Supplementary Material

1 SCOPE OF SUPPLEMENTARY MATERIAL

This document elaborates on supplementary material for our publication *Application-aware Intrusion Detection: A Systematic Literature Review, Implications for Automotive Systems, and Applicability of AutoML*. Section 2 introduces summaries of all publications selected for classification. Furthermore, Section 3 shows the pipelines configured by the AutoML runs. Additionally, we provide two Excel documents. *Table 1.xlsx* shows all 844 initial publications and the reasons for their inclusion or exclusion respectively. *Table 2.xlsx* shows the classification of the publications and can be used to infer which publications select a certain feature.

Please note that the summaries with the exception of Sections 2.7, 2.11, 2.13, and 2.16 are heavily based on the master's thesis of one of the authors Eikerling (2018). These texts have been updated in several iterations with respect to slight rephrasings, the evolving structure of the underlying feature model, and the new context of the publication. Thus, we refrain from using parenthetical citations.

2 SUMMARIES

This section presents the extracted summaries from the sources included in the final pool. Each source is presented with its title and the leaf features from its feature model variant. This allows to easily glance which parts of the feature model are fulfilled by the source. In addition the source summaries follow the top level feature model structure after a general introduction: with Approach, Architecture, Context, Monitored Data, Analysis Techniques, and Evaluation being discussed in this order.

2.1 ANJ17 Trust in IoT: Dynamic Remote Attestation through Efficient Behavior Capture (Ali et al., 2017)

- Anomaly Detection
- Application, Linux
- Design Driver, Embedded System
- Agent Based
- System Calls
- Length, Naïve Bayes
- Length, Hierarchical Naïve Bayes
- Length, Average One-Dependence Estimator
- Documented Attack, Productive System

Ali et al. detail an approach for remotely checking application behavior on devices in an "Internet of Things" context in a scalable manner. In addition to an approach for attestation, they also include classifiers for rating the applications' behavior as malicious or benign. Since the authors do not go into great detail on their approach, the classification may appear incomplete in parts.

We classify the general approach as anomaly detection, since the system logs normal application behavior. The architecture of the system is agent-based, since monitoring and an attestation agent reside on remote

devices, which are connected to centralized stakeholder machines wanting to validate the application behavior.

A prototype of the approach is implemented on a Linux platform, running different applications, such as the web browser Firefox and AMIDE, which is a medical imaging tool. The approach is geared towards "Internet of Things". In their logging approach, the authors opt to monitor system calls, specifically, unique system call sequences of fixed length.

The authors do not detail their pre-proccessing steps, only mentioning that system call sequences are logged and stored in secured memory locations called platform configuration registers (PCR). We assume that these logs are used to train machine learning classifiers, since the authors perform an evaluation using different machine learning approaches.

Particularly, the authors present results from classifiers that are variants of Bayes classifiers: hierarchical naïve Bayes models (HNB), averaged one-dependence estimator (AODE), Bayes network, and a naïve Bayes classifier. The results stem from exposing the system under scrutiny to documented attacks.

2.2 AP13 SHADuDT: Secure Hypervisor-Based Anomaly Detection Using Danger Theory (Azmi and Pishgoo, 2013)

- Anomaly Detection
- Design Driver, Rootkit
- Centralized
- System Calls
- Benign, Dynamic, Feature Vector
- Benign, Danger Theory
- Benign, Malicious, Danger Theory
- Distance Based, Event
- Documented Attack, Productive System

Azmi and Pishgoo propose an anomaly detection approach called SHADuDT, based on the idea to use an artificial immune system as a detector for malicious behavior. In order to train their Artificial Immune Systems (AIS), they employ both benign and malicious data during different steps in the process. Therefore, we categorize their approach as both anomaly and misuse detection. Their approach relies on a hypervisor for monitoring data and they indicate a centralized architecture in their work. SHADuDT is meant to target kernel level rootkits, which are used to evaluate the prototype in a case study. To detect such intrusions, the authors opt to monitor system calls using their hypervisor architecture.

Since the approach is based on AIS, many of the terms used in the work by Azmi and Pishgoo are from this domain. An introduction of AIS and all its terminology would be out of scope here, so we describe the employed analysis techniques in general terms.

Azmi and Pishgoo's approach falls in the category of danger theory, a branch of artificial immune systems. As in many machine learning approaches, the first step is to extract feature vectors from audited data, i.e., the monitored system calls. In this case, the feature vectors consist of system call number, access mode, flag arguments, process ID, user ID, and group ID.

The next pre-processing step is the training of the artificial immune system. This step consists of two parts. The first step is to generate the anomaly detectors (naïve T Cells) using normal training data.

In the second step, the signals that are received by the anomaly detectors are defined. The thresholds for the activation of a signal receptor are trained using normal and malicious data, depending on the receptor.

After the training phases, the classification of samples can be performed. This step entails evaluating the whole artificial immune system model. The evaluation is in part distance based and also consists of evaluating various formulas.

In order to evaluate their approach, the authors perform a case study with their own data. To gather the data, the authors let users work on a productive system. For gathering the anomalous data, two documented attacks were installed and executed on the system. The authors evaluate their prototype with a variety of parameters and compare themselves to other artificial immune system approaches.

2.3 ASW+18 SAMADroid: A Novel 3-Level Hybrid Malware Detection Model for Android Operating System (Arshad et al., 2018)

- Misuse Detection, Anomaly Detection
- Design Driver, Android
- Agent Based
- Application Identifiers
- Pre-Processing, Dynamic, Feature Vector
- Conformance, Batch
- Pre-Processing, Static, Feature Vector
- Benign, Malicious, Support Vector Machine
- Batch, Support Vector Machine
- Drebin

Arshad et al. propose a malware detection tool for resource constrained devices, i.e, Android smartphones. Their approach includes static and dynamic analysis techniques, which are executed on a remote server so as not to burden the smartphone with the detection overhead. Arshad et al.'s system, dubbed SAMADroid, trains an Support Vector Machine (SVM) classifier using malicious and benign data. Therefore, the approach is classified as misuse detection and anomaly detection. Since performing the classification of malware on the smartphone itself would produce too much overhead, the authors follow an agent-based approach. Data monitoring is performed using a client application, which sends the data to the server, which then performs the machine learning and classification tasks, notifying the client if malware has been detected.

The client monitors metadata of the applications under scrutiny as well as system calls. On the server side, the metadata is used to retrieve the binary code of the application in order to perform static analysis.

On the smartphone client, the first step is to monitor applications executed by the user through the SAMADroid interface. The client monitor first sends application metadata to the server, so it can be identified in order to begin the static analysis. In addition, the client monitors system calls using strace and stores them in a database. It then builds a feature vector containing the frequencies for ten different system calls and sends it to the server.

On the server side, more pre-processing steps follow. First, the server checks the application identifier received from the server to check its database for an existing classification of the application. If the application is already classified, the result is sent to the client. Otherwise, the application binary is downloaded and analyzed using static analysis. The result is a collection of features containing information

on the application's used hardware components, permissions, application components, intents, and API calls. The extracted features, both static and dynamic, are then mapped to the vector space to form feature vectors.

For classifying the feature vectors, SAMADroid uses an SVM, trained using existing malicious and benign applications. The training dataset used is the DREBIN dataset [1]. After the training, the feature vectors can be classified using the SVM. Since the client side monitors system calls as long as the application is actively in use, we assume that the system call logs are transferred to the server after application use in a batch style. Since the classification of dynamic and static feature vectors may differ, SAMADroid knows three levels of result: legitimate, if both feature vectors are classified as benign, malware, if both are classifed as malicious, and risky, if either one is deemed malicious. After classification, the results are sent back to the client application, where the user is notified of the results. The authors evaluate their approach using the aforementioned public DREBIN dataset. They also evaluate other machine learning techniques, namely Random Forest, Decision Tree, and a Naïve Bayes approach.

2.4 BJT17 PbMMD: A Novel Policy Based Multi-Process Malware Detection (Bidoki et al., 2017)

- Misuse Detection, Anomaly Detection
- Windows, Multi Process Malware
- System Calls, API Calls
- Benign, Malicious, Dynamic, Regular Expression
- Benign, Malicious, Clustering
- Benign, Malicious, PAM Algorithm
- Benign, Malicious, TD Lambda
- Benign, Malicious, TD Lambda
- Benign, Malicious, Support Vector Machine
- Conformance, Batch
- Conformance, Batch
- Documented Attack, Productive System

Bidoki et al.'s approach targets multi-process malware and uses both representations of normal behavior and malware patterns. Therefore, we classify the general approach as both malware and anomaly detection simultaneously. The authors evaluate their multi-process malware detection in a Windows XP environment. The approach, called PbMMD, requires the system and API call data for learning the behavior policies and the classification process.

The analysis process is split into three main parts: pre-processing, learning, and detection. Note that the pre-processing step is applied to the training data as well as the monitored data before it is classified. The first pre-processing step is to remove redundancy. The system calls are enumerated and repeating sequences of system calls are transformed to regular expressions with an indication of the frequency of occurrence. This allows for calculating similarity scores between sequences through sequence alignment. Since the exact number of repetitions for a certain system call sequence is not important, the sequences are categorized according to frequency thresholds.

[1] https://www.sec.cs.tu-bs.de/ danarp/drebin/

In a second pre-processing step, the sequences are transformed into a more coarsely grained structure, i.e., clusters, to enable more efficient analysis. The clusters are called system events and are created by executing the BLAST sequence matching algorithm iteratively over pairs of the system call sequences. During monitoring, system call sequences can be mapped to the clusters using pattern matching algorithms, examples of which are given by the authors.

The next step is the learning phase. In the learning phase, execution policies are learned by the approach. The first step is to cluster programs behaving similarly together. Since benign and malicious software is used for the learning process, the starting clusters are programs falling into the benign or malicious category. System events are observed and similarly in the training data and clustered together based on similarity, using the PAM algorithm from the K-medoids family.

For each of the clusters derived using PAM, an execution policy is learned. As mentioned before, the clusters consist of similar event sequences and each event sequence is the result of the execution of a single process program. In this step, the authors opted to use the reinforcement learning algorithm TD(λ). The algorithm considers each system event as a state and the execution policy for the particular cluster is the path of states that fits the cluster optimally.

Having learned execution policies, the next step is to models for single-process malware detection. This time, the TD(λ) is used on the whole dataset of event sequences and not the clusters as before. Each event sequence is labeled malicious or benign. The algorithm is rewarded for terminal states (events) that are marked as malicious. In this phase, events are given a value indicating its maliciousness. To calculate the threshold at which a sequence of events is considered malicious, the authors use another machine learning approach, namely an SVM.

For the classification of program behavior and especially identifying malware spanning multiple processes, it is first required to find cooperating processes, since the execution policies and malware models were learned on single processes. To this end, monitoring data is recorded from the inception of the targeted process and all subsequently created processes. This constitutes batch granularity, since the detector has to wait until the processes spawned during the execution of the target process are terminated, which may take an arbitrary amount of time.

First, PbMMD identifies which execution policy the target process follows by checking the conformance of the monitored events to the learned clusters. Then, it is checked whether a subsequent process' behavior belongs to the same cluster. If so, the processes are assumed to be cooperating and regarded as a single process in further testing.

To identify whether the behavior is malicious, the detector accumulates the sequence of events of the cooperating processes and checks the learned values of maliciousness. If the summed up values are larger than the learned threshold, an alarm is raised.

The authors evaluate the approach in a case study using their own data, generated using a productive Windows XP installation running user applications and utilities in a virtual machine. After collecting benign data, the authors executed a collection of malware to gather malicious data samples. The Windows utility procmon was used for collecting the data.

