O\}^A$ is a set of anomalous objects significantly different from normal (which is shown by the second equation, the symbol "$\not\approx$" is used for significant difference); *Given* is an indication of given solution parameters as two options (through operator OR—"$\parallel$"): None ($\varnothing$) or a set of pre-known training objects (O^i_{Train} and O^j_{Train}), assigned to normal ($\{O\}^N$) and abnormal

($\{O\}^A$). Thus, the first part of the last equation is true in the case of detection without a teacher, and the second part is true with a teacher.

Regression is the prediction of a continuous value for datasets (as opposed to classification, where a discrete value—a class—is determined). Similarly to classification, the application requires training with a teacher. In IS, it can be used to predict errors in future programs.

The formal notation of a solution to the regression problem is as follows:

$$\begin{cases} \{O\} \to \{D\} \\ D^i = F(O^i) \\ Given : \{F(O^i_{Train}) = D^i\} \end{cases}, \qquad (8)$$

where $\{O\}$ is the set of objects to determine the regression; $\{D\}$ is the set of numbers from the set of real "$\mathbb{R}$", matched to objects $\{O\}$; D^i is i-th number matched to i-th object O^i using some function $F()$; *Given* is the indication of given solution parameters as a set of predefined objects O^i_{Train}, matched (using function $F()$) to the corresponding numbers D^i.

Clustering is the division into groups according to their attributes. Application is possible without a teacher, because the classification looks for internal patterns in the data. Generally, clustering is applied before other classifications. However, in the IS field, it can be applied to search for malware clones in the code of the IoTS objects under study.

The formal notation of a solution to the classification problem is as follows:

$$\begin{cases} \{O\} \to \{O\}^{K_1} ... \{O\}^{K_N} \\ i \neq j : \{O\}^{K_i} \cap \{O\}^{K_j} = \varnothing \\ Given : N \end{cases}, \qquad (9)$$

where $\{O\}$ is the set of clustered objects; $\{O\}^{K_i}$ is the set of objects belonging only to cluster K_i (following the second equation of the formula, the intersection of the objects set from different clusters equals an empty set); and *Given* is an indication of the given solution parameters as their absence ($\varnothing$).

Generalization (dimensionality reduction) is transformation of a feature space of one dimension into a space of smaller dimension with minimal loss of contained information. For reasons similar to clustering, the application of regression is possible with learning without training. Similar to clustering, generalization is applied as preprocessing of data before other classifications. In the IS field, dimensionality reduction can also be used for auxiliary tasks related to threat mapping and selection of protection recommendations.

The formal notation of a solution to the generalization problem is as follows:

$$\begin{cases} \{O \sim X\} \to \{O \sim Y\} \\ X^i = \{x^i_k\}^{k=1...N} \\ Y^i = \{y^i_k\}^{k=1...M} \\ N > M \\ Given : \varnothing \end{cases}, \qquad (10)$$

where $\{O\}$ is a set of objects described (using operator "$\sim$") by feature set $\{X\}$ of dimension N; $\{Y\}$ is a set of features of objects with less dimension M than the original; x^i_k and y^i_k is a value of individual k-th feature for i-th object; and *Given* is an indication of given solution parameters as number of clusters, by which the original objects will be distributed.

4.3. SA Model Representation

The main stages of SA (Section 3.1) and the main tasks to be solved in ML (Section 3.2) were specified earlier. Furthermore, a formalized description of stages and tasks was given. Thus, the previously established hypothesis regarding the application of ML to SA can be transformed to verify that each of the stages of SA can be based on the solution of one of the ML tasks.

We represent this assumption in the form of a matrix model, where the columns are SA stages, the rows are ML tasks, and the cells are a formalized record of stage actions based on the solution of one of the ML tasks. The possibility of filling all the cells of the matrix of such a model will mean a theoretical confirmation of the hypothesis. The analytical model of ML solutions for the SA stages is presented in Table 2.

Table 2. Machine-learning decision matrix for static analysis stages.

Tasks	**Stage 1. Data Collection**	**Stage 2. Data Preparation**	**Stage 3. Data Processing**	**Stage 4. Result Formation**
Task 1. Classification	$\{C\} \rightarrow$ $\bigcup\limits_{T1}\{C\}^{T1}, 0 \times \bigcup\limits_{T2}\{C\}^{T2}$ $T1 + T2 = T$	$\{F\} \rightarrow$ $\bigcup\limits_{T1}\{F\}^{T1}, 0 \times \bigcup\limits_{T2}\{F\}^{T2}$ $T1 + T2 = T$	$\{C\} \rightarrow$ $\bigcup\limits_{T1_C}\{C\}^{T1_C}, 0 \times \bigcup\limits_{T2_C}\{C\}^{T2_C}$ $T1_C + T2_C = T_C$ $\{F\} \rightarrow$ $\bigcup\limits_{T1_F}\{F\}^{T1_F}, 0 \times \bigcup\limits_{T2_F}\{F\}^{T2_F}$ $T1_F + T2_F = T_F$	$\{F\} \rightarrow$ $\bigcup\limits_{T1}\{F\}^{T1}, 0 \times \bigcup\limits_{T2}\{F\}^{T2}$ $T1 + T2 = T$
Task 2. Anomaly detection	$\{C\} \rightarrow \{C\}^{A}, 0 \times \{C\}^{N}$	$\{F\} \rightarrow \{F\}^{A}, 0 \times \{F\}^{N}$	$\{C\} \rightarrow \{C\}^{A}, 0 \times \{C\}^{N}$ $\{F\} \rightarrow \{F\}^{A}, 0 \times \{F\}^{N}$	$\{F\} \rightarrow \{F\}^{A}, 0 \times \{F\}^{N}$
Task 3. Regression	$\{C\} \rightarrow \{D\}^{T1}, 0 \times \{D\}^{T2}$ T1+T2=T	$\{F\} \rightarrow \{F\}^{T1}, 0 \times \{F\}^{T2}$	$\{C\} \rightarrow$ $\{D_C\}^{T1_C}, 0 \times \{D_C\}^{T2_C}$ $T1_C + T2_C = T_C$ $\{F\} \rightarrow$ $\{D_F\}^{T1_C}, 0 \times \{D_F\}^{T2_C}$ $T1_F + T2_F = T_F$	$\{F\} \rightarrow \{F\}^{T1}, 0 \times \{F\}^{T2}$ $T1 + T2 = T$
Task 4. Clustering	$\{C\} \rightarrow$ $\bigcup\limits_{K1}\{C\}^{K1}, 0 \times \bigcup\limits_{K2}\{C\}^{K2}$ $K1 + K2 = K$	$\{F\} \rightarrow$ $\bigcup\limits_{K1}\{F\}^{K1}, 0 \times \bigcup\limits_{K2}\{F\}^{K2}$ $K1 + K2 = K$	$\{C\} \rightarrow$ $\bigcup\limits_{K1_C}\{C\}^{K1_C}, 0 \times \bigcup\limits_{K2_C}\{C\}^{K2_C}$ $K1_C + K2_C = K_C$ $\{F\} \rightarrow$ $\bigcup\limits_{K1_F}\{F\}^{K1_F}, 0 \times \bigcup\limits_{K2_F}\{F\}^{K2_F}$ $K1_F + K2_F = K_F$	$\{F\} \rightarrow$ $\bigcup\limits_{K1}\{F\}^{K1}, 0 \times \bigcup\limits_{K2}\{F\}^{K2}$
Task 5. Generalization	$\{C \sim X\} \rightarrow$ $\{C \sim Y\}^{R1}, \{C \sim Y\}^{R2}$ $R1 + R2 = R$	$\{F \sim X\} \rightarrow$ $\{F \sim Y\}^{R1}, \{F \sim Y\}^{R2}$ $R1 + R2 = R$	$\{C \sim X_C\} \rightarrow$ $\{C \sim Y_C\}^{R1_F}, \{C \sim Y_C\}^{R2_C}$ $R1_C + R2_C = R_C$ $\{F \sim X_F\} \rightarrow$ $\{F \sim Y_F\}^{R1_F}, \{F \sim Y_F\}^{R2_F}$ $R1_F + R2_F = R_F$	$\{F \sim X\} \rightarrow$ $\{F \sim Y\}^{R1}, \{F \sim Y\}^{R2}$ $R1 + R2 = R$

Note. In the table, the expression with multiplication by zero “0 x” means that in the process of applying the solution from the ML field data were obtained, which are not used to this end of SA at this stage. Thus, some classes $\{C\}$, regression numbers $\{D\}$, clusters $\{O\}^{K}$ and reduced feature dimensions $\{O \sim Z\}$ are divided into two sets (with indices T1 and T2 and a common T, with indices K1 and K2 and a common K, and with indices R1 and R2 and a common R), the second of which is not used in the interest of SA.

Each cell of Table 2 represents an ML statement for the stage operation, taking a tuple from Form and Content input data (obtained from the IoTS or previous) and returning the same tuple from the output data (produced by the current stage). To simplify the record from the formal entries of ML problem solutions, we use only the first line describing the main action, thereby, omitting the rest, indicating the conditions on the input and output variables.

There will be preceding and succeeding non-ML statements in addition to the above-mentioned ML statements, since in addition to the intelligent component, each step performs strictly defined rules (e.g., unpacking archives with files, ranking documents by their size, searching for malicious code areas in databases with signatures, translating tabular data into the form of graphs, etc.).

Expert analysis of the matrix (see Table 2) allows us to conclude that the cells of the matrix are written correctly. Hence, each of the ML problem solutions can be the basis for transformations of each of the ML stages. Thus, the first subtask can be considered solved, which confirms the main hypothesis of the current scientific research from the theoretical point of view.

When conducting an expert analysis, 10 specialists were selected, meeting the following requirements:

- PhD for more than 5 years,
- acquaintance and application in practice of system analysis,
- knowledge of the basics of machine learning,
- experience in the field of the IoT security,
- analytical modeling skills,
- lack of explicit affiliation with the authors of the current investigation, and
- general competence in the subject area.

Then, each selected expert was introduced to the specifics and conducting SA Steps as well as with the goal to create an analytical model of CA using ML. Each expert was presented with an analytical model in matrix form to study (see Table 2). The expert conducted an individual analysis of the resulting analytical model. Then, a survey was conducted with the experts regarding each cell of the table ($4\ Stages \times 5\ Tasks = 20\ cells$) on the following five questions:

1. Do you understand the meaning of the entry?
2. Is the formal notation correct?
3. Does the entry match this SA Stage (column heading)?
4. Does the record correspond to the given ML Task (row header)?
5. To what extent the receipt of the record of the current stage logically follows from the record of the previous stage (except for Stage 1)?

The experts evaluated the model (i.e., answers to questions) using a point system, from 1—the answer is completely negative, to 5—the answer is completely positive.

Then, the answers for each cell of the table were summarized and averaged. The distribution of average responses across all cells ranged from 4.5 to 5 points. This result confirms the correctness of the obtained analytical model.

5. Systematization of SA Stages and ML Solutions

Based on the task, we present the model that systematizes the existing research in the form of the following table: columns are represented by stages of IoTS analysis; rows—tasks solved using ML. Next, we consider the research papers applicable to IoTS analysis. We assign each work by its attributes to both one (or several) stages and one (or several) ML tasks. Thus, the entire set of such studies should hypothetically allow us to fill all the cells of the table.

We introduce the notation work groups located in cells as follows—Sx_Ty, where x is the number of analysis stage ($x = 1...4$), y is the number of ML task ($y = 1...5$). For example, if a scientific paper outlines the process of searching for vulnerabilities in code (Stage 3) through classification (Task 1), then it belongs to the group S3_T1.

A large number of papers were considered for the following reasons.

First, the presence of more than a single study on the application of the solution of each ML problem at each stage of the SA will be considered more reliable evidence of the hypothesis.