2.5 EBFS18 On Early Detection of Application-Level Resource Exhaustion and Starvation (Elsabagh et al., 2018)

- Anomaly Detection

- Application, Server, Utility, Linux
- Design Driver, Resource Starvation, Denial Of Service
- Centralized
- Kernel Events, Stack Operation Calls
- Benign, Static, Control Flow Graph
- Benign, K-Means
- Benign, Dynamic, Probabilistic Finite Automaton
- Conformance, Self Adaptive, Process Termination, Event
- Documented Attack, Testing

Elsabagh et al. present Radmin, an anomaly detection system to prevent denial of service and resource starvation attacks. Such attacks aim at crippling the attacked system by using up all computing resources, with the system not being able to restrict the resources consumed by a malicious process.

In their implementation, the authors focus on Linux systems running on x86 architectures. It places no limitation on the microarchitecture, binary format, or runtime environment. The intention behind Radmin is the detection of resource starvation or denial of service attacks. Since Radmin is supposed to not only detect resource related attacks but also protect against them, it is centrally deployed on the observed system. To detect resource starvation attacks, Radmin monitors resource related events in both kernel and user space. In the kernel space, it monitors kernel events that are related memory allocation, file descriptors, stack manipulation, CPU time allocation, and process child tasks. This is done by binding probes to the appropriate tracepoints.

For user space tracing, Radmin requires a compiled binary of the observed program. It parses the binary and extracts a Control Flow Graph (CFG). The CFG is analyzed for instructions that operate on the stack dynamically, which are then marked to be monitored later.

Analysis and protection is performed by the Radmin Guard. First, codebooks are learned, one for each resource type, that represent the consumption of a resources. The raw measurements (consumed kernel/user time, resource value) are processed using k-means clustering. Details on the clustering process can be found in the work by Arthur and Vassilvitskii Arthur and Vassilvitskii (2007).

Once the codebooks are learned, Radmin constructs a Probabilistic Finite Automaton (PFA) from a probabilistic suffix tree, which is a bounded variable-order Markov model. The probabilistic suffix tree contains the paths between resource consumption levels. To learn the PFA, Radmin needs benign monitoring data from the targeted program. This data is gained through dry runs, testing, automated functionality testing, or end users, with the latter option carrying the risk of introducing anomalous behavior into the training data.

During the classification phase, the Radmin guard receives the resource consumption measurements, encodes them according to the learned codebooks, and executes the learned PFA along with the monitored program. An alarm is raised if the current resource consumption attempt does not conform to the codebook, the transition taken by the PFA has a very low probability, or one or more PFAs time out. An active response is possible by terminating the violating process. Radmin allows for adaptation after the training phase: it can either continue to learn behavior during classification or be locked down, depending on the system policy.

The authors evaluate Radmin in a case study with several experiments using their own data under varying scenarios. These scenarios include the test of server applications, regular user applications, and utilities. Documented attacks are used to provoke malicious behavior on the test system. In order to report on the overhead of their detection method, the authors employ the commonly available UnixBench benchmark.

Note that in addition to the regular version of Radmin, a variant called URadmin was also developed by the authors. The latter operates solely in the user space, but is otherwise similar.

2.6 EYCM16 Inferring Software Component Interaction Dependencies for Adaptation Support (Esfahani et al., 2016)

- Anomaly Detection
- Server, Mac OS
- Design Driver, Server
- Agent Based
- Component Interaction
- Pre-Processing, Dynamic, Event Trace
- Pre-Processing, Dynamic, Itemset
- Pre-Processing, Association Rule Mining
- Conformance, Event
- Documented Attack, Simulated System

Esfahani et al. propose an approach for mining component interaction models in order to facilitate automated adaptations. In addition, they state that their models, generated through mining, can be used to detect abnormal behavior and, thus, malicious activity in the software system.

Since no representation of malicious behavior is used in this case, the approach is classified as anomaly detection. The architecture of the approach is based on the MAPE-K reference architecture by Kephart and Chess Kephart and Chess (2003). Since the authors imply that the targeted system consists of components that may be distributed, the behavior learning and monitoring system could be agent-based or distributed as well. However, the authors indicate that the monitoring system provides an interface for other components to write to a centralized event log, so the architecture is not fully distributed, according to the usage of the term “distributed” in the context of this publication. Since the authors indicate the distribution of monitoring components to monitored components which are distributed themselves, the approach is categorized as agent-based, with agents writing to a centralized log.

The authors approach, dubbed Mining Of Safe Adaptation Interveals for Component-based Software (MOSAIC), targets large scale software systems consisting of components that are in the scope of, e.g., Enterprise JavaBeans or web applications. A demonstrator is implemented and evaluated on a MacOS computer, however MacOS is not a driving factor behind the design.

In order to form component interaction profiles, the authors rely on a representation of component communication that needs to be monitored. The specifics depend on the form of communication between components, but examples are method calls or network message based component communication in an XML format. Here, we classify the monitored data as component interaction events in general, since the authors describe the monitored communication as events as well.

Once enough component communication has been logged by MOSAIC, it is possible to mine the event log to define a component interaction model. Note that the behavior of the event log is mostly normal, but may contain anomalous behavior as well. An event is defined as a message between to components. However, the authors aim to mine their rules based on transactions. Transactions are defined as the a start event, end event, and a number of events resulting from this transaction. For example, if component A sends a message to component B, a transaction is started. The transaction contains all following events, i.e. messages that were sent because of the transaction, until a response event is sent to component A. (Transactions encompass the event flow captured by activation boxes in UML2 sequence diagrams.) Top-level transactions, which are not invoked by other transactions, are equivalent to system use cases. From this point on, the approach works on transactions instead of events.

In the next pre-processing step, the approach forms itemsets from the transactions. An itemset is a set of transactions that occur temporally close to one another during the execution of the system. For each top-level transaction, an itemset is created, if it does not completely fall within another transaction. All other transactions are placed in the itemset that temporally surrounds them. From then on, timing information can be pruned, as it is not of interest any longer.

The next step is to derive rules, called Transaction Association Rules (TARs), from these itemsets. They contain the probability, with which a certain set of transactions implies the occurrence of another set of transactions. TARs are generated using a machine learning technique known as ”association rule mining”.

The derived TARs are then pruned, because the mining algorithm does not take domain knowledge into account. For example, ordering information of transactions is ignored during rule mining. In addition, the confidence level required for a TAR is set to be low, so the number of produced TARs is very large. The pruning is done according to heuristics that target redundant TARs.

The authors describe more data analysis steps for the purpose of system adaptation and reconfiguration. In this case, we are only interested in the anomaly detection capabilities and therefore omit these steps. The basic principle of the anomaly detection approach is simple: for a top-level transaction, the transactions occurring within them can be found by analyzing the timestamps of exchanged messages. Then, sequences of transactions are formed, so that each transaction is temporally contained in the previous transaction. If one of these sequences is not found in the rule-base, it can be assumed to be anomalous. Note that due to the large number of false positive when concurrency is high in the observed system, the authors devised a statistical method to mitigate this problem. It is not made clear by the authors if this approach is performed continuously on streams of component messages or if the log is analyzed in batches.

The anomaly detection capabilities of MOSAIC are evaluated in a case study on a simulated instance of a component-based software system. Normal system use cases were performed, in addition to injected attack scenarios.

2.6.0.1 GPW+15 LEAPS: Detecting Camouflaged Attacks with Statistical Learning Guided by Program Analysis Gu et al. (2015)

- Misuse Detection, Anomaly Detection
- System Events
- Benign, Malicious, Dynamic, Event Trace
- Benign, Malicious, Hierarchical Clustering
- Benign, Malicious, Dynamic, Control Flow Graph

- Benign, Malicious, Weighted SVM
- Batch, Conformance
- Batch, Weighted SVM
- Productive System
- Design Driver, Malware Injection
- Application, Windows

Gu et al. propose an intrusion detection approach based on dynamic program analysis. The approach targets camouflaged attacks, i.e., malware injected in to regular applications. The dynamic analysis is executed on both regular and infected versions of the software in order to gather training data.

Therefore, the approach is categorized both as anomaly and misuse detection, since both regular and malicious program behavior is used during the training phase.

The data for training and evaluation is collected by the authors on a productive Windows system from executing a variety of applications, such as Notepad++ or Putty. Since the authors target camouflaged attacks, the applications are injected with malware to generate appropriate training data for this context. The raw data collected by the authors are system events, collected using the "Event Tracing for Windows" framework.

Gu et al. have devised a multi-step process in order to format their data and train their detection model. Firstly, the collected system event log data is converted into an event trace comprised of system events correlated to function and library information for each process from a stack walk. Then, the resulting data is split into application and system stack traces.

The system stack trace is then hierarchically clustered in order to group functions and libraries, which is later used in the machine learning process.

A separate module infers control flow graphs from both normal and malicious application stack traces. Note that the malicious traces do not contain purely malicious behavior, but are interspersed with the regular application behavior. The result are two control flow graphs, one containing benign and the other mixed behavior. Then, the mixed and normal execution paths in the control flow graphs are compared in order to form a weighted dataset of seystem events.

The resulting dataset is then used to train a weighted support vector machine, using the benign system event dataset as the positive and the mixed weighted dataset as the negative samples.

The authors then use two separate techniques to detect attacks. First, they employ the benign and mixed call graphs in order to categorize system events into either benign or malicious, depending on which graph matches the stack trace more closely.

The second way to classify events is the weighted support vector machine, trained on the weighted dataset.

The authors use a batch analysis approach for their evaluation, noting that the weighted SVM shows much better performance than the call graph-based classification.

2.7 HEL+17 Watch Me, but Don't Touch Me! Contactless Control Flow Monitoring via Electromagnetic Emanations Han et al. (2017)

- Anomaly Detection

- Design Driver, PLC
- Control Channel Attacks
- Electromagnetic Emanations
- Benign, Spectrum Sequences
- Batch, LSTM
- Own Data

Han et al. propose an IDS for programmable logic controllers (PLC), which constitute the typical hardware used in industrial control systems. These controllers are often operating in saftey-critical contexts and do not tolerate overhead introduced by monitoring. Thus, Han et al. follow a contactless approach that utilizes the electromagnic emanations of the PLCs to ensure control flow integrity.

Their approach only depends on the binaries of benign executables, which makes it an Anomaly Detection based approach. They create the traces used for training by executing the binaries on the target hardware while attaching special sensors that monitor the corresponding electromagnetic emanations. They show that these emanations are sufficient to fingerprint a program. Furthermore, they ensure full coverage of the program during execution via symbolic execution and counterexample-guided inductive synthesis of the tests to be executed.

Given the traces of the benign programs, they conduct a preprocessing phase that creates so called Spectrum Sequences via a sliding-window-based approach. These sequences are then used to train a long term short memory neural network (LSTM), which is a well known technique for sequence classification. The LSTM model is then used to classify the traces in the deployment stage. We label this approach as Batch because Han et al. classifiy complete traces in their paper. It is unclear whether their approach is also capable of providing more timely feedback during the execution of a program.

For the evaluation, Han et al. classify the traces of ten benign programs. For the use-case of malicious execution detection they utilize two other programs. However, it is unclear, how they produced the malicious code.

2.8 IKHL18 Anomaly Detection Techniques Based on Kappa-Pruned Ensembles (Islam et al., 2018)

- Anomaly Detection
- Linux, Windows
- System Calls
- Benign, Hidden Markov Model
- Length, Hidden Markov Model
- Length, Boolean, Pruned
- Length, Hidden Markov Model
- ADFA-LD, CANALI-WD

Islam et al., with co-authorship from Khreich, propose a method for combining many detectors based on boolean combination, similar to Khreich et al. (2018). However, in this instance, they do not combine different types of detectors, but Hidden Markov Model (HMM) classifiers trained with different parameters. The approach is improved compared to the previous work by including a pruning step that removes redundant classifiers.

As with the previous works by Khreich et al., the general approach is anomaly detection and is evaluated on system call datasets from Linux and Windows environments. In the training phase, HMMs are trained using different parameters on benign system call sequences. The number of observation symbols and hidden states are fixed before training and influence the performance of the HMM. In addition, the initial distributions for the stochastic processes of the HMM, namely state transition, observation symbol, and state transition distribution also influence the training and later results. The authors train a large number of HMMs with different initial parameters.

The trained HMMs are then used to classify the validation set to find their true positive and false positive rates. Similar to the boolean combination approach employed by Khreich et al. (2018), the classifiers are evaluated in the Receiver Operating Characteristic (ROC) space. The described approach is also an iterative boolean combination algorithm, but uses the Kappa measure of disagreement between classifiers to eliminate redundant classifiers. Once the combination is computed, the combined HMMs can be used to classify sequential system call traces.

The authors evaluate their approach using the common CANALI-WD and ADFA-LD datasets, with the finding that the new combination algorithm improves on their previous efforts.

2.9 KKHT17 An Anomaly Detection System Based on Variable N-Gram Features and One-Class SVM (Khreich et al., 2017)

- Anomaly Detection
- Linux
- System Calls
- Benign, Dynamic, Feature Vector
- Benign, Support Vector Machine
- Length, Support Vector Machine
- Length, Dynamic, Feature Vector
- ADFA-LD

Khreich et al. propose a system call based anomaly detection approach, relying on an SVM classifier. It should be noted, that Khreich is co-author of three sources that are part of the selected sources in this systematic literature review. The approach is implemented on a Linux system, but does not explicitly target a certain context.

These system calls are first pre-processed to prepare the machine learning training phase. Particularly, the authors extract feature vectors from benign system calls. These feature vectors consist of sequences of system calls of varying sizes, weighted by their frequency of appearance in the trace used for the training. Moreover, the feature vectors retain the temporal information regarding the sequence of system calls.

The extracted features are then used in a second pre-processing step to train the one-class SVM. Lastly, the trained one-class SVM is used to classify system call traces. The approach allows for continuously classifying system calls in segments of a given length. The authors also present an alternative classification technique based on the Euclidean distance between the benign feature vectors and the currently observed features.

To evaluate both classification approaches, the authors provide a case study employing the common ADFA-LD dataset. They also compare them to other established techniques, namely Sequence Time-Delay Embedding (STIDE), an HMM based approach, and other n-gram approaches.

2.10 KMHT18 Combining Heterogeneous Anomaly Detectors for Improved Software Security (Khreich et al., 2018)

- Anomaly Detection
- Linux, Windows
- System Calls
- Benign, Dynamic, Tree
- Distance Based, Length
- Benign, Hidden Markov Model
- Length, Hidden Markov Model
- Frequency Based, Benign, Malicious
- Benign, One-class SVM
- Length, One-class SVM
- Length, Boolean
- ADFA-LD, CANALI-WD

Another approach by Khreich et al. combines different classifiers to improve anomaly detection performance by maximizing true positive and minimizing false positive classification results.

The approach combines STIDE, first introduced by Hofmeyr et al. (1998), an HMM detector, and a one-class SVM. The classifiers or detectors are combined using an iterative boolean combination algorithm to form a single anomaly detector. While not specifically targeted at a certain context, the approach is evaluated in both Linux and Windows environments, using appropriate system call datasets. The analysis techniques employed by the authors are manifold, since three different detectors are used, which require different pre-processing steps. Readying the STIDE detector requires the extraction of unique system calls sequences from a dataset of benign system calls (Hofmeyr et al., 1998). The extracted sequences are stored in the form of trees with a singular system call as the root to improve searchability. These trees make up the rule database STIDE uses. To improve the detection accuracy, the length of the system call sequences (the window size) is chosen appropriately using the performance on a validation set as an indicator.

In order to classify anomalous behavior, STIDE monitors system calls and checks the observed sequences for conformance to the normal system call sequences from the rule database. Khreich et al. employ the same methodology as Hofmeyr et al., who use the Hamming distance between sequences to determine the level of divergence, or lack thereof, between them (Hofmeyr et al., 1998).

The second detector is an HMM. It is also trained using benign system call data and the initial HMM parameters are estimated using the iterative Baum-Welch algorithm (Baum et al., 1970). After training, the HMM is ready to be used for classification.

The third detector is a one-class SVM, as in Khreich et al.'s previous work (Khreich et al., 2017). Firstly, features are extracted from the set of benign system calls, which are sequences weighted by their frequence of appearance.

The extracted features are then used to train the one-class SVM. Then, the one-class SVM is used to classify system call traces. Lastly, the results of all detectors are combined using an iterative boolean combination algorithm. The algorithm maximizes the detection accuracy by combining the results of the classifiers performing best in the ROC curve space. The ROC curve is a plot of the true positive rate compared with the false positive rate.

In their evaluation, the authors rely on commonly available system call datasets, collected from both Windows and Linux environments, namely the ADFA-LD and the CANALI-WD datasets.

2.11 LL19 Data-driven Anomaly Detection with Timing Features for Embedded Systems (Lu and Lysecky, 2019)

- Anomaly Detection
- Design Driver, Embedded System
- Malware
- Centralized
- System Calls, Function Calls, Instructions, Interrupts
- Benign, Dynamic, Timing Model
- Conformance, Event
- Benign, Hierarchical Clustering
- Conformance, Event
- Benign, Dynamic, Subcomponent Time Model
- Conformance, Event
- Benign, Hierarchical Clustering
- Distance Based, Event
- Benign, One-class SVM
- Event, One-class SVM
- Own Data

Lu and Lysecky present an approach for embedded systems that makes use of timing features as an addition to an already existing sequence based approach. The main contribution of this publication are the introduction and comparison of several models for the representation of benign system behavior. The detection itself is done via a conformance or distance-based check to these models, or by using an One-class SVM trained via benign events. This makes it an anomaly detection based approach.

Due to the tight time constraints of the targeted real-time systems, the authors argue for a centralized hardware-based implementation of their approach. The monitoring is done via the processor trace port of the system under consideration. For the prototypical implementation of their approach the authors utilize system/function calls, instructions, and interrupts.

The first introduced model is a lumped (system-wide) timing model that represents a single range from best to worst case execution time per event. The corresponding classification at runtime simply checks whether the timing of the currently monitored event falls within this range. In a refinement of this approach the single ranges are further separated via a hierarchical clustering approach.

As an additional approach a time model is constructed for subcomponents of the timing (e.g. instruction cache misses). Again, the authors elaborate on conformance-based classification and a further refinement via hierarchical clustering. Furthermore, the authors use a one-class SVM in the context of the subcomponent timing model.

For the evaluation the authors developed FPGA-based prototypes for an unmanned aerial vehicle and a network-connected pacemaker. Furthermore, they developed different types of malware that, e.g., manipulate encrypted data or adds noise to images. They compare the detection accuracy of their techniques with another known technique, i.e., SecureCore Yoon et al. (2013).

2.12 MAEG14 Generating profile-based signatures for online intrusion and failure detection (Masri et al., 2014)

- Misuse Detection
- Application, Server, Java, Exploit Vulnerability
- Basic Block, Basic Block Branch, Method Calls, Variable Definition, Variable Use
- Benign, Malicious, Dynamic, Event Trace
- Benign, Malicious, Dynamic, Itemset
- Benign, Malicious, Genetic Algorithm
- Conformance, Passive, Event
- Known Vulnerability, Testing

Masri et al. propose a signature based approach to intrusion detection. They derive vulnerability signatures, which characterize behavior that correlates with program vulnerability exploitation, using a genetic algorithm. Since the signatures contain potentially malicious behavior, the approach can be characterized as misuse detection. A commonality with anomaly detection approaches is the usage of normal program execution data during the signature derivation. The authors themselves refer to it as a vulnerability signature based approach.

The approach targets program vulnerabilities and their exploits. It is evaluated on a Java platform using server and utility applications, however, the authors state that the approach should translate to other programming languages or compiled binaries as well.

In order to obtain the vulnerability signatures and later perform online monitoring of a program, the approach monitors elements of an instrumented program, namely: method calls, variable definitions, variable uses, basic blocks, and basic block edges. In previous work, the authors have devised their own profiling tool for Java to obtain these elements Masri et al. (2006, 2007); Masri and Podgurski (2009); Masri and Halabi (2011).

The first step is to create a training set, i.e., parameters for the execution of the program, that adequately represent the normal and malicious behavior of the targeted program. To obtain the training set, the authors propose to use existing collections of test cases for the program and to augment the cases with parameters that would be used to exploit a vulnerability. This pre-processing step therefore requires benign and malicious data.

Once the training set is established, it is used to execute the program. During this step, an execution profile per test case is generated dynamically for the application. The execution profile is a simple model

that captures only if a certain program element occurred during a malicious or benign test case. Frequency and sequentiality of events is disregarded.

The genetic algorithm devised by the authors for this approach then generates signatures of vulnerability exploits from the execution profiles. The program elements that correlate most with the attacks witnessed during training are used as attack signatures. This process results in one signature per vulnerability, which are later combined for each application.

During the classification step, a target system is monitored using the aforementioned Java instrumentation developed by the authors. The monitored program elements are checked for conformance to the vulnerability signatures. If a current program element is found in a signature, it is flagged. Then, when all elements of a signature are flagged, an alarm is executed since that would indicate vulnerability exploitation behavior.

The authors evaluate their approach using seven server and utility applications written in Java. Test cases are used to establish normal behavior and known vulnerabilities are used to add malicious training data.

2.13 MC19 Adaptive Security Monitoring for Next-Generation Routers (Mansour and Chasaki, 2019)

- Anomaly Detection
- Design Driver, Embedded System, Protocol Processing
- Centralized
- Instructions
- Benign, Static, Hash Value
- Conformance, Event
- Benign, Static, Nondeterministic Finite Automaton
- Conformance, Substitute Version, Event
- Own Data

Mansour and Chasaki propose an anomaly detection based approach for network processors of next-generation routers. To this end, they introduce a monitoring subsystem that aims to detect attacks that change the processing behavior of these processors.

We classify the architecture as being centralized because their hardware monitor, which is not exclusively monitoring but also performing the analysis, is meant to be a subsystem of the router. The authors' prototypical implementation is based on an FPGA.

Generally the approach is based on the idea of software diversity, where several semantically equivalent versions of a certain protocol implementation exist. The authors generate several versions of the corresponding binaries in an offline phase. Additionally, they generate a hash value for each version and a monitoring graph in the form of a nondeterministic finite automaton utilizing a static analysis. Here, the states of the automaton represent basic blocks.

At runtime, the hash value is checked when the executable is loaded on the network processor. This realizes an integrity check of the binary. Furthermore, the hardware monitor checks the flow of instructions against the monitoring graph. If an anomaly is detected, another version of the protocol is loaded on the network processor. Here, the authors benefit from the fact that the protocols that they target are stateless and unreliable. Thus, there is no need for a state transfer and packets can be dropped when substituting versions. In their evaluation the authors focus on the resource consumption and performance of their design

in comparison with two other approaches taken from literature Ragel and Parameswaran (2006); Arora et al. (2005).