Secondly, it is necessary to determine the presence and degree of research of each stage of the SA from the standpoint of solving each task of the MO. This will allow one to identify the most unaffected areas (compared to the rest) and pay special attention to them.

Thirdly, the statistical distribution of the characteristics of work (for example, the stage of SA, the task of MO, bringing to the experiment) over the years will make it possible to predict the success of each affected area.

It is most difficult to divide into groups S2_T5 and S3_T1, since data is often prepared by downsizing before the typical classification. We further divide the works into these groups based on whether the data after downgrading can be used in isolation (group S2_T5) or whether they are for training only (group S3_T1). Additionally, we group all the works considered further by the SA stages.

5.1. Stage 1. Data Collection

The authors of this paper proposed and tested the typing of basic information systems files, including IoTS (with the extensions .exe, .py, .doc, .png, .txt, .c, and .cpp) [81], and binary code processor architectures (amd64, arm64, armel, armhf, i386, mips, mips64el, mipsel, ppc64el, and s390x) [82–84], using ML classifiers. These works belong to the S1_T1 group.

Similarly to papers [81–85] applied an SVM (Support Vector Machine) to classify files by blocks in the file system, which may be needed in forensics. The work belongs to the S1_T1 group.

An earlier work [86] solved the problem of preprocessing data files without any description, that are the result of events such as System Crash, data interception by secret services, storing information in "impersonal" files to prevent leaks, etc. For this purpose, similarities between files are sought by creating clusters of their attributes. In [87], a classical application of clustering for grouping text documents using the TF-IDF (Term Frequency—Inverse Document Frequency) scheme is described.

This approach can be applied to the initial collection of documents. All of them will have common features and, therefore, will be investigated by similar methods. Similarly, the work [88] discusses clustering algorithms (K-means, K-medoids, Single Link, Complete Link, and Average Link) for preparing files for forensic processing by experts of the corresponding field. The work belongs to the S1_T4 group.

In [89], a method for searching and localizing signatures and machine-printed text in images using classification is described. MIL (Multiple Instance Learning) is used for this purpose. The method is not directly related to IS but can be used for document indexing in forensic and business fields (as in [90]). The work belongs to the group S1_T1. In [91], MIL is used for clustering. Thus, the work can be classified as S1_T4.

In [92], a classical method for identifying packed executable files (PE format) using the SVM classifier is described. This makes it possible to define objects to be first unpacked and then processed. The work belongs to the S1_T1 group.

In [93], a method for identifying documents containing packed executables is described. In contrast to [92], an anomaly detection method is used for this purpose. Thus, the work belongs to the S1_T2 group.

The work [94] indicates the need to identify anomalies in file integration systems. The proposed solution is based on the principles of self-learning. The work belongs to the S1_T2 group.

In [95], the applied problem of file optimization is solved by predicting the lifetime of files by absolute path symbols. By lifetime, we mean the interval between file creation and the last reading. Regression methods based on Random Forest and Convolutional Neural Network Model are used for prediction. The method can be used to sort files by lifetime and select for processing those that are within a given range. This can be used in forensics to establish files related to the chronology of cybercrime. Thus, the work belongs to the S1_T3 group.

In [96], the classical problem of stego attachment detection is solved. For this purpose, a universal solution based on Multiple Linear Regression is proposed. Additionally, the length of the nested message is obtained. The images found in this way can be subjected to additional processing using the revealed information. Thus, the work belongs to the S1_T3 group and, to some extent, to the S2_T3 group as well (due to the partial localization of the stego field).

In [97], the classical method of document classification based on SVM with prior dimensionality reduction using PCA (Principal Component Analysis) is described. The work belongs to the S1_T1 group.

In [98], a method to recognize handwritten inscriptions in graphical images is investigated. A convolutional and recurrent neural network is used for this purpose. The inscriptions may contain sensitive information (e.g., passwords or personal data) and, therefore, their detection and verification can be applied in the interest of IS. In addition to the fact that a handwritten inscription is present, its translation into a specific text is performed and hence the work can be classified as S1_T1 and S2_T1 groups.

The work [99] deals with enhancing activities in forensics. For this purpose, it is proposed to use machine learning in terms of classification and clustering to process documents and cell phone applications. The authors propose to apply SVM and kNN for this purpose. Thus, the work belongs to groups S1_T1 and S1_T4.

5.2. Stage 2. Data Preparation

According to the reviewed review [8], the works [21] and [22] can be classified as S2_T1.

In [100], the secondary problem of decompilation is solved—determination of variable types. For this purpose, it is proposed to use the SVM and Random Free classifiers, which showed better results in comparison with others. The work belongs to the group S2_T1.

In [101], an ML-based classification method is described, which allows to determine Function Entry Points (the starting byte of each function) with high accuracy. This is necessary for disassembling function instructions in the interest of further analysis. The work belongs to the S2_T1 group.

The review [102] discusses the evolution of architecture reconstruction methods. It contains discussion and references to works which use clustering for reconstruction of program architecture. In the future, the obtained architecture can be processed for the detection of high-level vulnerabilities. The work belongs to the S2_T4 group.

The work [103] solves the problem of software refactoring. It is proposed to apply the HASP clustering algorithm to group software classes into packages.

Work [104] is devoted to a new approach for recovering binary malware code running on embedded devices considered by Sykipot on Smart cards. This is done by collecting data from the side channels concerning the power consumption of the device. PCA and LDA (Linear Discriminant Analysis) methods are used to reduce the dimensionality of the feature space and then kNN (K-Nearest Neighbors) classification is applied. It is shown that this technique can be used in practice, not only in theory. This work is similar to that described in [105]. The work can equally be attributed to the S2_T5 and S3_T5 groups.

In [106], a sequence of executable binary instructions is converted into a grayscale image. Dimensionality reduction by LDA is then applied, both to reduce the image and to obtain a more optimal training sample. Thus, the work belongs to the groups adjacent with respect to Stage 3: S2_T5 and S4_T5.

In [107], a method built on a logistic-regression classifier that predicts row and column separators in tabular data is proposed. This task can be common when preparing data in files for processing by the methods in Stage 3. Thus, the work belongs to the S2_T3 group.

In [108], a method based on deep neural networks is described that aims at analyzing log text. The anomalies detected in the text will carry the most useful information about application startup failures. In the interest of SA, this can be used to isolate large data files of the most suspicious information: for example, the failure of critical services in the IS. The application of the method will help form more substantial results in Stage 4. Thus, the work belongs to groups S2_T2 and S4_T2.

Similar to [108], the article [109] describes the experiments to identify anomalies in the system logs of OpenStack. For this, a SVM with different cores is used. The work belongs to the S4_T2 group.

The previously described work [96] belongs to group S2_T3, and work [98] belongs to group S2_T1.

5.3. Stage 3. Data Processing

According to the examined review [8], the following works can be classified as S3_T1: [17–20] since they detect vulnerabilities by means of classification. We also mention a work [58] that applies regression to predict vulnerabilities in new Test Cases on existing ones for one set of programs. The work can be classified as S3_T3.

According to review [7], the works [23–29] detect anomalies in the code and belong to group S3_T2.

According to review [9] works [49–58] belong to group S3_T1, works [40–48] are S3_T2, works [30–39] are S3_T3 and works [52,56] are S3_T4.

The work [110] is devoted to the detection of malicious code in software using machine learning. Taxonometry is introduced, highlighting such process steps as the presentation of the file with the code, the identification of signs and the direct classification of the malicious code. At the last step, classifiers, such as Artificial Neural Networks, Bayesian Network, Naive Bayes, Decision Trees, kNN, Boosted Algorithms, SVM, Voting Feature Intervals, and OneR are applied. The work belongs to the S3_T1 group.

In [111], a text categorization approach is applied to n-grams of binary program code. Various classification methods are then applied to categorize the sample as safe or malicious software. The work belongs to the group S3_T1.

In [112], we describe the application of classification based on a neural network (with layers: input, embedding, bidirectional long-term memory, attention, and output) trained on the binary code of known vulnerabilities (from NVD and CWE databases). The application of the method to real VLC and LibTIFF software, often used in IoT devices, is shown. The work should be classified as S3_T1.

In [113], the developed software HOSTBAD (Host-based Anomaly Detection) for detecting malicious Android applications is described. It solves the task of detecting anomalies with ML using the features: received/sent SMS, received/sent calls, device activity state, and running applications/processes. A solution similar to [113] was proposed in [114] but was based on a DCA (Dendritic Cell Algorithm). Both works belong to the S3_T2 group.

In [115], an approach for malware classification in Android applications is described, the main one being ensembles of methods, including T-SNE (t-Distributed Stochastic Neighbor Embedding). Although T-SNE is mainly used to visualize data by downsizing the space (to 2D or 3D), in this case, it provides significant assistance to other ensemble classifiers: Gradient Boost Decision Tree, k-NN, Extra-Trees, Logistic Regression, and Neural Network. As a result, it provides more accurate detection of malware. Thus, the work belongs to the group S3_T5.

In [116], various binary data feature detection methods (CFsSubset, Principal Components, InfoGainAttribute, Correlation AttributeEval, GainRatioAttribute, and SymmetricalUncertAttribute) based on n-grams are discussed and then used for classification and malware detection. The best results are achieved by applying PCA and SVM. The work can be equally attributed to groups S3_T1 and S3_T5.

For applications requiring user permission, [117] discusses the method of malware detection as follows. First, the permissions dimensionality of the Android application is lowered using PCA. Second, the SVM classifier is applied to detect malware. The work can be attributed equally to groups S3_T1 and S3_T5.

In [118], the possibility of counteracting attacks using ML without the direct usage of files is discussed. For this purpose, Perceptron is used to detect anomalies in the command lines of standard Windows operating system utilities. The work belongs to the S3_T2 group.

In [119], a way to detect malware in PDF (Portable Document Format) files is described. PCA and the artificial neural network are used for this purpose. The work belongs to the group S3_T1.

In [120], it is proposed to visualize malicious Android applications and then classify the obtained images. SVM, KNN, and Random Forest are used for this purpose. The work belongs to the S3_T1 group.

Work [121] is devoted to the security of IoT devices with command line interpreters typical for Linux shells. It is assumed that malicious software, using shell commands, can perform both system hacking and further infection. The proposed solution is the ShellCore software solution based on static code analysis detecting the malware by its use of shell commands. The solution is based on classification, which allows assigning the work to groups S3_T1 and S3_T5.

The previously described works [104,105] belong to the S3_T5 group.

5.4. Stage 4. Result Formation

In [122], an o-glasses method is proposed, which visualizes document files (not necessarily executable) to search for shellcode in them. A high F-measure is claimed (about 99.95 percent) for $\times$86 binary code. A special one-dimensional convolutional neural network (1d-CNN) is used for this purpose. Although the method implements the full cycle of code analysis, it solves the task of malware visualization. Therefore, the work should be classified as S4_T1.

The work [123] deals with two problems related to the detection of malware: (1) the detection of malware signatures from logs (e.g., the xml created when executing an .exe file in a sandbox) for further training of classifiers and (2) the precise detection of groups of mutant malware. To solve the first problem, ensembles of classifiers are proposed. To solve the second problem, clustering is proposed. To visualize the obtained results in 2D space, an algorithm for decreasing the dimensionality of t-SNE (t-distributed Stochastic Neighbor Embedding) space is used. Thus, the two approaches to solving problems, as well as the way to visualize the malware, refer the work to the groups S4_1, S4_4 and S4_5, respectively.

The work [124] addresses the issue of effective interaction between visualization methods and anomaly detection methods (as outlier). For this purpose, the author’s algorithm «hdoutliers», different from special methods (described in many articles [125–127], etc.), is proposed. Thus, the work belongs to the S4_T2 group.