2.14 RRL+12 Intrusion Detection for Resource-Constrained Embedded Control Systems in the Power Grid (Reeves et al., 2012)

- Anomaly Detection
- Design Driver, Embedded System, Linux, Rootkit, Pointer Hijacking
- Function Calls
- Benign, Dynamic, Itemset
- Conformance, Passive, Event
- Documented Attack

Reeves et al. propose an intrusion detection approach for embedded devices in the power grid. They intend to find control flow hooking rootkits and pointer hijacking attacks using anomaly detection. Specifically, the approach targets exploits in which a malicious function is placed between a function pointer and the original function it pointed to. The malicious function will call the original function at some point, so as to make the behavior seem normal while actually being compromised. The approach is implemented on a Linux operating system. To identify the targeted control flow alterations, function calls need to be monitored.

During the learning phase, the Intrusion Detection System (IDS) scans the kernel for potential function pointers, on which probes (KProbes) are placed. Then, the system is exercised by executing as much normal behavior as possible using tests or specific use cases. The result is a list of functions that were called by a function pointer. These functions are tagged, in order to place KProbes on them for the detection phase. Now, the IDS can check if the called functions conform to the called functions collected in the learning phase. If an anomaly in the called functions is found, the user can be alerted immediately or the anomaly is logged.

The authors evaluate their approach in a case study using documented attacks to test the effectiveness of the system. For measuring the overhead of the approach, the authors use common benchmarks.

2.15 SYRJ17 Long-Span Program Behavior Modeling and Attack Detection (Shu et al., 2017)

- Anomaly Detection
- Server, Utility, Linux
- Design Driver, Data Oriented Programming
- System Calls, Function Calls
- Benign, Dynamic, Call Graph
- Benign, Agglomerative Clustering
- Passive, Batch, Agglomerative Clustering
- Benign, Support Vector Machine
- Frequency Based, Batch
- Manual Interaction, Productive System

Shu et al. present an approach for the detection of stealthy attacks in long system call traces, which uses machine learning techniques to correlate events occurring with significant time between them.

The main goal of this approach is detecting attacks that indirectly alter the application control flow, such as data oriented programming attacks (Hu et al. (2016)), or exploit legal control flows, such as denial of service attacks. As a basis for the approach, they propose a context sensitive grammar. The general approach is classified as anomaly detection, which they evaluate on a Linux system using both server and utility applications.

In order to realize this approach, the authors propose monitoring system events such as jumps, function calls, or generic instructions in program traces. However, their own implementation that is evaluated in the source uses system and function calls, depending on the application that is monitored.

Two main phases make up the anomaly detection procedure in this case: training and detection. Both phases are based on the "behavior profiling" step, which parts raw program traces into trace segments. The segments are used to create matrices indicating event co-occurrence and event transition frequency, i.e., how often certain routines occur together and the frequency of calls from one routine to another. In order to name the routines, the authors employ static analysis, with the matrices basically representing a dynamic call graph.

After this step, the training is started using an agglomerative clustering algorithm to identify clusters of normal behavior. Then, the first detection step checks whether a behavior instance fits into one of the normal clusters by checking the co-occurred events of the behavior instance and the cluster, if they match, the behavior instance fits. If no fitting cluster can be found, an alarm is raised, otherwise the instance is passed to a occurrence frequency analysis step.

The occurrence frequency analysis is based on a second training step called intra-cluster modeling. Intra-cluster modeling uses a deterministic and a probabilistic method to derive a refinement of the boundary for normal behavior within the clusters from the co-occurrence frequency of events. The probabilistic methods is an SVM, while the deterministic method employs variable range analysis. The detection step that follows examines the behavior instances for quantitative frequency relational anomalies. If the frequencies do not fit the models derived int the intra-cluster modeling step, the instance is reported as anomalous.

In their paper, the authors evaluate their approach using a prototypical implementation on a Linux platforms. They use the applications sendmail, sshd, and libpcre for testing. Depending on the application, training data is gathered either through monitoring a productive system or specific manual interaction.

2.16 TA19 RAMD: Registry-based Anomaly Malware Detection using One-Class Ensemble Classifiers (Tajoddin and Abadi, 2019)

- Anomaly Detection
- Design Driver, Windows, Malware
- Centralized
- Registry Events
- Benign, Dynamic, Feature Vector
- Benign, Gaussian Classifier
- Benign, K-Means
- Benign, K-Nearest-Neighbor
- Benign, Parzen Window

- Batch, Fibonacci-based superincreasing ordered weighted averaging, Memetic Firefly-Based Ensemble Classifier Pruning
- Productive System, Known Vulnerability

Tajoddin and Abadi present a solution for detecting malware on the Windows operating system using an ensemble of one-class classifiers. These classifiers are trained using benign behavior data, meaning that the approach is classified as anomaly detection. Tajoddin and Abadi focus on single systems, thus employing a centralized architecture.

They monitor access to the Windows Registry, which is a unique component of the operating systems, clearly setting the focus of their approach. In particular, they identify registry operations and keys which are especially security relevant.

In order to detect malicious behavior in registry access, Tajoddin and Abadi use multiple analysis techniques sequentially. First, they generate feature vectors from logs of accesses to security sensitive registry keys.. The logs are generated by running benign programs. Then, the set of features is randomly divided into subsets, each of which is used to train a one-class classifier. At this stage, the authors use four different classifiers, namely a Gaussian classifier, K-Means and K-Nearest-Neighbor algorithms, as well as a Parzen window classifier. These trained classifiers are then combined into an ensemble classifier.

In order to keep only the most accurate classifiers in the ensemble, the authors employ two pruning and optimization techniques: first, an algorithm called memetic firefly-based ensemble classifier pruning and then the Fibonacci-based superincreasing ordered weighted averaging aggregation operator (FSOWA). The firefly-based pruning approach results in an optimized ensemble classifier, which contains only the most accurate classifiers. Then, in the detection step, the FSOWA operator assigns weights to the classifiers in the ensemble, according to the classifier's confidence in its decision. The detection step is performed in a batch style, extracting the same feature vectors from logs as in the training phase.

The evaluation of the system is performed in the form of a case study, in which known malware is run on a system under test. This mirrors the training phase of the ensemble classifier, with the difference that the data is benign or malicious, since both malware and benign software is executed.

2.17 XZTZ16 Unifying Intrusion Detection and Forensic Analysis via Provenance Awareness (Xie et al., 2016)

- Anomaly Detection
- Design Driver, Exploit Vulnerability
- Application, Linux
- System Calls, Process Attributes, Socket Attributes, File Attributes
- Benign, Dynamic, Provenance Graph
- Benign, Dynamic, Signature
- Conformance, Self Adaptive, Length
- Batch, Dynamic, Provenance Graph
- Documented Attack, Manual Interaction

Xie et al. present an approach that includes intrusion detection and supports forensic analysis. The system is dubbed *PIDAS*, which stands for *Provenance aware Intrusion Detection and Analysis System*.

Their anomaly detection approach is based on the provenance of system objects, i.e., the history of files, processes, or sockets, to represent data flows and dependency relationships.

The general goal of the approach is to detect exploits that target process vulnerabilities. In their case study, the authors evaluate the approach on a Linux system using server and general purpose applications.

In order to capture the provenance, or history of system objects, the approach uses an existing provenance aware storage system developed by Muniswamy-Reddy et al., aptly named *PASS* Muniswamy-Reddy et al. (2006). PASS monitors files, processes, and sockets by collecting system calls pertaining to these objects and metadata regarding the objects, e.g., create time, creator, and the source in the case of networked objects.

The first phase in the analysis of a system using PIDAS is building the provenance graph by running PASS on the target system and executing benign programs. In doing so, PASS builds a directed acyclic graph representing the relationships and history between objects, capturing the representation of normal system behavior.

Once the provenance database has been established, the detector component of PIDAS extracts rules from this database by statistically analyzing the events occurring the most during normal system execution. The rules are in the form of dependencies, e.g., a process creating a file would in indicate a dependency from the process to the file.

The classification of the monitored system is performed by organizing the dependencies in a graph. That way, event segments captured on the target system can be searched for in the dependency database using depth first search.

The system calculates a likelihood for the monitored segment to be normal or abnormal, depending on how much of the segment conforms to the dependency database. The length of the segments and the threshold for abnormality can be set by a system administrator. During the classification stage, the system allows for self adaptation if a segment that is not part of the rule database is deemed normal. If abnormal behavior is found during the classification stage, a warning report is generated to allow the system administrator to react accordingly.

In addition to the monitoring functionality, PIDAS supports system administrators by providing an analyzer that generates graphs representing the events before or after suspected intrusion. The analyzer is another classification step that can operate forwards and backwards in time on batches of logs. The authors evaluate their approach by providing a case study using data created by interacting with the system under test and then submit it to known attacks.

2.18 ZHB+13 Secloud: A Cloud-Based Comprehensive and Lightweight Security Solution for Smartphones (Zonouz et al., 2013)

- Misuse Detection, Anomaly Detection
- Android
- Agent Based
- System Calls, Physical Input, File Events, File Attributes
- Conformance, Process Termination, Periodic Backup, File Removal, Network Filter, Quarantine, Event
- Conformance, Process Termination, Periodic Backup, File Removal, Network Filter, Quarantine, Time

- Conformance, Process Termination, Periodic Backup, File Removal, Network Filter, Quarantine, Length
- Conformance, Process Termination, Periodic Backup, File Removal, Network Filter, Quarantine, Event
- Documented Attack

Zonouz et al. propose Secloud, an intrusion detection approach for smartphones, which lightens the monitoring load on the device itself by performing the different analyses in a cloud computing environment. The basic idea is to create a carbon copy of the smartphone operating system and software, run it in a virtualized environment in the cloud, and performing intrusion detection on the virtual smartphone so as to not negatively affecting the real smartphones performance. Consistency is ensured by using an agent that is deployed on the real smartphone to record all inputs and mirroring these inputs on the simulated smartphone.

The authors employ a variety of analysis techniques, comprising both misuse and anomaly detection. Since the software on the monitored smartphone is only providing information for the simulated cloud smartphone and all the analysis steps are performed in the cloud, the architecture is agent-based with the agent being deployed on the smartphone and the centralized analysis located in the cloud computing environment. The authors do not target specific threats, but mobile devices and smartphones in genreal. Their prototype is implemented on and Android phone, but should be applicable to other mobile operating systems as well. The requirement for the smartphone operating system is, that physical and other inputs need to be monitored on the smartphone for synchronizing the two versions. In the cloud environment, the approach requires monitoring system calls, files and corresponding events or attributes, as well as network traffic.

The collected data is analyzed by a variety of detectors. The detectors are a classical malware detector (ClamAV[2]), the file integrity monitoring solution SAMHAIN[3], the Network-Based Intrusion Detection System (NIDS) Snort[4], and the system call anomaly detection approach detailed by Forrest et al. in multiple publications (Forrest et al., 2008; Hofmeyr et al., 1998). Since the authors do not detail the pre-processing steps entailed in order to use the different detectors, we only describe the classification methods of the approaches.

The virus checker ClamAV operates on a database of malware behaviors. It checks if currently observed system events conform to a database entry. If this is the case, the malware is reported.

The second analysis technique is checking the integrity of files using SAMHAIN. SAMHAIN checks the integrity of files every 20 seconds in this instance; it is therefore a continuous approach with fixed time intervals. The goal of this analysis technique is to find out if any files have been tampered with, which could imply malicious behavior.

Thirdly, the authors employ system call-based anomaly detection as developed by Forrest et al. (Forrest et al., 2008; Hofmeyr et al., 1998). This approach allows for a variety of machine learning or other classifiers, however, Zonouz et al. do not detail which is used in this specific case. They could, e.g., use an HMM like Khreich et al. (2018) in their implementation of the approach by Forrest et al.. Classification is performed on sequences of system calls of a certain length which are checked for conformance to a database containing the normal system behavior.

[2] https://www.clamav.net/

[3] la-samhna.de/samhain/

[4] www.snort.org/

The last analysis technique is the network-based IDS Snort. Snort monitors network traffic and uses signatures of known malicious behavior to detect intrusions. It monitors on a per event basis.

When an intrusion is detected, Secloud has the ability to respond passively or actively. The passive response would be a notification or Email to the smartphone user. Active response can be the removal of infected files, terminating the malicious process, perform a backup or restoration, filtering network traffic, and quarantining the device to prevent it from affecting others. Secloud is evaluated in a case study using known attacks to examine its capability of detecting them.

2.19 ZKS15 Detection of Anomalies in behavior of the Software with Usage of Markov Chains (Zegzhda et al., 2015)

- Anomaly Detection
- Linux, Windows
- System Calls
- Benign, Markov Model
- Length, Markov Model
- Own Data

Zegzhda et al. propose an anomaly detection approach employing markov chains. The approach does not designed to target specfic vulnerabilities, but has been evaluated on both Linux and Windows systems running various applications. In order to detect anomalies, the system monitors system calls to capture the targeted applications behavior.

A collection of benign system calls executed by the targeted applications is used to train a markov chain, in which the each state's dimensionality is dependent of the windows size, i.e., the number of system calls that are examined as one segment. This markov chain is then used to classify segments collected from a system running the targeted applications. In a case study, the authors demonstrate the effectiveness of their classifier on data they generated themselves.

2.20 ZRJ14 A Defense Framework Against Malware and Vulnerability Exploits (Zhang et al., 2014)

- Misuse Detection, Anomaly Detection
- Design Driver, Malware, Exploit Vulnerability
- Linux
- Hybrid
- Basic Block, Basic Block Branch, Method Calls
- Pre-Processing, Static, Event Trace
- Pre-Processing, Dynamic, Event Trace
- Pre-Processing, Dynamic, Control Flow Graph
- Conformance, Self Adaptive, Program Suspension, Event
- Documented Attack, Manual Interaction, Known Vulnerability

Zhang et al. propose an IDS framework, in which a developer can submit an application to a publishing entity, where it is tested under user provided security policies. If the application is deemed secure, a behavioral model is built and sent, along with the application, to the user. The user can then run the

application and monitor it for conformance to the secure behavioral model. The basic approach the authors employ is two fold. When examining the application, the publishing entity enforces security policies which disallow behavior that is typical of malware. This approach entails defining malware behavior signatures and therefore is labeled as misuse detection.

The monitoring component on the user side, however, checks adherence to a behavior model that is deemed secure, which means that the run time monitoring component follows an anomaly detection approach.

The architecture in this case is hybrid. The publisher is centralized and performs the initial examination of an application for malicious behavior. However, the monitoring is then performed strictly on the user host, which can operate independently, once it has received the application and behavioral model.

Zhang et al.'s proposed framework targets applications containing malware and vulnerabilities for exploits. It is evaluated on Linux systems running on an Intel platform.

For data monitoring during run time, the application under scrutiny is instrumented using the Intel "Pin" dynamic binary instrumentation tool[5]. The tool monitors application instructions, called execution trace by the authors, and searches for matches in the basic blocks of the behavioral model.

The sequence of analysis techniques used in this approach are described in the following. First, disassembled binaries are processed to resolve loop unrolling and function calls. Then, symbolic propagation analysis finds input-dependent paths.

For each instruction, the analysis checks if it depends on a symbolic input, if so, it is resolved using the "simple theorem prover" conditional formula (Ganesh and Dill, 2007). Then, all path conditions are checked for satisfiability. If they are satisfiable, a new input sequence is generated that triggers a new path execution. It is crucial that a majority of code is covered by the sequences generated in this step. If the input sequence does not cover enough execution paths, the input sequence is augmented by user provided inputs.

In the next step, untrusted applications are instrumented to trace their execution. The monitored execution information is transformed to an execution trace divided into basic blocks by control flow transfer instructions. Traces of different executions are joined using an algorithm detailed by the authors. Invariants found on an execution path in one execution are updated in other execution traces that share the same path.

Then, the traces are checked according to security policies. Security policies are defined as sequences of actions or properties contained in basic blocks. If an execution trace contains the exact sequence defined by the security policy, a violation is reported and testing is aborted. If all paths are cleared using the security policies, the behavioral model can be built. It contains only blocks related to security policies such as function calls, memory allocation components, function returns etc and invariants such as acceptable return addresses, base pointer address, function pointer values. The behavioral model contains the security policies embedded as flags.

Having built the execution model, the user can monitor application behavior during run-time. In order to build the user's execution trace, the application is executed under Intel's "Pin" tool. This user execution trace is delineated by basic blocks, like the model derived in the pre-processing phases. If the users's trace conforms to the security model, the application is executed as normal, otherwise, execution is suspended. In the case of execution paths occurring in the user's execution that are not present in the model, restrictive

[5] https://software.intel.com/en-us/articles/pin-a-binary-instrumentation-tool-papers

security policies are enforced and the program is suspended when the policies are violated. If this newly discovered path is deemed safe, it may be added to the behavioral model.

The proposed framework is evaluated in a case study, using applications known to be malicious, applications containing obfuscated malicious behavior, programs with documented vulnerabilities, and known benign programs to check for false positives. Both the publisher and the user side were evaluated in the study.

3 PIPELINES

This section introduces the pipelines configured by the AutoML runs on which we report in our main paper. We simply stay with the format given by the corresponding framework.