In [128], a method for detecting anomalies in the system log containing both natural language text and numerical values is proposed. This can be applied in the last stage of SA for the data collected by the methods in Stage 3. Therefore, the work belongs to the S4_T2 group.

In [129], a method for automatically classifying vulnerabilities by their textual description using machine learning is described. Such a method can be applied to the detected vulnerabilities at the SA result formatting stage so that they are better presented to the expert, which places this work in the S4_T1 group.

In [130], an attempt is made to predict 0-day vulnerabilities in products based on vulnerabilities contained in NVDs. Linear and quadratic regression models are used for this purpose. Clearly, the method can be applied to predict new IP vulnerabilities based on those found (in Stage 3). Thus, the work belongs to the S4_T3 group.

The work [131] evaluates the influence of the depth of field of the image on the subjective “attractiveness” and image quality. Using logical regression and a deep neural network, it is possible to predict the attractiveness of an image by its quality. A number of experiments were performed. Thus, it is possible to adapt the methods of visualizing the results of SA for further processing. The work belongs to the S4_T3 group.

A solution similar to [129], but which classifies Transport Infrastructure of a Smart City threats, is described in [132]. For this purpose, a partitioning of threats into clusters (based on machine learning without a teacher) is applied, each of which is mapped to a certain class. Therefore, the work belongs to the S4_T4 group.

The previously described work [106] belongs to the S4_T5 group, and [108] belongs to the S4_T2 group.

6. Review Model

A summary of publication reviews (in amount of 85) is presented in four parts (Tables 3–5). Columns of the tables have the following designations:

1. Ref. (short for Reference)—link to the publication.
2. Title—publication title.
3. Year—year of publication.
4. Type—type of publication (conference, journal, workshop, lecture notes, report, or preprint).
5. Stage—publication's affiliation with the corresponding stage of SA.
6. Task—publication's affiliation with the corresponding ML problem's solution.
7. Content—main scientific and practical content of the publication, carrying the following meanings (as completed):
    - Theory—full-fledged theory with possible partial realization of a prototype.
    - Experiment—partially realized prototype with full-fledged experiments.
    - Practice—full-fledged prototype or an entire software product.

We examine information from the Tables 3–5 in detail in the following order.

First, we analyze publication numbers by year, presented as a histogram in Figure 2.

Thus, first, the minimum year of publication is 1997, and the maximum is 2021, while the average year of publication is 2013. At the same time, after 2007, there is a clear tendency for research in this subject area, although no rapid growth (for example, in recent years) is detected. In the context of the current research task, it can be assumed that some saturation of existing solutions has been achieved in the applying ML for IoTS SA. Therefore, new breakthrough ideas are required for its further development.

Secondly, we analyze the number of publications by type, presented as a bar chart in Figure 3.

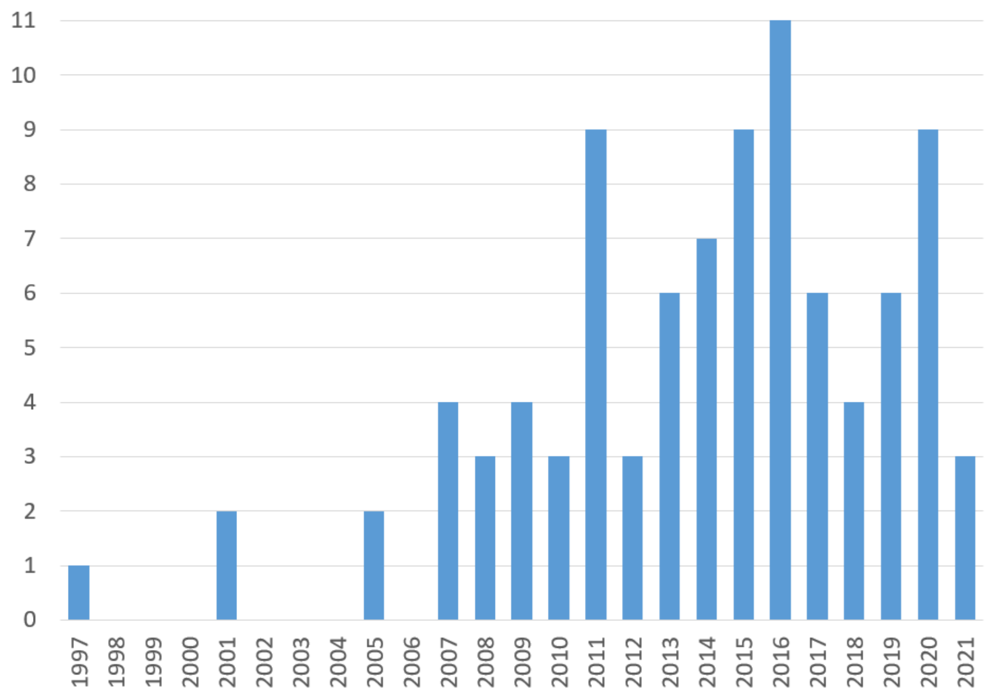

Figure 2. Distribution of publications by year.

Table 3. Publication summary (Part 1).

Ref.	Title	Year	Type	Stage	Task	Content
[58]	Toward Large-scale Vulnerability Discovery Using Machine Learning	2016	Conference	3 3	1 3	Practice
[110]	Detection of malicious code by applying machine-learning classifiers on static features: A state-of-the-art survey	2009	Journal	3	1	Experiment
[111]	Malicious Code Detection Using Active Learning	2008	Conference	3	1	Theory
[100]	Type Learning for Binaries and Its Applications	2019	Conference	2	1	Experiment
[81]	Method for classification of files based on machine-learning technology	2020	Journal	1	1	Practice
[82–84]	Identification of Processor's Architecture of Executable Code Based on Machine Learning	2020	Journal	1	1	Practice
[101]	Machine Learning-Assisted Binary Code Analysis	2007	Workshop	2	1	Theory
[122]	o-glasses: Visualizing x86 Code from Binary Using a 1d-CNN	2020	Conference	4	1	Theory
[112]	Cyber Vulnerability Intelligence for Internet of Things Binary	2020	Conference	3	1	Experiment
[113]	A machine-learning approach to anomaly-based detection on Android platforms	2015	Journal	3	2	Practice
[114]	Android malware detection using the dendritic cell algorithm	2014	Conference	3	2	Experiment
[86]	Similarity detection among data files—a machine-learning approach	1997	Conference	1	4	Theory
[88]	Document Clustering for Forensic Computing: An Approach for Improving Computer Inspection	2011	Conference	1	4	Experiment
[87]	Document Clustering—A Feasible Demonstration with K-means Algorithm	2019	Conference	1	4	Theory
[102]	Evolution in Software Architecture Recovery Techniques—A Survey	2017	Conference	2	4	Theory
[103]	A Hierarchical Clustering-Based Approach for Software Restructuring at the Package Level	2017	Conference	2	4	Practice
[123]	A Novel Solutions for Malicious Code Detection and Family Clustering Based on Machine Learning	2019	Conference	4 4 4	1 4 5	Theory
[115]	Android malware detection using 3-level ensemble	2016	Conference	3	5	Experiment
[104]	Reverse engineering smart card malware using side channel analysis with machine-learning techniques	2016	Conference	2 3	5 5	Theory
[116]	Feature selection and machine-learning classification for malware detection	2015	Journal	3 3	1 5	Theory
[117]	Android malware detection based on permissions	2014	Conference	3 3	1 5	Practice
[106]	Android ransomware detection using reduced opcode sequence and image similarity	2017	Conference	2 4	5 5	Experiment
[85]	File Block Classification by Support Vector Machine	2011	Conference	1	1	Theory
[118]	Preventing File-Less Attacks with Machine Learning Techniques	2019	Conference	3	2	Theory
[89]	Document Image Classification and Labeling using Multiple Instance Learning	2011	Conference	1	1	Experiment
[90]	Multi-scale Structural Saliency for Signature Detection	2007	Conference	1	1	Practice
[91]	Multi-instance clustering with applications to multi-instance prediction	2009	Journal	1	4	Experiment
[92]	Detection of packed executables using support vector machines	2011	Conference	1	1	Practice
[93]	Detecting Packed Executable File: Supervised or Anomaly Detection Method?	2016	Conference	1	2	Experiment
[94]	An anomaly detection system proposal to ensure information security for file integrations	2018	Conference	1	2	Theory
[124]	Visualizing Big Data Outliers through Distributed Aggregation	2018	Conference	4	2	Theory

Table 4. Publication summary (Part 2).

Ref.	Title	Year	Type	Stage	Task	Content
[128]	Relational Synthesis of Text and Numeric Data for Anomaly Detection on Computing System Logs	2016	Conference	4	2	Practice
[95]	Predicting File Lifetimes with Machine Learning	2019	Lecture Notes	1	3	Theory
[107]	A Machine-Learning Approach to Automatic Detection of Delimiters in Tabular Data Files	2016	Conference	2	3	Theory
[96]	Multiple linear regression for universal steganalysis of images	2018	Conference	1 2	3 3	Experiment
[108]	Log File Anomaly Detection	2016	Report	2 4	2 2	Theory
[109]	Experimentations with OpenStack System Logs and Support Vector Machine for an Anomaly Detection Model in a Private Cloud Infrastructure	2020	Conference	2	2	Experiment
[130]	Forecasting Zero-Day Vulnerabilities	2016	Conference	4	3	Practice
[131]	The Effects of Depth of Field on Subjective Evaluation of Aesthetic Appeal and Image Quality of Photographs	2020	Journal	4	3	Experiment
[97]	Text Document Classification with PCA and One-Class SVM	2017	Conference	1	1	Theory
[119]	Machine Learning With Feature Selection Using Principal Component Analysis for Malware Detection—A Case Study	2019	Journal	3	1	Theory
[105]	Power-based Side-Channel Instruction-level Disassembler	2018	Conference	2 3	5 5	Practice
[17]	Data mining methods for detection of new malicious executables	2001	Conference	3	1	Practice
[18]	Integrated static and dynamic analysis for malware detection	2015	Journal	3	1	Experiment
[19]	Classification of malware families based on N-grams sequential pattern features	2013	Conference	3	1	Experiment
[20]	Malware detection using machine learning	2009	Conference	3	1	Practice
[21]	Byteweight: Learning to recognize functions in binary code	2014	Conference	2	1	Practice
[22]	Recognizing functions in binaries with neural networks	2015	Conference	2	1	Theory
[23]	Automatically learning semantic features for defect prediction	2016	Conference	3	2	Theory
[24]	Emergent, crowd-scale programming practice in the IDE	2014	Conference	3	2	Practice
[25]	Using web corpus statistics for program analysis	2014	Conference	3	2	Practice
[26]	Bugram: bug detection with n-gram language models	2016	Conference	3	2	Practice
[27]	Finding Likely Errors with Bayesian Specifications	2017	Preprint	3	2	Practice
[28]	Learning to Represent Programs with Graphs	2018	Conference	3	2	Theory
[29]	Deep Learning to Find Bugs	2017	Journal	3	2	Practice
[30]	Strengthening the empirical analysis of the relationship between linus’ law and software security	2010	Conference	3	3	Theory
[31]	An empirical study of the evolution of PHP web application security	2011	Conference	3	3	Theory
[32]	Can traditional fault prediction models be used for vulnerability prediction?	2013	Journal	3	3	Theory
[33]	An initial study on the use of execution complexity metrics as indicators of software vulnerabilities	2011	Conference	3	3	Theory
[34]	Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities	2011	Journal	3	3	Theory
[35]	Using complexity metrics to improve software security	2013	Journal	3	3	Theory
[36]	Predicting vulnerable components: Software metrics vs text mining	2014	Conference	3	3	Theory

Table 5. Publication summary (Part 3).

Ref.	Title	Year	Type	Stage	Task	Content
[37]	Challenges with applying vulnerability prediction models	2015	Conference	3	3	Theory
[38]	To fear or not to fear that is the question: Code characteristics of a vulnerable function with an existing exploit	2016	Conference	3	3	Theory
[39]	Searching for a needle in a haystack: Predicting security vulnerabilities for windows vista	2010	Conference	3	3	Theory
[40]	Bugs as deviant behavior: A general approach to inferring errors in systems code	2001	Conference	3	2	Practice
[41]	DynaMine: Finding common error patterns by mining software revision histories	2005	Conference	3	2	Practice
[42]	PR-miner: Automatically extracting implicit programming rules and detecting violations in large software code	2005	Conference	3	2	Practice
[43]	Detecting object usage anomalies	2007	Conference	3	2	Practice
[44]	Mining API patterns as partial orders from source code: From usage scenarios to specifications	2007	Conference	3	2	Practice
[45]	Alattin: Mining alternative patterns for detecting neglected conditions	2009	Conference	3	2	Theory
[46]	Learning from 6000 projects: Lightweight cross-project anomaly detection	2010	Conference	3	2	Theory
[47]	Discovering neglected conditions in software by mining dependence graphs	2008	Journal	3	2	Theory
[48]	Chucky: Exposing missing checks in source code for vulnerability discovery	2013	Conference	3	2	Theory
[49]	Vulnerability extrapolation: Assisted discovery of vulnerabilities using machine learning	2011	Conference	3	1	Experiment
[50]	Generalized vulnerability extrapolation using abstract syntax trees	2012	Conference	3	1	Theory
[51]	Predicting common web application vulnerabilities from input validation and sanitization code patterns	2012	Conference	3	1	Experiment
[52]	Predicting SQL injection and cross site scripting vulnerabilities through mining input sanitization patterns	2013	Journal	3 3	1 4	Practice
[53]	Mining SQL injection and cross site scripting vulnerabilities using hybrid program analysis	2013	Conference	3	1	Experiment
[54]	Web application vulnerability prediction using hybrid program analysis and machine learning	2015	Journal	3	1	Theory
[55]	Predicting vulnerable software components via text mining	2014	Journal	3	1	Theory
[56]	Automatic inference of search patterns for taintstyle vulnerabilities	2015	Conference	3 3	1 4	Experiment
[57]	Predicting vulnerable software components through N-gram analysis and statistical feature selection	2015	Conference	3	1	Theory
[120]	Classification and Analysis of Android Malware Images Using Feature Fusion Technique	2021	Conference	3	1	Practice
[121]	SHELLCORE: Automating Malicious IoT Software Detection Using Shell Commands Representation	2021	Conference	3 3	1 5	Practice
[98]	Machine Learning Tensor Flow Based Platform for Recognition of Hand Written Text	2021	Conference	1 2	1 1	Practice
[99]	A Machine Learning-Based Framework for Mobile Forensics	2020	Conference	1 1	1 4	Practice
[129]	Automation of Vulnerability Classification from its Description using Machine Learning	2020	Conference	4	1	Practice
[132]	Threats Classification Method for the Transport Infrastructure of a Smart City	2020	Conference	4	4	Theory

The vast majority of the publications belong to conferences, a quarter to journals, and 1 to the other types. The main reason for this distribution is most likely the greater ease of acceptance into a conference proceedings collection than into a journal. Nevertheless, we can reasonably conclude that the scientific community has sufficient awareness of decisions in the field, since information from conferences is more open than from journals. Thus, decisions regarding the use of machine learning for static analysis are of a discussion nature.

Consequently, the systematization of all such decisions in this study will accelerate the bringing of the remaining studies to practical implementation. This determines the value of the current survey.

Thirdly, we analyze the number of publications according to their scientific and practical content, presented as a bar chart in Figure 4.

We can see that the majority of publications refer more to theoretical research (creation of models and algorithms), although featuring the conduct of certain experiments. Slightly fewer publications are devoted to already finished prototypes, which have passed the minimum necessary testing and are used for experimental evaluations. Substantially fewer publications describe experiments on a prototype with the minimum required functionality. This distribution can be attempted to explain by the fact that the vast majority of the studies, after the theory had been worked out, failed to implement it in practice.

While the part of the research, in which the prototype was nevertheless implemented, successfully brought it to a completed state. Only a small part of the research refused to implement the practical part after conducting experiments. Thus, the following conclusions can be drawn. Bringing the research to its logical conclusion directly depends on the "success" of the choice of the initial idea. The proposed systematization of machine-learning applications for analysis can serve as sources of new, more grounded ideas, supported by already obtained research results.

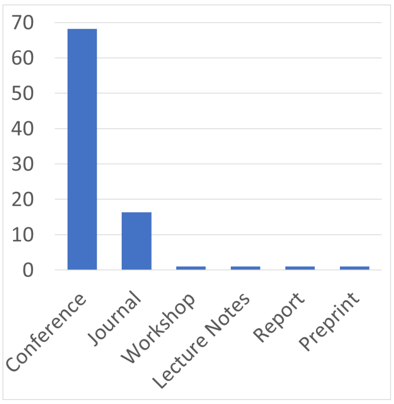

Figure 3. Distribution of publications by type.

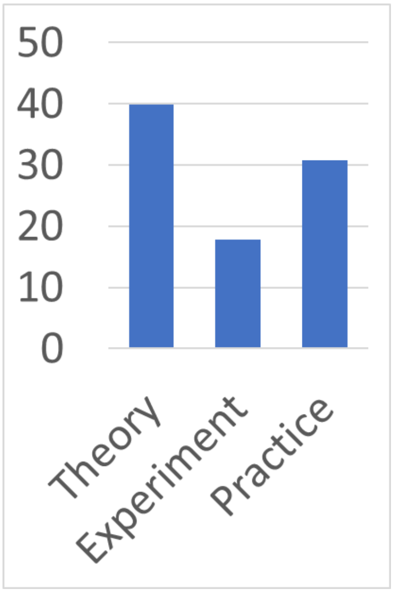

Figure 4. Distribution of publications by content.

Based on the above conclusions on results of quantitative analysis by year, type, and content publications, we can draw the following most important conclusion, which also determines the general direction of development in this subject area:

"While there is sufficient awareness of ML-enabled SA solutions, new breakthrough ideas are needed to evolve IS approaches for IoTS to full-fledged practical products."

It is in this direction that the efforts of authors of this review are focused at.

We separately analyze the attribution each of the publications to the SA stage and the ML tasks. In doing so, some of the publications ([56,96,98,99,104–106,108,116,117,121,123]) were assigned to several such pairs at once. This would create an appropriate review model, which is presented in matrix form in Table 6. The model describes the works in which ML solutions are applied and which can be applied at each stage of the SA IoTS.

The authors attempted to search and select articles in the following way. The search methodology itself consisted of the following actions: entering a keyword in the *IEEE Explorer*, *ResearchGate*, and *Google Scholar* databases; searching for and studying suitable papers; studying references in papers; searching for new keywords from found papers; etc.

The obtained model has an important methodological value since it almost completely covers the area of study—"SA + ML". Moreover, it allows us to evaluate the state of elaboration of this area. The search was carried out in two stages—main and advanced.

During the main stage search, the keywords were used that characterize the application of one of the machine-learning tasks for this from the stages of static analysis. Combinations of names of stages and machine-learning tasks (as well as their synonyms or analogs) were used as keywords. For example, to search for the S1_T4 group, the following keywords were selected: "data collection clusterization", "data gather clusterization", "data collection cluster algorithm", etc. Thus, 20 (i.e., 4 Stages x 5 Tasks) search passes were made in all databases.

For the keywords of each SA stage and the ML task, a main search was made for scientific publications in each of the scientific databases (IEEE Explorer, ResearchGate, and Google Scholar). Of these, 100 relevant in each of the bases were selected. Then, out of 100 publications, close topics were selected: the current stage of the SA, the solution of the current task of the ML, and the applicability to the IoTS.

In the advanced stage search, instead of the task names (T1–T2), more general keywords were used—machine learning, intelligent, etc. Similarly to the main stage search, the 50 most relevant works were selected for the advanced stage search.

The main criteria for selecting articles for the review was their compliance, both with one of the stages and with one of the tasks. Additional criteria were their application for IoTS analysis. The adaptation of solutions in other areas to this one was taken into account—applying to the IoT.

The total number of found and selected works for each SA stage and ML task is shown in Table 6 and is equal to 91 (including 3 parts of the one investigation—[82–84]).

Thus, we can say with some certainty that the resulting distribution of publications in Table 6 reflects the current state and trends. Therefore, a significant amount of research is devoted to the processing of IoTS data in the interest of detecting IS violations by classifying and detecting anomalies (groups S3_T1—22 papers and S3_T2—19 papers), and predicting vulnerabilities using regression (group S3_T3—11 papers), often using generalization (group S3_T5—6 papers).

The attention of scientific papers is focused on the selection of IoTS data using classification (group S1_T1—9 papers). In contrast, for a number of groups (S1_T2, S2_T2, S1_T3, S2_T3, S4_T3, S2_T4, S4_T4, and S2_T5), there were only two studies each. No studies were found on the selection of objects in Stage 1 with the use of dimension reduction (group S1_T5—0 works).

Table 6. Overview model of scientific works on the implementation of the static analysis stages using machine learning.

Task Name	Stage 1. Data Collection	Stage 2. Data Preparation	Stage 3. Data Processing	Stage 4. Result Formation
Task 1. Classification	[81–85,89,90,92,97–99]	[21,22,98,100,101]	[17–20,49–58,110–112,116,117,119–121]	[122,123,129]
Task 2. Anomaly detection	[93,94]	[108,109]	[23–29,40–48,113,114,118]	[108,124,128]
Task 3. Regression	[95,96]	[96,107]	[30–39,58]	[130,131]
Task 4. Clustering	[86–88,91,99]	[102,103]	[52,56]	[123,132]
Task 5. Generalization		[104–106]	[104,105,115–117,121]	[106,123]

The absence of works for group S1_T5 can be explained by the rarity of the generalization task in IS data collection, nevertheless, such a situation is quite possible; for example, using probabilistic downscaling LSH (Locality-Sensitive Hashing) to find IoTS documents [133] close to a given one, with their merging to apply the other stages of SA.

Such a distribution of the research work of each SA stage and the ML problem solved in the process is well represented by the histogram in Figure 5.

In the author's opinion, the uneven distribution of articles in the model is a disadvantage: some subareas appear to be understudied. A number of papers cover several groups at once ([96,98,104–106,108,116,117,123]). All this suggests a targeted predisposition of the proposed solutions—capturing a few specific tasks (in each group) out of all of them.

Perhaps it would be more feasible to direct the research towards solving all of the subtasks individually, mapped to each group, i.e., filling each cell of Table 6 completely and evenly. In this way, the possibilities of applying intelligent methods to all groups would have been fully "uncovered".