3.1 Auto-sklearn One-hot-encoded

Listing 1. Pipeline resulting from the auto-sklearn run on encoded data

```
(0.260000,
SimpleClassificationPipeline({
        'balancing:strategy': 'weighting',
        'categorical_encoding:__choice__': 'one_hot_encoding',
        'classifier:__choice__': 'random_forest',
        'imputation:strategy': 'median',
        'preprocessor:__choice__': 'truncatedSVD',
        'rescaling:__choice__': 'standardize',
        'categorical_encoding:one_hot_encoding:use_minimum_fraction': 'True',
        'classifier:random_forest:bootstrap': 'False',
        'classifier:random_forest:criterion': 'entropy',
        'classifier:random_forest:max_depth': 'None',
        'classifier:random_forest:max_features': 0.6471330546448919,
        'classifier:random_forest:max_leaf_nodes': 'None',
        'classifier:random_forest:min_impurity_decrease': 0.0,
        'classifier:random_forest:min_samples_leaf': 2,
        'classifier:random_forest:min_samples_split': 2,
        'classifier:random_forest:min_weight_fraction_leaf': 0.0,
        'classifier:random_forest:n_estimators': 100,
        'preprocessor:truncatedSVD:target_dim': 31,
  'categorical_encoding:one_hot_encoding:minimum_fraction': 0.001155931999909784},
dataset_properties={
        'task': 1,
        'sparse': True,
        'multilabel': False,
        'multiclass': False,
        'target_type': 'classification',
        'signed': False})),
(0.180000,
SimpleClassificationPipeline({
        'balancing:strategy': 'none',
        'categorical_encoding:__choice__': 'no_encoding',
        'classifier:__choice__': 'random_forest',
        'imputation:strategy': 'mean',
        'preprocessor:__choice__': 'truncatedSVD',
        'rescaling:__choice__': 'standardize',
        'classifier:random_forest:bootstrap': 'False',
```

```
        'classifier:random_forest:criterion': 'gini',
        'classifier:random_forest:max_depth': 'None',
        'classifier:random_forest:max_features': 0.5625796853303024,
        'classifier:random_forest:max_leaf_nodes': 'None',
        'classifier:random_forest:min_impurity_decrease': 0.0,
        'classifier:random_forest:min_samples_leaf': 1,
        'classifier:random_forest:min_samples_split': 8,
        'classifier:random_forest:min_weight_fraction_leaf': 0.0,
        'classifier:random_forest:n_estimators': 100,
        'preprocessor:truncatedSVD:target_dim': 49},
dataset_properties={
        'task': 1,
        'sparse': True,
        'multilabel': False,
        'multiclass': False,
        'target_type': 'classification',
        'signed': False})),
(0.140000,
SimpleClassificationPipeline({
        'balancing:strategy': 'weighting',
        'categorical_encoding:__choice__': 'no_encoding',
        'classifier:__choice__': 'random_forest',
        'imputation:strategy': 'median',
        'preprocessor:__choice__': 'truncatedSVD',
        'rescaling:__choice__': 'standardize',
        'classifier:random_forest:bootstrap': 'False',
        'classifier:random_forest:criterion': 'gini',
        'classifier:random_forest:max_depth': 'None',
        'classifier:random_forest:max_features': 0.6471330546448919,
        'classifier:random_forest:max_leaf_nodes': 'None',
        'classifier:random_forest:min_impurity_decrease': 0.0,
        'classifier:random_forest:min_samples_leaf': 2,
        'classifier:random_forest:min_samples_split': 2,
        'classifier:random_forest:min_weight_fraction_leaf': 0.0,
        'classifier:random_forest:n_estimators': 100,
        'preprocessor:truncatedSVD:target_dim': 26},
dataset_properties={
        'task': 1,
        'sparse': True,
        'multilabel': False,
        'multiclass': False,
        'target_type': 'classification',
        'signed': False})),
(0.080000,
SimpleClassificationPipeline({
        'balancing:strategy': 'none',
        'categorical_encoding:__choice__': 'no_encoding',
        'classifier:__choice__': 'random_forest',
        'imputation:strategy': 'mean',
        'preprocessor:__choice__': 'truncatedSVD',
        'rescaling:__choice__': 'standardize',
        'classifier:random_forest:bootstrap': 'False',
        'classifier:random_forest:criterion': 'entropy',
        'classifier:random_forest:max_depth': 'None',
```

```
        'classifier:random_forest:max_features': 0.5625796853303024,
        'classifier:random_forest:max_leaf_nodes': 'None',
        'classifier:random_forest:min_impurity_decrease': 0.0,
        'classifier:random_forest:min_samples_leaf': 5,
        'classifier:random_forest:min_samples_split': 8,
        'classifier:random_forest:min_weight_fraction_leaf': 0.0,
        'classifier:random_forest:n_estimators': 100,
        'preprocessor:truncatedSVD:target_dim': 49},
dataset_properties={
        'task': 1,
        'sparse': True,
        'multilabel': False,
        'multiclass': False,
        'target_type': 'classification',
        'signed': False})),
(0.060000,
SimpleClassificationPipeline({
        'balancing:strategy': 'none',
        'categorical_encoding:__choice__': 'no_encoding',
        'classifier:__choice__': 'random_forest',
        'imputation:strategy': 'most_frequent',
        'preprocessor:__choice__': 'truncatedSVD',
        'rescaling:__choice__': 'standardize',
        'classifier:random_forest:bootstrap': 'False',
        'classifier:random_forest:criterion': 'entropy',
        'classifier:random_forest:max_depth': 'None',
        'classifier:random_forest:max_features': 0.5746378932571852,
        'classifier:random_forest:max_leaf_nodes': 'None',
        'classifier:random_forest:min_impurity_decrease': 0.0,
        'classifier:random_forest:min_samples_leaf': 5,
        'classifier:random_forest:min_samples_split': 6,
        'classifier:random_forest:min_weight_fraction_leaf': 0.0,
        'classifier:random_forest:n_estimators': 100,
        'preprocessor:truncatedSVD:target_dim': 49},
dataset_properties={
        'task': 1,
        'sparse': True,
        'multilabel': False,
        'multiclass': False,
        'target_type': 'classification',
        'signed': False})),
(0.060000,
SimpleClassificationPipeline({
        'balancing:strategy': 'none',
        'categorical_encoding:__choice__': 'no_encoding',
        'classifier:__choice__': 'random_forest',
        'imputation:strategy': 'mean',
        'preprocessor:__choice__': 'truncatedSVD',
        'rescaling:__choice__': 'standardize',
        'classifier:random_forest:bootstrap': 'False',
        'classifier:random_forest:criterion': 'entropy',
        'classifier:random_forest:max_depth': 'None',
        'classifier:random_forest:max_features': 0.5625796853303024,
        'classifier:random_forest:max_leaf_nodes': 'None',
```

```
        'classifier:random_forest:min_impurity_decrease': 0.0,
        'classifier:random_forest:min_samples_leaf': 5,
        'classifier:random_forest:min_samples_split': 7,
        'classifier:random_forest:min_weight_fraction_leaf': 0.0,
        'classifier:random_forest:n_estimators': 100,
        'preprocessor:truncatedSVD:target_dim': 49},
dataset_properties={
        'task': 1,
        'sparse': True,
        'multilabel': False,
        'multiclass': False,
        'target_type': 'classification',
        'signed': False})),
(0.040000,
SimpleClassificationPipeline({
        'balancing:strategy': 'none',
        'categorical_encoding:__choice__': 'one_hot_encoding',
        'classifier:__choice__': 'random_forest',
        'imputation:strategy': 'mean',
        'preprocessor:__choice__': 'truncatedSVD',
        'rescaling:__choice__': 'standardize',
        'categorical_encoding:one_hot_encoding:use_minimum_fraction': 'True',
        'classifier:random_forest:bootstrap': 'False',
        'classifier:random_forest:criterion': 'entropy',
        'classifier:random_forest:max_depth': 'None',
        'classifier:random_forest:max_features': 0.61862536409033,
        'classifier:random_forest:max_leaf_nodes': 'None',
        'classifier:random_forest:min_impurity_decrease': 0.0,
        'classifier:random_forest:min_samples_leaf': 2,
        'classifier:random_forest:min_samples_split': 7,
        'classifier:random_forest:min_weight_fraction_leaf': 0.0,
        'classifier:random_forest:n_estimators': 100,
        'preprocessor:truncatedSVD:target_dim': 37,
        'categorical_encoding:one_hot_encoding:minimum_fraction': 0.0002947219194808727},
dataset_properties={
        'task': 1,
        'sparse': True,
        'multilabel': False,
        'multiclass': False,
        'target_type': 'classification',
        'signed': False})),
(0.040000,
SimpleClassificationPipeline({
        'balancing:strategy': 'weighting',
        'categorical_encoding:__choice__': 'no_encoding',
        'classifier:__choice__': 'random_forest',
        'imputation:strategy': 'most_frequent',
        'preprocessor:__choice__': 'truncatedSVD',
        'rescaling:__choice__': 'standardize',
        'classifier:random_forest:bootstrap': 'False',
        'classifier:random_forest:criterion': 'gini',
        'classifier:random_forest:max_depth': 'None',
        'classifier:random_forest:max_features': 0.6471330546448919,
        'classifier:random_forest:max_leaf_nodes': 'None',
```

```
        'classifier:random_forest:min_impurity_decrease': 0.0,
        'classifier:random_forest:min_samples_leaf': 2,
        'classifier:random_forest:min_samples_split': 2,
        'classifier:random_forest:min_weight_fraction_leaf': 0.0,
        'classifier:random_forest:n_estimators': 100,
        'preprocessor:truncatedSVD:target_dim': 26},
dataset_properties={
        'task': 1,
        'sparse': True,
        'multilabel': False,
        'multiclass': False,
        'target_type': 'classification',
        'signed': False})),
(0.040000,
SimpleClassificationPipeline({
        'balancing:strategy': 'none',
        'categorical_encoding:__choice__': 'no_encoding',
        'classifier:__choice__': 'random_forest',
        'imputation:strategy': 'mean',
        'preprocessor:__choice__': 'truncatedSVD',
        'rescaling:__choice__': 'standardize',
        'classifier:random_forest:bootstrap': 'False',
        'classifier:random_forest:criterion': 'entropy',
        'classifier:random_forest:max_depth': 'None',
        'classifier:random_forest:max_features': 0.5710064798914133,
        'classifier:random_forest:max_leaf_nodes': 'None',
        'classifier:random_forest:min_impurity_decrease': 0.0,
        'classifier:random_forest:min_samples_leaf': 5,
        'classifier:random_forest:min_samples_split': 9,
        'classifier:random_forest:min_weight_fraction_leaf': 0.0,
        'classifier:random_forest:n_estimators': 100,
        'preprocessor:truncatedSVD:target_dim': 33},
dataset_properties={
        'task': 1,
        'sparse': True,
        'multilabel': False,
        'multiclass': False,
        'target_type': 'classification',
        'signed': False})),
(0.020000,
SimpleClassificationPipeline({
        'balancing:strategy': 'none',
        'categorical_encoding:__choice__': 'no_encoding',
        'classifier:__choice__': 'random_forest',
        'imputation:strategy': 'most_frequent',
        'preprocessor:__choice__': 'truncatedSVD',
        'rescaling:__choice__': 'standardize',
        'classifier:random_forest:bootstrap': 'False',
        'classifier:random_forest:criterion': 'entropy',
        'classifier:random_forest:max_depth': 'None',
        'classifier:random_forest:max_features': 0.5625796853303024,
        'classifier:random_forest:max_leaf_nodes': 'None',
        'classifier:random_forest:min_impurity_decrease': 0.0,
        'classifier:random_forest:min_samples_leaf': 5,
```

```
        'classifier:random_forest:min_samples_split': 8,
        'classifier:random_forest:min_weight_fraction_leaf': 0.0,
        'classifier:random_forest:n_estimators': 100,
        'preprocessor:truncatedSVD:target_dim': 49},
dataset_properties={
        'task': 1,
        'sparse': True,
        'multilabel': False,
        'multiclass': False,
        'target_type': 'classification',
        'signed': False})),
(0.020000,
SimpleClassificationPipeline({
        'balancing:strategy': 'none',
        'categorical_encoding:__choice__': 'no_encoding',
        'classifier:__choice__': 'random_forest',
        'imputation:strategy': 'mean',
        'preprocessor:__choice__': 'truncatedSVD',
        'rescaling:__choice__': 'standardize',
        'classifier:random_forest:bootstrap': 'False',
        'classifier:random_forest:criterion': 'entropy',
        'classifier:random_forest:max_depth': 'None',
        'classifier:random_forest:max_features': 0.5625796853303024,
        'classifier:random_forest:max_leaf_nodes': 'None',
        'classifier:random_forest:min_impurity_decrease': 0.0,
        'classifier:random_forest:min_samples_leaf': 7,
        'classifier:random_forest:min_samples_split': 7,
        'classifier:random_forest:min_weight_fraction_leaf': 0.0,
        'classifier:random_forest:n_estimators': 100,
        'preprocessor:truncatedSVD:target_dim': 49},
dataset_properties={
        'task': 1,
        'sparse': True,
        'multilabel': False,
        'multiclass': False,
        'target_type': 'classification',
        'signed': False})),
(0.020000,
SimpleClassificationPipeline({
        'balancing:strategy': 'none',
        'categorical_encoding:__choice__': 'no_encoding',
        'classifier:__choice__': 'random_forest',
        'imputation:strategy': 'mean',
        'preprocessor:__choice__': 'truncatedSVD',
        'rescaling:__choice__': 'standardize',
        'classifier:random_forest:bootstrap': 'False',
        'classifier:random_forest:criterion': 'gini',
        'classifier:random_forest:max_depth': 'None',
        'classifier:random_forest:max_features': 0.5625796853303024,
        'classifier:random_forest:max_leaf_nodes': 'None',
        'classifier:random_forest:min_impurity_decrease': 0.0,
        'classifier:random_forest:min_samples_leaf': 5,
        'classifier:random_forest:min_samples_split': 4,
        'classifier:random_forest:min_weight_fraction_leaf': 0.0,
```

```
        'classifier:random_forest:n_estimators': 100,
        'preprocessor:truncatedSVD:target_dim': 49},
dataset_properties={
        'task': 1,
        'sparse': True,
        'multilabel': False,
        'multiclass': False,
        'target_type': 'classification',
        'signed': False})),
(0.020000,
SimpleClassificationPipeline({
        'balancing:strategy': 'none',
        'categorical_encoding:__choice__': 'no_encoding',
        'classifier:__choice__': 'random_forest',
        'imputation:strategy': 'mean',
        'preprocessor:__choice__': 'truncatedSVD',
        'rescaling:__choice__': 'standardize',
        'classifier:random_forest:bootstrap': 'False',
        'classifier:random_forest:criterion': 'entropy',
        'classifier:random_forest:max_depth': 'None',
        'classifier:random_forest:max_features': 0.6000333850123087,
        'classifier:random_forest:max_leaf_nodes': 'None',
        'classifier:random_forest:min_impurity_decrease': 0.0,
        'classifier:random_forest:min_samples_leaf': 8,
        'classifier:random_forest:min_samples_split': 9,
        'classifier:random_forest:min_weight_fraction_leaf': 0.0,
        'classifier:random_forest:n_estimators': 100,
        'preprocessor:truncatedSVD:target_dim': 48},
dataset_properties={
        'task': 1,
        'sparse': True,
        'multilabel': False,
        'multiclass': False,
        'target_type': 'classification',
        'signed': False})),
(0.020000,
SimpleClassificationPipeline({
        'balancing:strategy': 'none',
        'categorical_encoding:__choice__': 'no_encoding',
        'classifier:__choice__': 'random_forest',
        'imputation:strategy': 'mean',
        'preprocessor:__choice__': 'truncatedSVD',
        'rescaling:__choice__': 'standardize',
        'classifier:random_forest:bootstrap': 'False',
        'classifier:random_forest:criterion': 'entropy',
        'classifier:random_forest:max_depth': 'None',
        'classifier:random_forest:max_features': 0.5625796853303024,
        'classifier:random_forest:max_leaf_nodes': 'None',
        'classifier:random_forest:min_impurity_decrease': 0.0,
        'classifier:random_forest:min_samples_leaf': 8,
        'classifier:random_forest:min_samples_split': 5,
        'classifier:random_forest:min_weight_fraction_leaf': 0.0,
        'classifier:random_forest:n_estimators': 100,
        'preprocessor:truncatedSVD:target_dim': 49},
```

```
dataset_properties={
        'task': 1,
        'sparse': True,
        'multilabel': False,
        'multiclass': False,
        'target_type': 'classification',
        'signed': False}))
```