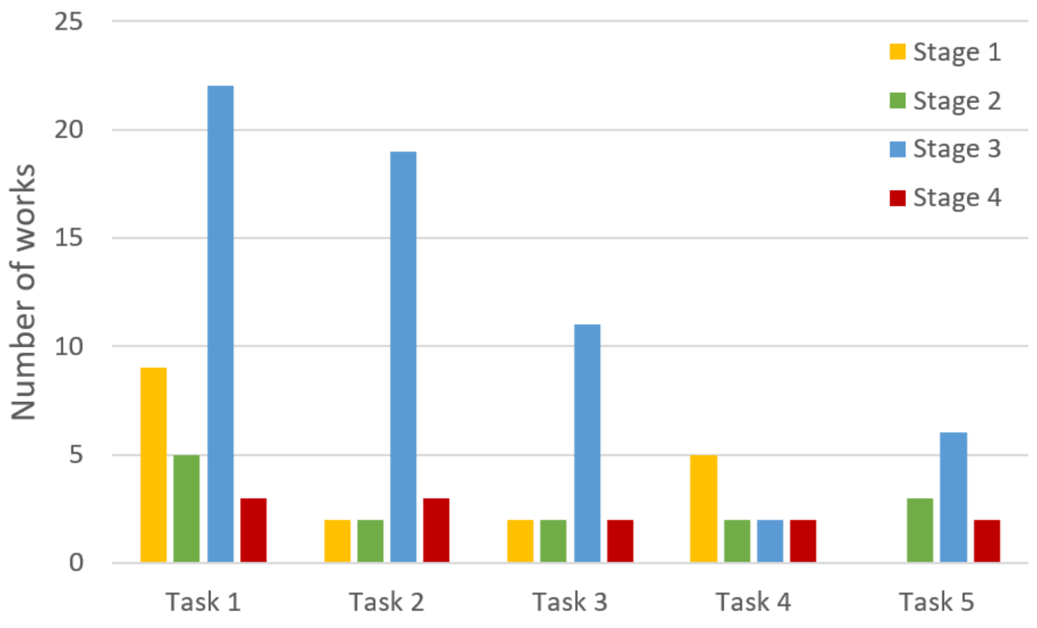

Figure 5. Histogram of research work distribution for static analysis and machine-learning tasks.

The general conclusion from the conducted analysis of scientific publications and their systematization in the form of a table is that for almost every combination of the SA stage and the ML problem there are a number of works substantiating such "symbiosis". Thus, the second set subtask can be considered solved, which confirms the main hypothesis of the current scientific research from a practical point of view.

A practical suggestion for confirming the hypothesis can be the creation of an intelligent SA framework of IoTS using ML methods. The framework can provide the ability to build complex meta-algorithms for analysis from basic blocks consisting of SA stages. Each of these building blocks can harness the full power of ML. Inputs data for some blocks (i.e., Stage 1) can be outputs from other blocks (i.e., Stage 4). Likewise, a block output (i.e., Stage 4) can serve as an input to other blocks (i.e., Stage 1). Thus, the framework allows building complex SA architectures for large IoTS from minimal ML-based SA blocks and pipes between them.

An example of a that intellectual framework as a set of interconnected single SA (as part of the described Stage 1–Stage 4) is shown in Figure 6.

The analysis process in Figure 6 is as follows:

1. Content of Logs and Scripts and File Attributes are analyzed by separate Pipes with numbers 1, 2, and 3.
2. Results of the first two Pipes are analyzed by the fourth Pipe and the third Pipe by the fifth Pipe.
3. Results of the fourth Pipe and the fifth Pipe are analyzed by the sixth Pipe, from which a Total Report is created.

Let us justify the possibility of conducting a complex SA based on an intelligent framework (similar to the one shown in Figure 6). Consider again the idea of a four-step process for a particular SA. As input (Stage 1), the SA process takes raw formalized data (usually files). In the main part of the SA (Steps 2 and 3), data is prepared, and problems in information security are searched for. The output of the SA process (Stage 4) generates a representation of the results in the form of formalized data (usually also files). However, the data obtained after Stage 4 may not be sufficient to detect problems on the IoTS.

For example, SA has identified graphs of suspicious interactions between programs, but information security problems themselves have not been detected. In this case, the received data must be subjected to a new SA process (also with four steps). For example, in the graph of interactions between programs, backdoors can be detected using a signature method. Thus, running a SA without any one of the four stages is impractical, and the output from one SA (files after Step 4) can be used as input from another (files from Step 1). Therefore, it makes sense to build an intelligent SA framework as shown in Figure 6.

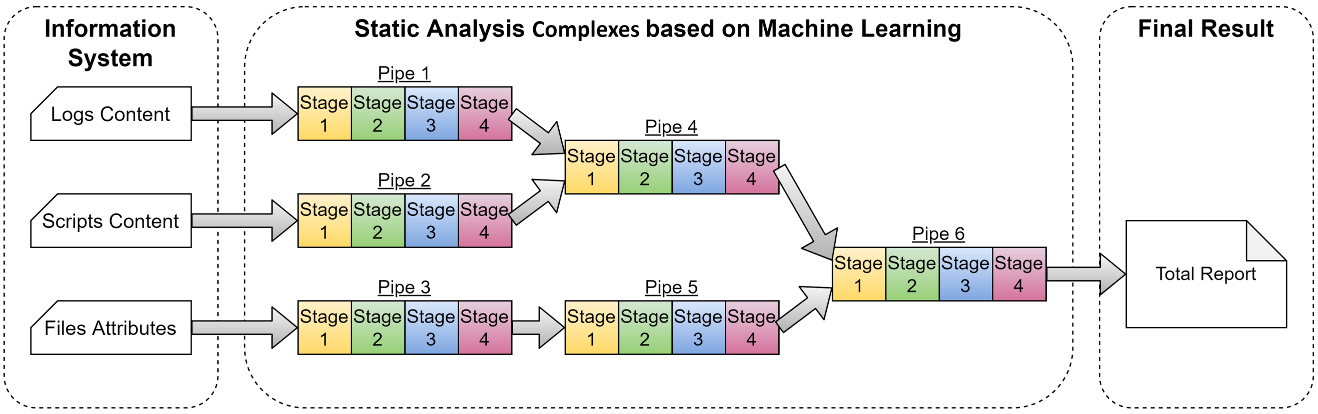

Figure 6. Example of complex static IoTS analysis using machine learning.

7. Conclusions

In the paper, a theoretical and practical proof of the hypothesis concerning the application of ML to SA was made.

Speaking about the first proof, in a formal form, the execution of each SA Stage and the solution of ML tasks on this stage was considered. Then, the stages and tasks were systematized into a generalized analytical model with a matrix form (see Table 2). The fact of existence and the correctness of the description of this model substantiates the hypothesis from a theoretical point of view. The significance of this scientific result lies in the approbation of the apparatus of formal proofs for solving certain problems by some decisions. The model can serve as the basis for practical implementations of intelligent SA algorithms.

Speaking of the second proof, a review of existing works applicable to the stages of SA of IoTS from the perspective of the tasks solved by ML was made to this end. The results are summarized in a matrix (see Table 6), and a review model was created. The advantage of the model is its necessity and sufficiency—almost all the works refer to its table element. Filling in the cells of the model substantiated the hypothesis from a practical point of view.

The significance of this scientific result lies in the creation of a single consistent base of intelligent solutions (i.e., using ML) in the interests of SA. Thus, if necessary, developers of IoTS analysis systems can make an informed choice of one or another solution for this SA Stage. They can assess the degree of elaboration and technical implementation of the chosen solution path.

The resulting models linking SA and ML allow designing methodological solutions (theoretically and practically justified) in the interest of providing IS of complex IoTS. Naturally, this requires the creation of an appropriate framework to ensure the implementation of all phases using the full variety of ML methods for Big Data and heterogeneous data. An important feature of the framework will be intellectualization supported by the ML methods. As many studies [134–136] emphasize, such systems are in great demand in the IoTS IS field.

Despite the availability of reviews regarding ML applications for IoT security (for example, from the point of view of countering attacks [137]), the review and taxonometry proposed in the current article have a number of significant differences. First, the article is devoted specifically to the IoTS analysis but not the attack detection or neutralization. Secondly, the analysis is exactly static (but not dynamic), which allows detecting violations in the system before its immediate launch. Thirdly, a comprehensive consideration of the SA stages and ML tasks allows us to assume not only existing methods of analysis but also hypothetical ones (for example, for a S1_T5). No such reviews have been found in existing scientific papers.

The following directions for further research are proposed.

First, similar to the current review, it is necessary to investigate the possibilities to intellectualize the DA, also breaking it down into phases and introducing solutions from the ML field. As a result, the entire field of IoTS analysis for IS will be fully grasped.

Secondly, based on the overall complexity of SA (as well as DA) large-scale heterogeneous IoTS and their data, a deep and more mathematically correct elaboration of the corresponding mathematical apparatus is required. Although the representation of SA in the form of one of the ML solutions modeled for each stage of SA are given in the article, it is nevertheless more intuitive than objectively correct.

Thirdly, following the concept of an intelligent framework (for the SA of IoTS), it is necessary to create its architecture, synthesis of the basic algorithms, their implementation as a prototype and testing in the "battlefield conditions" (similar to the author's research [138]). The success of subsequent experiments will prove the functionality of hypothetical framework and allow forming the requirements for its full-fledged development.

Author Contributions: Conceptualization, I.K.; methodology, M.B.; validation, I.K., K.I. and M.B.; investigation, I.K., K.I. and M.B.; writing—original draft preparation, I.K., K.I. and M.B.; writing—review and editing, I.K., K.I. and M.B.; visualization, K.I. All authors have read and agreed to the published version of the manuscript.