3.2 Auto-sklearn

Listing 2. Pipeline resulting from the auto-sklearn run on non-encoded data

```
(0.160000,
SimpleClassificationPipeline({
        'balancing:strategy': 'none',
        'categorical_encoding:__choice__': 'no_encoding',
        'classifier:__choice__': 'random_forest',
        'imputation:strategy': 'median',
        'preprocessor:__choice__': 'no_preprocessing',
        'rescaling:__choice__': 'none',
        'classifier:random_forest:bootstrap': 'False',
        'classifier:random_forest:criterion': 'gini',
        'classifier:random_forest:max_depth': 'None',
        'classifier:random_forest:max_features': 0.7406733250617115,
        'classifier:random_forest:max_leaf_nodes': 'None',
        'classifier:random_forest:min_impurity_decrease': 0.0,
        'classifier:random_forest:min_samples_leaf': 1,
        'classifier:random_forest:min_samples_split': 2,
        'classifier:random_forest:min_weight_fraction_leaf': 0.0,
        'classifier:random_forest:n_estimators': 100},
dataset_properties={
        'task': 1,
        'sparse': False,
        'multilabel': False,
        'multiclass': False,
        'target_type': 'classification',
        'signed': False})),
(0.120000,
SimpleClassificationPipeline({
        'balancing:strategy': 'none',
        'categorical_encoding:__choice__': 'no_encoding',
        'classifier:__choice__': 'random_forest',
        'imputation:strategy': 'mean',
        'preprocessor:__choice__': 'feature_agglomeration',
        'rescaling:__choice__': 'none',
        'classifier:random_forest:bootstrap': 'False',
        'classifier:random_forest:criterion': 'entropy',
        'classifier:random_forest:max_depth': 'None',
        'classifier:random_forest:max_features': 0.8847685853813819,
        'classifier:random_forest:max_leaf_nodes': 'None',
        'classifier:random_forest:min_impurity_decrease': 0.0,
        'classifier:random_forest:min_samples_leaf': 2,
        'classifier:random_forest:min_samples_split': 3,
        'classifier:random_forest:min_weight_fraction_leaf': 0.0,
        'classifier:random_forest:n_estimators': 100,
```

```
        'preprocessor:feature_agglomeration:affinity': 'manhattan',
        'preprocessor:feature_agglomeration:linkage': 'complete',
        'preprocessor:feature_agglomeration:n_clusters': 378,
        'preprocessor:feature_agglomeration:pooling_func': 'max'},
dataset_properties={
        'task': 1,
        'sparse': False,
        'multilabel': False,
        'multiclass': False,
        'target_type': 'classification',
        'signed': False})),
(0.100000,
SimpleClassificationPipeline({
        'balancing:strategy': 'weighting',
        'categorical_encoding:__choice__': 'one_hot_encoding',
        'classifier:__choice__': 'random_forest',
        'imputation:strategy': 'most_frequent',
        'preprocessor:__choice__': 'feature_agglomeration',
        'rescaling:__choice__': 'quantile_transformer',
        'categorical_encoding:one_hot_encoding:use_minimum_fraction': 'True',
        'classifier:random_forest:bootstrap': 'False',
        'classifier:random_forest:criterion': 'entropy',
        'classifier:random_forest:max_depth': 'None',
        'classifier:random_forest:max_features': 0.9077581635305297,
        'classifier:random_forest:max_leaf_nodes': 'None',
        'classifier:random_forest:min_impurity_decrease': 0.0,
        'classifier:random_forest:min_samples_leaf': 1,
        'classifier:random_forest:min_samples_split': 3,
        'classifier:random_forest:min_weight_fraction_leaf': 0.0,
        'classifier:random_forest:n_estimators': 100,
        'preprocessor:feature_agglomeration:affinity': 'cosine',
        'preprocessor:feature_agglomeration:linkage': 'average',
        'preprocessor:feature_agglomeration:n_clusters': 332,
        'preprocessor:feature_agglomeration:pooling_func': 'median',
        'rescaling:quantile_transformer:n_quantiles': 1000,
        'rescaling:quantile_transformer:output_distribution': 'uniform',
        'categorical_encoding:one_hot_encoding:minimum_fraction': 0.03392625042313709},
dataset_properties={
        'task': 1,
        'sparse': False,
        'multilabel': False,
        'multiclass': False,
        'target_type': 'classification',
        'signed': False})),
(0.100000,
SimpleClassificationPipeline({
        'balancing:strategy': 'none',
        'categorical_encoding:__choice__': 'no_encoding',
        'classifier:__choice__': 'gaussian_nb',
        'imputation:strategy': 'median',
        'preprocessor:__choice__': 'no_preprocessing',
        'rescaling:__choice__': 'minmax'},
dataset_properties={
        'task': 1,
```

```
        'sparse': False,
        'multilabel': False,
        'multiclass': False,
        'target_type': 'classification',
        'signed': False})),
(0.060000,
SimpleClassificationPipeline({
        'balancing:strategy': 'none',
        'categorical_encoding:__choice__': 'one_hot_encoding',
        'classifier:__choice__': 'random_forest',
        'imputation:strategy': 'most_frequent',
        'preprocessor:__choice__': 'feature_agglomeration',
        'rescaling:__choice__': 'quantile_transformer',
        'categorical_encoding:one_hot_encoding:use_minimum_fraction': 'False',
        'classifier:random_forest:bootstrap': 'False',
        'classifier:random_forest:criterion': 'gini',
        'classifier:random_forest:max_depth': 'None',
        'classifier:random_forest:max_features': 0.8982794213962207,
        'classifier:random_forest:max_leaf_nodes': 'None',
        'classifier:random_forest:min_impurity_decrease': 0.0,
        'classifier:random_forest:min_samples_leaf': 1,
        'classifier:random_forest:min_samples_split': 6,
        'classifier:random_forest:min_weight_fraction_leaf': 0.0,
        'classifier:random_forest:n_estimators': 100,
        'preprocessor:feature_agglomeration:affinity': 'euclidean',
        'preprocessor:feature_agglomeration:linkage': 'ward',
        'preprocessor:feature_agglomeration:n_clusters': 25,
        'preprocessor:feature_agglomeration:pooling_func': 'mean',
        'rescaling:quantile_transformer:n_quantiles': 83,
        'rescaling:quantile_transformer:output_distribution': 'uniform'},
dataset_properties={
        'task': 1,
        'sparse': False,
        'multilabel': False,
        'multiclass': False,
        'target_type': 'classification',
        'signed': False})),
(0.060000,
SimpleClassificationPipeline({
        'balancing:strategy': 'weighting',
        'categorical_encoding:__choice__': 'one_hot_encoding',
        'classifier:__choice__': 'random_forest',
        'imputation:strategy': 'most_frequent',
        'preprocessor:__choice__': 'feature_agglomeration',
        'rescaling:__choice__': 'quantile_transformer',
        'categorical_encoding:one_hot_encoding:use_minimum_fraction': 'False',
        'classifier:random_forest:bootstrap': 'True',
        'classifier:random_forest:criterion': 'gini',
        'classifier:random_forest:max_depth': 'None',
        'classifier:random_forest:max_features': 0.9834768425234652,
        'classifier:random_forest:max_leaf_nodes': 'None',
        'classifier:random_forest:min_impurity_decrease': 0.0,
        'classifier:random_forest:min_samples_leaf': 1,
        'classifier:random_forest:min_samples_split': 5,
```

```
        'classifier:random_forest:min_weight_fraction_leaf': 0.0,
        'classifier:random_forest:n_estimators': 100,
        'preprocessor:feature_agglomeration:affinity': 'manhattan',
        'preprocessor:feature_agglomeration:linkage': 'average',
        'preprocessor:feature_agglomeration:n_clusters': 301,
        'preprocessor:feature_agglomeration:pooling_func': 'max',
        'rescaling:quantile_transformer:n_quantiles': 554,
        'rescaling:quantile_transformer:output_distribution': 'uniform'},
dataset_properties={
        'task': 1,
        'sparse': False,
        'multilabel': False,
        'multiclass': False,
        'target_type': 'classification',
        'signed': False})),
(0.060000,
SimpleClassificationPipeline({
        'balancing:strategy': 'weighting',
        'categorical_encoding:__choice__': 'no_encoding',
        'classifier:__choice__': 'random_forest',
        'imputation:strategy': 'most_frequent',
        'preprocessor:__choice__': 'select_rates',
        'rescaling:__choice__': 'minmax',
        'classifier:random_forest:bootstrap': 'True',
        'classifier:random_forest:criterion': 'gini',
        'classifier:random_forest:max_depth': 'None',
        'classifier:random_forest:max_features': 0.8722325512075406,
        'classifier:random_forest:max_leaf_nodes': 'None',
        'classifier:random_forest:min_impurity_decrease': 0.0,
        'classifier:random_forest:min_samples_leaf': 2,
        'classifier:random_forest:min_samples_split': 4,
        'classifier:random_forest:min_weight_fraction_leaf': 0.0,
        'classifier:random_forest:n_estimators': 100,
        'preprocessor:select_rates:alpha': 0.1,
        'preprocessor:select_rates:mode': 'fpr',
        'preprocessor:select_rates:score_func': 'chi2'},
dataset_properties={
        'task': 1,
        'sparse': False,
        'multilabel': False,
        'multiclass': False,
        'target_type': 'classification',
        'signed': False})),
(0.060000,
SimpleClassificationPipeline({
        'balancing:strategy': 'none',
        'categorical_encoding:__choice__': 'one_hot_encoding',
        'classifier:__choice__': 'random_forest',
        'imputation:strategy': 'mean',
        'preprocessor:__choice__': 'extra_trees_preproc_for_classification',
        'rescaling:__choice__': 'quantile_transformer',
        'categorical_encoding:one_hot_encoding:use_minimum_fraction': 'False',
        'classifier:random_forest:bootstrap': 'True',
        'classifier:random_forest:criterion': 'gini',
```

```
        'classifier:random_forest:max_depth': 'None',
        'classifier:random_forest:max_features': 0.9106973701532353,
        'classifier:random_forest:max_leaf_nodes': 'None',
        'classifier:random_forest:min_impurity_decrease': 0.0,
        'classifier:random_forest:min_samples_leaf': 2,
        'classifier:random_forest:min_samples_split': 6,
        'classifier:random_forest:min_weight_fraction_leaf': 0.0,
        'classifier:random_forest:n_estimators': 100,
        'preprocessor:extra_trees_preproc_for_classification:bootstrap': 'True',
        'preprocessor:extra_trees_preproc_for_classification:criterion': 'gini',
        'preprocessor:extra_trees_preproc_for_classification:max_depth': 'None',
        'preprocessor:extra_trees_preproc_for_classification:max_features': 0.8621352709576258,
        'preprocessor:extra_trees_preproc_for_classification:max_leaf_nodes': 'None',
        'preprocessor:extra_trees_preproc_for_classification:min_impurity_decrease': 0.0,
        'preprocessor:extra_trees_preproc_for_classification:min_samples_leaf': 8,
        'preprocessor:extra_trees_preproc_for_classification:min_samples_split': 13,
        'preprocessor:extra_trees_preproc_for_classification:min_weight_fraction_leaf': 0.0,
        'preprocessor:extra_trees_preproc_for_classification:n_estimators': 100,
        'rescaling:quantile_transformer:n_quantiles': 654,
        'rescaling:quantile_transformer:output_distribution': 'uniform'},
dataset_properties={
        'task': 1,
        'sparse': False,
        'multilabel': False,
        'multiclass': False,
        'target_type': 'classification',
        'signed': False})),
(0.040000,
SimpleClassificationPipeline({
        'balancing:strategy': 'none',
        'categorical_encoding:__choice__': 'no_encoding',
        'classifier:__choice__': 'random_forest',
        'imputation:strategy': 'most_frequent',
        'preprocessor:__choice__': 'select_rates',
        'rescaling:__choice__': 'quantile_transformer',
        'classifier:random_forest:bootstrap': 'False',
        'classifier:random_forest:criterion': 'gini',
        'classifier:random_forest:max_depth': 'None',
        'classifier:random_forest:max_features': 0.8982794213962207,