Funding: This research is being supported by the grant of RSF #21-71-20078 in St. Petersburg Federal Research Center of the Russian Academy of Sciences (SPC RAS).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data sharing is not applicable to this article.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Kucherova, K.; Mescheryakov, S.; Shchemelinin, D. Using Predictive Monitoring Models in Cloud Computing Systems. *Distributed Computer and Communication Networks*; Springer International Publishing: Cham, Switzerland, 2018; pp. 341–352.
2. Buinevich, M.; Izrailov, K.; Vladyko, A. Metric of vulnerability at the base of the life cycle of software representations. In Proceedings of the 2018 20th International Conference on Advanced Communication Technology (ICACT), Chuncheon, Korea, 11–14 February 2018. [CrossRef]
3. Komashinskiy, D.; Kotenko, I. Malware Detection by Data Mining Techniques Based on Positionally Dependent Features. In Proceedings of the 2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing, Pisa, Italy, 17–19 February 2010; pp. 617–623. [CrossRef]
4. Ageev, S.; Kopchak, Y.; Kotenko, I.; Saenko, I. Abnormal traffic detection in networks of the Internet of things based on fuzzy logical inference. In Proceedings of the 2015 XVIII International Conference on Soft Computing and Measurements (SCM), St. Petersburg, Russia, 19–21 May 2015; pp. 5–8. [CrossRef]
5. Desnitsky, V.A.; Kotenko, I.V.; Nogin, S.B. Detection of anomalies in data for monitoring of security components in the Internet of Things. In Proceedings of the 2015 XVIII International Conference on Soft Computing and Measurements (SCM), St. Petersburg, Russia, 19–21 May 2015; pp. 189–192. [CrossRef]
6. Kotenko, I.; Saenko, I.; Skorik, F.; Bushuev, S. Neural network approach to forecast the state of the Internet of Things elements. 2015 XVIII International Conference on Soft Computing and Measurements (SCM), St. Petersburg, Russia, 19–21 May 2015; pp. 133–135. [CrossRef]
7. Allamanis, M.; Barr, E.; Devanbu, P.; Sutton, C. A Survey of Machine Learning for Big Code and Naturalness. *ACM Comput. Surv.* **2017**, *51*, 36. [CrossRef]
8. Xue, H.; Sun, S.; Venkataramani, G.; Lan, T. Machine Learning-Based Analysis of Program Binaries: A Comprehensive Study. *IEEE Access* **2019**, *7*, 65889–65912. [CrossRef]
9. Ghaffarian, S.; Shahriari, H.R. Software Vulnerability Analysis and Discovery Using Machine-Learning and Data-Mining Techniques: A Survey. *ACM Comput. Surv.* **2017**, *50*, 1–36. [CrossRef]
10. Kotenko, I.; Saenko, I.; Kushnerevich, A.; Branitskiy, A. Attack Detection in IoT Critical Infrastructures: A Machine Learning and Big Data Processing Approach. In Proceedings of the 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), Pavia, Italy, 13–15 February 2019; pp. 340–347. [CrossRef]
11. Mescheryakov, S.; Shchemelinin, D.; Izrailov, K.; Pokussov, V. Digital Cloud Environment: Present Challenges and Future Forecast. *Future Internet* **2020**, *12*, 82. [CrossRef]
12. Fu, X.; Li, X.; Zhu, Y.; Wang, L.; Goh, R.S.M. An intelligent analysis and prediction model for on-demand cloud computing systems. In Proceedings of the International Joint Conference on Neural Networks, Beijing, China, 6–11 July 2014; IEEE: Beijing, China, 2014; pp. 1036–1041. [CrossRef]
13. Ardulov, Y.; Kucherova, K.; Mescheryakov, S.; Shchemelinin, D. Self-learning Machine Method for Anomaly Detection in Real Time Data. In Proceedings of the 10th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), Moscow, Russia, 5–9 November 2018; pp. 1–5. [CrossRef]
14. Borevich, E.; Mescheryakov, S.; Yanchus, V., Statistical Model of Computing Experiment on Digital Color Correction. In *Distributed Computer and Communication Networks*; Springer: Cham, Switzerland, 2019; pp. 140–150. [CrossRef]
15. Buinevich, M.; Izrailov, K.; Stolyarova, E.; Vladyko, A. Combine method of forecasting VANET cybersecurity for application of high priority way. In Proceedings of the 2018 20th International Conference on Advanced Communication Technology (ICACT), Chuncheon, Korea, 11–14 February 2018; pp. 266–271. [CrossRef]
16. Raju, A.D.; Abualhaol, I.Y.; Giagone, R.S.; Zhou, Y.; Huang, S. A Survey on Cross-Architectural IoT Malware Threat Hunting. *IEEE Access* **2021**, *9*, 91686–91709. [CrossRef]
17. Schultz, M.; Eskin, E.; Zadok, F.; Stolfo, S. Data mining methods for detection of new malicious executables. In Proceedings 2001 IEEE Symposium on Security and Privacy. S P 2001, Oakland, CA, USA, 14–16 May 2000; pp. 38–49. [CrossRef]
18. Shijo, P.; Salim, A. Integrated Static and Dynamic Analysis for Malware Detection. *Procedia Comput. Sci.* **2015**, *46*, 804–811. [CrossRef]
19. Sornil, O.; Liangboonprakong, C. Malware Classification Using N-grams Sequential Pattern Features. *Int. J. Inf. Process. Manag.* **2013**, *4*, 59–67.
20. Gavriluţ, D.; Cimpoeşu, M.; Anton, D.; Ciortuz, L. Malware detection using machine learning. In Proceedings of the International Multiconference on Computer Science and Information Technology, Mragowo, Poland, 12–14 October 2009; pp. 735–741. [CrossRef]
21. Bao, T.; Burket, J.; Woo, M.; Turner, R.; Brumley, D. BYTEWEIGHT: Learning to Recognize Functions in Binary Code. In Proceedings of the 23rd USENIX Conference on Security Symposium, San Diego, CA, USA, 20–22 August 2014; USENIX Association: San Jose, CA, USA, 2014; p. 845–860.
22. Shin, E.C.R.; Song, D.; Moazzezi, R. Recognizing Functions in Binaries with Neural Networks. In Proceedings of the 24th USENIX Security Symposium, Washington, DC, USA, 12–14 August 2015; USENIX Association: Washington, DC, USA, 2015; pp. 611–626.
23. Wang, S.; Liu, T.; Tan, L. Automatically Learning Semantic Features for Defect Prediction. In Proceedings of the 38th International Conference on Software Engineering, Austin, TX, USA, 14–22 May 2016; Association for Computing Machinery: New York, NY, USA, 2016; pp. 297–308. [CrossRef]

24. Fast, E.; Steffee, D.; Wang, L.; Brandt, J.R.; Bernstein, M.S. Emergent, Crowd-Scale Programming Practice in the IDE. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, Toronto, ON, Canada, 26 April–1 May 2014; Association for Computing Machinery: New York, NY, USA, 2014; pp. 2491–2500. [CrossRef]
25. Hsiao, C.H.; Cafarella, M.; Narayanasamy, S. Using Web Corpus Statistics for Program Analysis. *Sigplan Not.* **2014**, *49*, 49–65. [CrossRef]
26. Wang, S.; Chollak, D.; Movshovitz-Attias, D.; Tan, L. Bugram: Bug Detection with n-Gram Language Models. In Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, Singapore, 3–7 September 2016; Association for Computing Machinery: New York, NY, USA, 2016; pp. 708–719. [CrossRef]
27. Murali, V.; Chaudhuri, S.; Jermaine, C. Finding Likely Errors with Bayesian Specifications. *arXiv* **2017**, arXiv:1703.01370.
28. Allamanis, M.; Brockschmidt, M.; Khademi, M. Learning to Represent Programs with Graphs. *arXiv* **2017**, arXiv:1711.00740(2017).
29. Pradel, M.; Sen, K. *Deep Learning to Find Bugs*; Technical Report, Department of Computer Science, Technischen Universität Darmstadt: Hessen, Deutschland, 2017.
30. Meneely, A.; Williams, L. Strengthening the Empirical Analysis of the Relationship between Linus' Law and Software Security. In Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, Bolzano-Bozen, Italy, 16–17 September 2010; Association for Computing Machinery: New York, NY, USA, 2010. [CrossRef]
31. Doyle, M.; Walden, J. An Empirical Study of the Evolution of PHP Web Application Security. In Proceedings of the 3th International Workshop on Security Measurements and Metrics, Banff, AB, Canada, 21 September 2011; pp. 11–20. [CrossRef]
32. Shin, Y.; Williams, L.A. Can traditional fault prediction models be used for vulnerability prediction? *Empir. Softw. Eng.* **2013**, *18*, 25–59. [CrossRef]
33. Shin, Y.; Williams, L. An Initial Study on the Use of Execution Complexity Metrics as Indicators of Software Vulnerabilities. In Proceedings of the 7th International Workshop on Software Engineering for Secure Systems, Honolulu, HI, USA, 22 May 2011; Association for Computing Machinery: New York, NY, USA, 2011; p. 1–7. [CrossRef]
34. Shin, Y.; Meneely, A.; Williams, L.; Osborne, J.A. Evaluating Complexity, Code Churn, and Developer Activity Metrics as Indicators of Software Vulnerabilities. *IEEE Trans. Softw. Eng.* **2011**, *37*, 772–787. [CrossRef]
35. Moshtari, S.; Sami, A.; Azimi, M. Using complexity metrics to improve software security. *Comput. Fraud. Secur.* **2013**, *2013*, 8–17. [CrossRef]
36. Walden, J.; Stuckman, J.; Scandariato, R. Predicting Vulnerable Components: Software Metrics vs Text Mining. In Proceedings of the IEEE 25th International Symposium on Software Reliability Engineering, Naples, Italy, 3–6 November 2014; pp. 23–33. [CrossRef]
37. Morrison, P.; Herzig, K.; Murphy, B.; Williams, L. Challenges with Applying Vulnerability Prediction Models. In Proceedings of the 2015 Symposium and Bootcamp on the Science of Security, Urbana, IL, USA, 21–22 April 2015; Association for Computing Machinery: New York, NY, USA, 2015. [CrossRef]
38. Younis, A.; Malaiya, Y.; Anderson, C.; Ray, I. To Fear or Not to Fear That is the Question: Code Characteristics of a Vulnerable Functionwith an Existing Exploit. In Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy, New Orleans, LA, USA, 9–11 March 2016; Association for Computing Machinery: New York, NY, USA, 2016; p. 97–104. [CrossRef]
39. Zimmermann, T.; Nagappan, N.; Williams, L. Searching for a Needle in a Haystack: Predicting Security Vulnerabilities for Windows Vista. In Proceedings of the 3th International Conference on Software Testing, Verification and Validation, Paris, France, 6–10 April 2010; pp. 421–428. [CrossRef]
40. Engler, D.; Chen, D.Y.; Hallem, S.; Chou, A.; Chelf, B. Bugs as Deviant Behavior: A General Approach to Inferring Errors in Systems Code. *ACM Sigops Oper. Syst. Rev.* **2001**, *35*, 57–72. [CrossRef]
41. Livshits, B.; Zimmermann, T. DynaMine: Finding Common Error Patterns by Mining Software Revision Histories. *SIGSOFT Softw. Eng. Notes* **2005**, *30*, 296–305. [CrossRef]
42. Li, Z.; Zhou, Y. PR-Miner: Automatically Extracting Implicit Programming Rules and Detecting Violations in Large Software Code. *SIGSOFT Softw. Eng. Notes* **2005**, *30*, 306–315. [CrossRef]
43. Wasylkowski, A.; Zeller, A.; Lindig, C. Detecting Object Usage Anomalies. In Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering, Dubrovnik, Croatia, 3–7 September 2007; Association for Computing Machinery: New York, NY, USA, 2007; pp. 35–44. [CrossRef]
44. Acharya, M.; Xie, T.; Pei, J.; Xu, J. Mining API Patterns as Partial Orders from Source Code: From Usage Scenarios to Specifications. In Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering, Dubrovnik, Croatia, 3–7 September 2007; Association for Computing Machinery: New York, NY, USA, 2007; pp. 25–34. [CrossRef]
45. Thummalapenta, S.; Xie, T. Alattin: Mining Alternative Patterns for Detecting Neglected Conditions. In Proceedings of the IEEE/ACM International Conference on Automated Software Engineering, Auckland, New Zealand, 16–20 November 2009; pp. 283–294. [CrossRef]
46. Gruska, N.; Wasylkowski, A.; Zeller, A. Learning from 6,000 Projects: Lightweight Cross-Project Anomaly Detection. In Proceedings of the 19th International Symposium on Software Testing and Analysis, New York, NY, USA, 12–16 July 2010; Association for Computing Machinery: New York, NY, USA, 2010; pp. 119–130. [CrossRef]
47. Chang, R.Y.; Podgurski, A.; Yang, J. Discovering Neglected Conditions in Software by Mining Dependence Graphs. *IEEE Trans. Softw. Eng.* **2008**, *34*, 579–596. [CrossRef]

48. Yamaguchi, F.; Wressnegger, C.; Gascon, H.; Rieck, K. Chucky: Exposing Missing Checks in Source Code for Vulnerability Discovery. In Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security, Berlin, Germany, 4–8 November 2013; Association for Computing Machinery: New York, NY, USA, 2013; pp. 499–510. [CrossRef]
49. Yamaguchi, F.; Lindner, F.; Rieck, K. Vulnerability Extrapolation: Assisted Discovery of Vulnerabilities Using Machine Learning. In Proceedings of the 5th USENIX Conference on Offensive Technologies, San Francisco, CA, USA, 8 August 2011; USENIX Association: San Jose, CA, USA, 2011; p. 13.
50. Yamaguchi, F.; Lottmann, M.; Rieck, K. Generalized Vulnerability Extrapolation Using Abstract Syntax Trees. In Proceedings of the 28th Annual Computer Security Applications Conference, Orlando, FL, USA, 3–7 December 2012; Association for Computing Machinery: New York, NY, USA, 2012; pp. 359–368. [CrossRef]
51. Shar, L.K.; Tan, H.B.K. Predicting common web application vulnerabilities from input validation and sanitization code patterns. In Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering, Essen, Germany, 3–7 September 2012; pp. 310–313. [CrossRef]
52. Shar, L.K.; Tan, H.B.K. Predicting SQL injection and cross site scripting vulnerabilities through mining input sanitization patterns. *Inf. Softw. Technol.* **2013**, *55*, 1767–1780. [CrossRef]
53. Shar, L.K.; Beng Kuan Tan, H.; Briand, L.C. Mining SQL injection and cross site scripting vulnerabilities using hybrid program analysis. In Proceedings of the 35th International Conference on Software Engineering (ICSE), San Francisco, CA, USA, 18–26 May 2013; pp. 642–651. [CrossRef]
54. Shar, L.K.; Briand, L.C.; Tan, H.B.K. Web Application Vulnerability Prediction Using Hybrid Program Analysis and Machine Learning. *IEEE Trans. Dependable Secur. Comput.* **2015**, *12*, 688–707. [CrossRef]
55. Scandariato, R.; Walden, J.; Hovsepyan, A.; Joosen, W. Predicting Vulnerable Software Components via Text Mining. *IEEE Trans. Softw. Eng.* **2014**, *40*, 993–1006. [CrossRef]
56. Yamaguchi, F.; Maier, A.; Gascon, H.; Rieck, K. Automatic Inference of Search Patterns for Taint-Style Vulnerabilities. In Proceedings of the IEEE Symposium on Security and Privacy, San Jose, CA, USA, 17–21 May 2015; pp. 797–812. [CrossRef]
57. Pang, Y.; Xue, X.; Namin, A.S. Predicting Vulnerable Software Components through N-Gram Analysis and Statistical Feature Selection. In Proceedings of the IEEE 14th International Conference on Machine Learning and Applications (ICMLA), Miami, FL, USA, 9–11 December 2015; pp. 543–548. [CrossRef]
58. Grieco, G.; Grinblat, G.L.; Uzal, L.; Rawat, S.; Feist, J.; Mounier, L. Toward Large-Scale Vulnerability Discovery Using Machine Learning. In Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy, New Orleans, LA, USA, 9–11 March 2016; Association for Computing Machinery: New York, NY, USA, 2016; pp. 85–96. [CrossRef]
59. Sparks, S.; Embleton, S.; Cunningham, R.; Zou, C. Automated Vulnerability Analysis: Leveraging Control Flow for Evolutionary Input Crafting. In Proceedings of the 23th Annual Computer Security Applications Conference (ACSAC 2007), Miami Beach, FL, USA, 10–14 December 2007; pp. 477–486. [CrossRef]
60. Wijayasekara, D.; Manic, M.; Wright, J.L.; McQueen, M. Mining Bug Databases for Unidentified Software Vulnerabilities. In Proceedings of the 5th International Conference on Human System Interactions, Perth, WA, Australia, 6–8 June 2012; pp. 89–96. [CrossRef]
61. Wijayasekara, D.; Manic, M.; McQueen, M. Vulnerability identification and classification via text mining bug databases. In Proceedings of the 40th Annual Conference of the IEEE Industrial Electronics Society, Dallas, TX, USA, 29 October–1 November 2014; pp. 3612–3618. [CrossRef]
62. Alvares, M.; Marwala, T.; de Lima Neto, F.B. Applications of computational intelligence for static software checking against memory corruption vulnerabilities. In Proceedings of the IEEE Symposium on Computational Intelligence in Cyber Security (CICS), Singapore, 16–19 April 2013; pp. 59–66. [CrossRef]
63. Medeiros, I.; Neves, N.F.; Correia, M. Automatic Detection and Correction of Web Application Vulnerabilities Using Data Mining to Predict False Positives. In Proceedings of the 23rd International Conference on World Wide Web, Seoul, Korea, 7–11 April 2014; Association for Computing Machinery: New York, NY, USA, 2014; pp. 63–74. [CrossRef]
64. Sadeghi, A.; Esfahani, N.; Malek, S. Mining the Categorized Software Repositories to Improve the Analysis of Security Vulnerabilities. In *International Conference on Fundamental Approaches to Software Engineering*; Gnesi, S.; Rensink, A., Eds.; Springer: Berlin/Heidelberg, Germany, 2014; pp. 155–169.
65. Buinevich, M.; Izrailov, K.; Ganov, G. Intellectual method of program interactions visualisation in unix-like systems for information security purposes. In Proceedings of the 12th Majorov International Conference on Software Engineering and Computer Systems, Saint Petersburg, Russia, 10–11 December 2020; pp. 1–12.
66. Liu, Y.; Wang, J.; Li, J.; Niu, S.; Song, H. Machine Learning for the Detection and Identification of Internet of Things Devices: A Survey. *IEEE Internet Things J.* **2022**, *9*, 298–320. [CrossRef]
67. Harbi, Y.; Aliouat, Z.; Refoufi, A.; Harous, S. Recent Security Trends in Internet of Things: A Comprehensive Survey. *IEEE Access* **2021**, *9*, 113292–113314. [CrossRef]
68. Zaman, S.; Alhazmi, K.; Aseeri, M.A.; Ahmed, M.R.; Khan, R.T.; Kaiser, M.S.; Mahmud, M. Security Threats and Artificial Intelligence Based Countermeasures for Internet of Things Networks: A Comprehensive Survey. *IEEE Access* **2021**, *9*, 94668–94690. [CrossRef]
69. Nguyen, D.C.; Ding, M.; Pathirana, P.N.; Seneviratne, A.; Li, J.; Vincent Poor, H. Federated Learning for Internet of Things: A Comprehensive Survey. *IEEE Commun. Surv. Tutor.* **2021**, *23*, 1622–1658. [CrossRef]

70. Jiang, J.R. Short Survey on Physical Layer Authentication by Machine-Learning for 5G-based Internet of Things. In Proceedings of the 2020 3rd IEEE International Conference on Knowledge Innovation and Invention (ICKII), Kaohsiung, Taiwan, 21–23 August 2020; pp. 41–44. [CrossRef]
71. Babu, M.R.; Veena, K.N. A Survey on Attack Detection Methods For IOT Using Machine Learning And Deep Learning. In Proceedings of the 2021 3rd International Conference on Signal Processing and Communication (ICPSC), Coimbatore, India, 13–14 May 2021; pp. 625–630. [CrossRef]
72. Wu, H.; Han, H.; Wang, X.; Sun, S. Research on Artificial Intelligence Enhancing Internet of Things Security: A Survey. *IEEE Access* **2020**, *8*, 153826–153848. [CrossRef]
73. Matin, I.M.M.; Rahardjo, B. The Use of Honeypot in Machine Learning Based on Malware Detection: A Review. In Proceedings of the 2020 8th International Conference on Cyber and IT Service Management (CITSM), Pangkal, Indonesia, 23–24 October 2020; pp. 1–6. [CrossRef]
74. Uma, K.; Blessie, E.S. Survey on Android Malware Detection and Protection using Data Mining Algorithms. In Proceedings of the 2018 2nd International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), 2018 2nd International Conference on, Palladam, India, 30–31 August 2018; pp. 209–212. [CrossRef]
75. Ayewah, N.; Pugh, W.; Hovemeyer, D.; Morgenthaler, J.D.; Penix, J. Experiences Using Static Analysis to Find Bugs. *IEEE Softw.* **2008**, *25*, 22–29. [CrossRef]
76. Asryan, S.; Hakobyan, J.; Sargsyan, S.; Kurmangaleev, S. Combining dynamic symbolic execution, code static analysis and fuzzing. *Proc. Inst. Syst. Program. RAS* **2018**, *30*, 25–38. [CrossRef]
77. Aslanyan, H. Platform for interprocedural static analysis of binary code. *Proc. Inst. Syst. Program. RAS* **2018**, *30*, 89–100. [CrossRef]
78. Bergeron, J.; Debbabi, M.; Erhioui, M.; Ktari, B. Static analysis of binary code to isolate malicious behaviors. In Proceedings of the IEEE 8th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE'99), Stanford, CA, USA, 18 June 1999; pp. 184–189. [CrossRef]
79. L'Heureux, A.; Grolinger, K.; Elyamany, H.F.; Capretz, M.A.M. Machine Learning With Big Data: Challenges and Approaches. *IEEE Access* **2017**, *5*, 7776–7797. [CrossRef]
80. Wang, M.; Cui, Y.; Wang, X.; Xiao, S.; Jiang, J. Machine Learning for Networking: Workflow, Advances and Opportunities. *IEEE Netw.* **2018**, *32*, 92–99. [CrossRef]
81. Buinevich, M.; Izrailov, K. Method for classification of files based on machine learning technology. *Bull. St. Petersburg State Univ. Technol. Des. Ser. Nat. Tech. Sci.* **2020**, *1*, 34–41. [CrossRef]
82. Buinevich, M.; Izrailov, K. Identification of Processor's Architecture of Executable Code Based on Machine Learning. Part 1. Frequency Byte Model. *Proc. Telecommun. Univ.* **2020**, *6*, 77–85. [CrossRef]
83. Buinevich, M.; Izrailov, K. Identification of Processor's Architecture of Executable Code Based on Machine Learning. Part 2. Identification Method. *Proc. Telecommun. Univ.* **2020**, *6*, 104–112. [CrossRef]
84. Buinevich, M.; Izrailov, K. Identification of Processor's Architecture of Executable Code Based on Machine Learning. Part 3. Assessment Quality and Applicability Border. *Proc. Telecommun. Univ.* **2020**, *6*, 48–57. [CrossRef]
85. Sportiello, L.; Zanero, S. File Block Classification by Support Vector Machine. In Proceedings of the Sixth International Conference on Availability, Reliability and Security, Vienna, Austria, 22–26 August 2011; pp. 307–312. [CrossRef]
86. Dash, M.; Liu, H. Similarity detection among data files-a machine learning approach. In Proceedings of the 1997 IEEE Knowledge and Data Engineering Exchange Workshop, Newport Beach, CA, USA, 4 November 1997; pp. 172–179. [CrossRef]
87. Arif, W.; Mahoto, N.A. Document Clustering – A Feasible Demonstration with K-means Algorithm. In Proceedings of the 2nd International Conference on Computing, Mathematics and Engineering Technologies (iCoMET), Sukkur, Pakistan, 30–31 January 2019; pp. 1–6. [CrossRef]
88. da Cruz Nassif, L.F.; Hruschka, E.R. Document Clustering for Forensic Computing: An Approach for Improving Computer Inspection. In Proceedings of the 10th International Conference on Machine Learning and Applications and Workshops, Honolulu, HI, USA, 18–21 December 2011; Volume 1, pp. 265–268. [CrossRef]
89. Kumar, J.; Pillai, J.; Doermann, D. Document Image Classification and Labeling Using Multiple Instance Learning. In Proceedings of the International Conference on Document Analysis and Recognition, Beijing, China, 18–21 September 2011; pp. 1059–1063. [CrossRef]
90. Zhu, G.; Zheng, Y.; Doermann, D.; Jaeger, S. Multi-scale Structural Saliency for Signature Detection. In Proceedings of the 2007 IEEE Conference on Computer Vision and Pattern Recognition, Minneapolis, MN, USA, 17–22 June 2007; pp. 1–8. [CrossRef]
91. Zhang, M.L.; Zhou, Z.H. Multi-instance clustering with applications to multi-instance prediction. *Appl. Intell.* **2009**, *31*, 47–68. [CrossRef]
92. Wang, T.Y.; Wu, C.H. Detection of packed executables using support vector machines. In Proceedings of the International Conference on Machine Learning and Cybernetics, Guilin, China, 10–13 July 2011; Volume 2; pp. 717–722. [CrossRef]
93. Hubballi, N.; Dogra, H. Detecting Packed Executable File: Supervised or Anomaly Detection Method? In Proceedings of the 11th International Conference on Availability, Reliability and Security (ARES), Salzburg, Austria, 31 August–2 September 2016; pp. 638–643. [CrossRef]

94. Uzum, I.; Can, O. An anomaly detection system proposal to ensure information security for file integrations. In Proceedings of the 2018 26th Signal Processing and Communications Applications Conference (SIU), Izmir, Turkey, 2–5 May 2018; pp. 1–4. [CrossRef]
95. Monjalet, F.; Leibovici, T. Predicting File Lifetimes with Machine Learning. In *International Conference on High Performance Computing*; Springer: Cham, Switzerland, 2019; Volume 11887, pp. 288–299. [CrossRef]
96. Gomis, F.K.; Camara, M.S.; Diop, I.; Farssi, S.M.; Tall, K.; Diouf, B. Multiple linear regression for universal steganalysis of images. In Proceedings of the International Conference on Intelligent Systems and Computer Vision (ISCV), Fez, Morocco, 2–4 April 2018; pp. 1–4. [CrossRef]
97. Kumar, B.; Vadlamani, R. Text Document Classification with PCA and One-Class SVM. In *Proceedings of the 5th International Conference on Frontiers in Intelligent Computing: Theory and Applications*; Advances in Intelligent Systems and Computing; Springer: Singapore, 2017; Volume 515, pp. 107–115. [CrossRef]
98. Gupta, N.; Goyal, N. Machine Learning Tensor Flow Based Platform for Recognition of Hand Written Text. In Proceedings of the International Conference on Computer Communication and Informatics (ICCCI), Coimbatore, India, 27–29 January 2021; pp. 1–6. [CrossRef]
99. Peng, L.; Zhu, X.; Zhang, P. A Machine Learning-Based Framework for Mobile Forensics. In Proceedings of the IEEE 20th International Conference on Communication Technology (ICCT), Nanning, China, 28–31 October 2020; pp. 1551–1555. [CrossRef]
100. Xu, Z.; Wen, C.; Qin, S. Type Learning for Binaries and Its Applications. *IEEE Trans. Reliab.* **2019**, *68*, 893–912. [CrossRef]
101. Rosenblum, N.; Zhu, X.; Miller, B.; Hunt, K. Machine Learning-Assisted Binary Code Analysis. In Proceedings of the NIPS Workshop Machine Learning Adversarial Environment, Vancouver, BC, Canada, 7–8 December 2007; pp. 1–3.
102. Zahid, M.; Mehmmod, Z.; Inayat, I. Evolution in software architecture recovery techniques—A survey. In Proceedings of the 13th International Conference on Emerging Technologies (ICET), Islamabad, Pakistan, 27–28 December 2017; pp. 1–6. [CrossRef]
103. Marian, Z.; Czibula, I.G.; Czibula, G. A Hierarchical Clustering-Based Approach for Software Restructuring at the Package Level. In Proceedings of the 2017 19th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), Timisoara, Romania, 21–24 September 2017; pp. 239–246. [CrossRef]
104. Tsague, H.D.; Twala, B. Reverse engineering smart card malware using side channel analysis with machine learning techniques. In Proceedings of the IEEE International Conference on Big Data (Big Data), Washington, DC, USA, 5–8 December 2016; pp. 3713–3721. [CrossRef]
105. Park, J.; Xu, X.; Jin, Y.; Forte, D.; Tehranipoor, M. Power-based Side-Channel Instruction-level Disassembler. In Proceedings of the 55th ACM/ESDA/IEEE Design Automation Conference (DAC), San Francisco, CA, USA, 24–28 June 2018; pp. 1–6. [CrossRef]
106. Karimi, A.; Moattar, M.H. Android ransomware detection using reduced opcode sequence and image similarity. In Proceedings of the 7th International Conference on Computer and Knowledge Engineering (ICCKE), Mashhad, Iran, 26–27 October 2017; pp. 229–234. [CrossRef]
107. Saurav, S.; Schwarz, P. A Machine-Learning Approach to Automatic Detection of Delimiters in Tabular Data Files. In Proceedings of the IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS), Sydney, NSW, Australia, 12–14 December 2016; pp. 1501–1503. [CrossRef]
108. Yang, T.; Agrawal, V. *Log File Anomaly Detection*; Cource of Deep Learning for Natural Language (CS224d); Technical Report; Stanford University: Stanford, CA, USA, 2016.
109. Akanle, M.; Adetiba, E.; Akande, V.; Akinrinmade, A.; Ajala, S.; Moninuola, F.; Badejo, J.; Adebiyi, E. Experimentations with OpenStack System Logs and Support Vector Machine for an Anomaly Detection Model in a Private Cloud Infrastructure. In Proceedings of the 2020 International Conference on Artificial Intelligence, Big Data, Computing and Data Communication Systems (icABCD), Durban, South Africa, 6–7 August 2020; pp. 1–7. [CrossRef]
110. Shabtai, A.; Moskovitch, R.; Elovici, Y.; Glezer, C. Detection of malicious code by applying machine learning classifiers on static features: A state-of-the-art survey. *Inf. Secur. Tech. Rep.* **2009**, *14*, 16–29. [CrossRef]
111. Moskovitch, R.; Nissim, N.; Elovici, Y. Malicious Code Detection Using Active Learning. In *Privacy, Security, and Trust in KDD*; Bonchi, F., Ferrari, E., Jiang, W., Malin, B., Eds.; Springer: Berlin/Heidelberg, Germany, 2009; pp. 74–91.
112. Liu, S.; Dibaei, M.; Tai, Y.; Chen, C.; Zhang, J.; Xiang, Y. Cyber Vulnerability Intelligence for Internet of Things Binary. *IEEE Trans. Ind. Inform.* **2020**, *16*, 2154–2163. [CrossRef]
113. Abah, J.; Waziri, V.; Abdullahi, M.; U.M, A.; O.S, A. A Machine Learning Approach to Anomaly-Based Detection on Android Platforms. *Int. J. Netw. Secur. Its Appl.* **2015**, *7*, 15–35. [CrossRef]
114. Ng, D.V.; Hwang, J.I.G. Android malware detection using the dendritic cell algorithm. In Proceedings of the International Conference on Machine Learning and Cybernetics, Lanzhou, China, 13–16 July 2014; Volume 1; pp. 257–262. [CrossRef]
115. Ouyang, L.; Dong, F.; Zhang, M. Android malware detection using 3-level ensemble. In Proceedings of the 4th International Conference on Cloud Computing and Intelligence Systems (CCIS), Beijing, China, 17–19 August 2016; pp. 393–397. [CrossRef]
116. Khammas, B.M.; Monemi, A.; Stephen Bassi, J.; Ismail, I.; Mohd Nor, S.; Marsono, M.N. Feature selection and machine learning classification for malware detection. *J. Teknol.* **2015**, *77*, 243–250. [CrossRef]
117. Xiaoyan, Z.; Juan, F.; Xiujuan, W. Android malware detection based on permissions. In Proceedings of the International Conference on Information and Communications Technologies (ICT 2014), Nanjing, China, 15–17 May 2014; pp. 1–5. [CrossRef]

118. Bucevschi, A.G.; Balan, G.; Prelipcean, D.B. Preventing File-Less Attacks with Machine Learning Techniques. In Proceedings of the 21st International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), Timisoara, Romania, 4–7 September 2019; pp. 248–252. [CrossRef]
119. Zhang, J. *Machine Learning With Feature Selection Using Principal Component Analysis for Malware Detection: A Case Study*; Technical Report; Sophos: Abingdon, UK, 2019.
120. Singh, J.; Thakur, D.; Gera, T.; Shah, B.; Abuhmed, T.; Ali, F. Classification and Analysis of Android Malware Images Using Feature Fusion Technique. *IEEE Access* **2021**, *9*, 90102–90117. [CrossRef]
121. Alasmary, H.; Anwar, A.; Abusnaina, A.; Alabduljabbar, A.; Abuhamad, M.; Wang, A.; Nyang, D.H.; Awad, A.; Mohaisen, D. SHELLCORE: Automating Malicious IoT Software Detection Using Shell Commands Representation. *IEEE Internet Things J.* **2021**, *9*, 2485–2496. [CrossRef]
122. Otsubo, Y.; Otsuka, A.; Mimura, M.; Sakaki, T. o-glasses: Visualizing X86 Code From Binary Using a 1D-CNN. *IEEE Access* **2020**, *8*, 31753–31763. [CrossRef]
123. Yang, H.; Li, S.; Wu, X.; Lu, H.; Han, W. A Novel Solutions for Malicious Code Detection and Family Clustering Based on Machine Learning. *IEEE Access* **2019**, *7*, 148853–148860. [CrossRef]
124. Wilkinson, L. Visualizing Big Data Outliers Through Distributed Aggregation. *IEEE Trans. Vis. Comput. Graph.* **2018**, *24*, 256–266. [CrossRef]
125. Henry, T. *Testing For Normality*; CRC Press: Boca Raton, FL, USA, 2002; p. 368. [CrossRef]
126. Boris Iglewicz, D.C.H. *Volume 16: How to Detect and Handle Outliers*; ASQC Quality Press: Milwaukee, WI, USA, 2013; p. 87.
127. Hinneburg, A.; Keim, D.; Wawryniuk, M. HD-Eye: visual mining of high-dimensional data. *IEEE Comput. Graph. Appl.* **1999**, *19*, 22–31. [CrossRef]
128. Baseman, E.; Blanchard, S.; Li, Z.; Fu, S. Relational Synthesis of Text and Numeric Data for Anomaly Detection on Computing System Logs. In Proceedings of the 15th IEEE International Conference on Machine Learning and Applications (ICMLA), Anaheim, CA, USA, 18–20 December 2016; pp. 882–885. [CrossRef]
129. Aota, M.; Kanehara, H.; Kubo, M.; Murata, N.; Sun, B.; Takahashi, T. Automation of Vulnerability Classification from its Description using Machine Learning. In Proceedings of the IEEE Symposium on Computers and Communications (ISCC), Rennes, France, 7–10 July 2020; pp. 1–7. [CrossRef]
130. Last, D. Forecasting Zero-Day Vulnerabilities. In Proceedings of the 11th Annual Cyber and Information Security Research Conference, Oak Ridge, TN, USA, 5–7 April 2016; Association for Computing Machinery: New York, NY, USA, 2016; pp. 1–4. [CrossRef]
131. Zhang, T.; Xie, J.; Zhou, X.; Choi, C. The Effects of Depth of Field on Subjective Evaluation of Aesthetic Appeal and Image Quality of Photographs. *IEEE Access* **2020**, *8*, 13467–13475. [CrossRef]
132. Izrailov, K.; Chechulin, A.; Vitkova, L. Threats Classification Method for the Transport Infrastructure of a Smart City. In Proceedings of the IEEE 14th International Conference on Application of Information and Communication Technologies (AICT), Tashkent, Uzbekistan, 7–9 October 2020; pp. 1–6. [CrossRef]
133. Durmaz, O.; Bılge, H.S. Fast image search with distrubuted hashing. In Proceedings of the 26th Signal Processing and Communications Applications Conference (SIU), Izmir, Turkey, 2–5 May 2018; pp. 1–4. [CrossRef]
134. Aslanyan, H.; Asryan, S.; Hakobyan, J.; Vardanyan, V.; Sargsyan, S.; Kurmangaleev, S. Multiplatform Static Analysis Framework for Program Defects Detection. In Proceedings of the International Conference on Computer Sciences and Information Technologies, Helsinki, Finland, 21–23 August 2017; pp. 1–5.
135. Lee, S.; Dolby, J.; Ryu, S. HybriDroid: Static analysis framework for Android hybrid applications. In Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (ASE), Singapore, 3–7 September 2016; pp. 250–261.
136. Mihancea, P.F. Towards a Reverse Engineering Dataflow Analysis Framework for Java and C++. In Proceedings of the 10th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, Timisoara, Romania, 26–29 September 2008; pp. 285–288. [CrossRef]
137. Tahsien, S.; Karimipour, H.; Spachos, P. Machine learning based solutions for security of Internet of Things (IoT): A survey. *J. Netw. Comput. Appl.* **2020**, *161*, 102630. [CrossRef]
138. Kotenko, I.; Izrailov, K.; Buinevich, M. Analytical Modeling for Identification of the Machine Code Architecture of Cyberphysical Devices in Smart Homes. *Sensors* **2022**, *22*, 1017. [CrossRef]