        'classifier:random_forest:max_leaf_nodes': 'None',
        'classifier:random_forest:min_impurity_decrease': 0.0,
        'classifier:random_forest:min_samples_leaf': 1,
        'classifier:random_forest:min_samples_split': 6,
        'classifier:random_forest:min_weight_fraction_leaf': 0.0,
        'classifier:random_forest:n_estimators': 100,
        'preprocessor:select_rates:alpha': 0.1,
        'preprocessor:select_rates:mode': 'fpr',
        'preprocessor:select_rates:score_func': 'chi2',
        'rescaling:quantile_transformer:n_quantiles': 34,
        'rescaling:quantile_transformer:output_distribution': 'uniform'},
dataset_properties={
        'task': 1,
        'sparse': False,
        'multilabel': False,
```

```
        'multiclass': False,
        'target_type': 'classification',
        'signed': False})),
(0.040000,
SimpleClassificationPipeline({
        'balancing:strategy': 'none',
        'categorical_encoding:__choice__': 'one_hot_encoding',
        'classifier:__choice__': 'random_forest',
        'imputation:strategy': 'median',
        'preprocessor:__choice__': 'polynomial',
        'rescaling:__choice__': 'quantile_transformer',
        'categorical_encoding:one_hot_encoding:use_minimum_fraction': 'True',
        'classifier:random_forest:bootstrap': 'False',
        'classifier:random_forest:criterion': 'entropy',
        'classifier:random_forest:max_depth': 'None',
        'classifier:random_forest:max_features': 0.49684980706382087,
        'classifier:random_forest:max_leaf_nodes': 'None',
        'classifier:random_forest:min_impurity_decrease': 0.0,
        'classifier:random_forest:min_samples_leaf': 2,
        'classifier:random_forest:min_samples_split': 5,
        'classifier:random_forest:min_weight_fraction_leaf': 0.0,
        'classifier:random_forest:n_estimators': 100,
        'preprocessor:polynomial:degree': 2,
        'preprocessor:polynomial:include_bias': 'False',
        'preprocessor:polynomial:interaction_only': 'True',
        'rescaling:quantile_transformer:n_quantiles': 1365,
        'rescaling:quantile_transformer:output_distribution': 'uniform',
        'categorical_encoding:one_hot_encoding:minimum_fraction': 0.005204700804987528},
dataset_properties={
        'task': 1,
        'sparse': False,
        'multilabel': False,
        'multiclass': False,
        'target_type': 'classification',
        'signed': False})),
(0.040000,
SimpleClassificationPipeline({
        'balancing:strategy': 'none',
        'categorical_encoding:__choice__': 'no_encoding',
        'classifier:__choice__': 'random_forest',
        'imputation:strategy': 'most_frequent',
        'preprocessor:__choice__': 'feature_agglomeration',
        'rescaling:__choice__': 'quantile_transformer',
        'classifier:random_forest:bootstrap': 'False',
        'classifier:random_forest:criterion': 'gini',
        'classifier:random_forest:max_depth': 'None',
        'classifier:random_forest:max_features': 0.8982794213962207,
        'classifier:random_forest:max_leaf_nodes': 'None',
        'classifier:random_forest:min_impurity_decrease': 0.0,
        'classifier:random_forest:min_samples_leaf': 1,
        'classifier:random_forest:min_samples_split': 5,
        'classifier:random_forest:min_weight_fraction_leaf': 0.0,
        'classifier:random_forest:n_estimators': 100,
        'preprocessor:feature_agglomeration:affinity': 'euclidean',
```

```
        'preprocessor:feature_agglomeration:linkage': 'ward',
        'preprocessor:feature_agglomeration:n_clusters': 25,
        'preprocessor:feature_agglomeration:pooling_func': 'median',
        'rescaling:quantile_transformer:n_quantiles': 55,
        'rescaling:quantile_transformer:output_distribution': 'uniform'},
dataset_properties={
        'task': 1,
        'sparse': False,
        'multilabel': False,
        'multiclass': False,
        'target_type': 'classification',
        'signed': False})),
(0.040000,
SimpleClassificationPipeline({
        'balancing:strategy': 'weighting',
        'categorical_encoding:__choice__': 'one_hot_encoding',
        'classifier:__choice__': 'lda',
        'imputation:strategy': 'mean',
        'preprocessor:__choice__': 'liblinear_svc_preprocessor',
        'rescaling:__choice__': 'quantile_transformer',
        'categorical_encoding:one_hot_encoding:use_minimum_fraction': 'True',
        'classifier:lda:n_components': 113,
        'classifier:lda:shrinkage': 'auto',
        'classifier:lda:tol': 0.007910428523698795,
        'preprocessor:liblinear_svc_preprocessor:C': 173.0722191525089,
        'preprocessor:liblinear_svc_preprocessor:dual': 'False',
        'preprocessor:liblinear_svc_preprocessor:fit_intercept': 'True',
        'preprocessor:liblinear_svc_preprocessor:intercept_scaling': 1,
        'preprocessor:liblinear_svc_preprocessor:loss': 'squared_hinge',
        'preprocessor:liblinear_svc_preprocessor:multi_class': 'ovr',
        'preprocessor:liblinear_svc_preprocessor:penalty': 'l1',
        'preprocessor:liblinear_svc_preprocessor:tol': 0.042073318306322716,
        'rescaling:quantile_transformer:n_quantiles': 1445,
        'rescaling:quantile_transformer:output_distribution': 'uniform',
        'categorical_encoding:one_hot_encoding:minimum_fraction': 0.0005427958220594335},
dataset_properties={
        'task': 1,
        'sparse': False,
        'multilabel': False,
        'multiclass': False,
        'target_type': 'classification',
        'signed': False})),
(0.040000,
SimpleClassificationPipeline({
        'balancing:strategy': 'none',
        'categorical_encoding:__choice__': 'one_hot_encoding',
        'classifier:__choice__': 'adaboost',
        'imputation:strategy': 'mean',
        'preprocessor:__choice__': 'select_rates',
        'rescaling:__choice__': 'none',
        'categorical_encoding:one_hot_encoding:use_minimum_fraction': 'False',
        'classifier:adaboost:algorithm': 'SAMME.R',
        'classifier:adaboost:learning_rate': 0.7539256137374791,
        'classifier:adaboost:max_depth': 6,
```

```
        'classifier:adaboost:n_estimators': 414,
        'preprocessor:select_rates:alpha': 0.3528046794905546,
        'preprocessor:select_rates:mode': 'fdr',
        'preprocessor:select_rates:score_func': 'f_classif'},
dataset_properties={
        'task': 1,
        'sparse': False,
        'multilabel': False,
        'multiclass': False,
        'target_type': 'classification',
        'signed': False})),
(0.020000,
SimpleClassificationPipeline({
        'balancing:strategy': 'none',
        'categorical_encoding:__choice__': 'one_hot_encoding',
        'classifier:__choice__': 'random_forest',
        'imputation:strategy': 'median',
        'preprocessor:__choice__': 'feature_agglomeration',
        'rescaling:__choice__': 'none',
        'categorical_encoding:one_hot_encoding:use_minimum_fraction': 'True',
        'classifier:random_forest:bootstrap': 'False',
        'classifier:random_forest:criterion': 'gini',
        'classifier:random_forest:max_depth': 'None',
        'classifier:random_forest:max_features': 0.8924418030653755,
        'classifier:random_forest:max_leaf_nodes': 'None',
        'classifier:random_forest:min_impurity_decrease': 0.0,
        'classifier:random_forest:min_samples_leaf': 1,
        'classifier:random_forest:min_samples_split': 3,
        'classifier:random_forest:min_weight_fraction_leaf': 0.0,
        'classifier:random_forest:n_estimators': 100,
        'preprocessor:feature_agglomeration:affinity': 'euclidean',
        'preprocessor:feature_agglomeration:linkage': 'average',
        'preprocessor:feature_agglomeration:n_clusters': 301,
        'preprocessor:feature_agglomeration:pooling_func': 'max',
        'categorical_encoding:one_hot_encoding:minimum_fraction': 0.010000000000000004},
dataset_properties={
        'task': 1,
        'sparse': False,
        'multilabel': False,
        'multiclass': False,
        'target_type': 'classification',
        'signed': False})),
(0.020000,
SimpleClassificationPipeline({
        'balancing:strategy': 'none',
        'categorical_encoding:__choice__': 'one_hot_encoding',
        'classifier:__choice__': 'lda',
        'imputation:strategy': 'median',
        'preprocessor:__choice__': 'feature_agglomeration',
        'rescaling:__choice__': 'quantile_transformer',
        'categorical_encoding:one_hot_encoding:use_minimum_fraction': 'False',
        'classifier:lda:n_components': 116,
        'classifier:lda:shrinkage': 'manual',
        'classifier:lda:tol': 0.017921201802049504,
```

```
        'preprocessor:feature_agglomeration:affinity': 'euclidean',
        'preprocessor:feature_agglomeration:linkage': 'ward',
        'preprocessor:feature_agglomeration:n_clusters': 373,
        'preprocessor:feature_agglomeration:pooling_func': 'mean',
        'rescaling:quantile_transformer:n_quantiles': 975,
        'rescaling:quantile_transformer:output_distribution': 'uniform',
        'classifier:lda:shrinkage_factor': 0.6614438704775101},
dataset_properties={
        'task': 1,
        'sparse': False,
        'multilabel': False,
        'multiclass': False,
        'target_type': 'classification',
        'signed': False})),
(0.020000,
SimpleClassificationPipeline({
        'balancing:strategy': 'none',
        'categorical_encoding:__choice__': 'no_encoding',
        'classifier:__choice__': 'random_forest',
        'imputation:strategy': 'most_frequent',
        'preprocessor:__choice__': 'no_preprocessing',
        'rescaling:__choice__': 'quantile_transformer',
        'classifier:random_forest:bootstrap': 'False',
        'classifier:random_forest:criterion': 'gini',
        'classifier:random_forest:max_depth': 'None',
        'classifier:random_forest:max_features': 0.8982794213962207,
        'classifier:random_forest:max_leaf_nodes': 'None',
        'classifier:random_forest:min_impurity_decrease': 0.0,
        'classifier:random_forest:min_samples_leaf': 1,
        'classifier:random_forest:min_samples_split': 9,
        'classifier:random_forest:min_weight_fraction_leaf': 0.0,
        'classifier:random_forest:n_estimators': 100,
        'rescaling:quantile_transformer:n_quantiles': 55,
        'rescaling:quantile_transformer:output_distribution': 'uniform'},
dataset_properties={
        'task': 1,
        'sparse': False,
        'multilabel': False,
        'multiclass': False,
        'target_type': 'classification',
        'signed': False})),
(0.020000,
SimpleClassificationPipeline({
        'balancing:strategy': 'weighting',
        'categorical_encoding:__choice__': 'one_hot_encoding',
        'classifier:__choice__': 'random_forest',
        'imputation:strategy': 'most_frequent',
        'preprocessor:__choice__': 'select_rates',
        'rescaling:__choice__': 'quantile_transformer',
        'categorical_encoding:one_hot_encoding:use_minimum_fraction': 'True',
        'classifier:random_forest:bootstrap': 'False',
        'classifier:random_forest:criterion': 'entropy',
        'classifier:random_forest:max_depth': 'None',
        'classifier:random_forest:max_features': 0.8868131108098152,
```

```
        'classifier:random_forest:max_leaf_nodes': 'None',
        'classifier:random_forest:min_impurity_decrease': 0.0,
        'classifier:random_forest:min_samples_leaf': 3,
        'classifier:random_forest:min_samples_split': 6,
        'classifier:random_forest:min_weight_fraction_leaf': 0.0,
        'classifier:random_forest:n_estimators': 100,
        'preprocessor:select_rates:alpha': 0.1,
        'preprocessor:select_rates:mode': 'fpr',
        'preprocessor:select_rates:score_func': 'chi2',
        'rescaling:quantile_transformer:n_quantiles': 1746,
        'rescaling:quantile_transformer:output_distribution': 'uniform',
        'categorical_encoding:one_hot_encoding:minimum_fraction': 0.3802535173087277},
dataset_properties={
        'task': 1,
        'sparse': False,
        'multilabel': False,
        'multiclass': False,
        'target_type': 'classification',
        'signed': False}))
```

3.3 TPOT One-hot-encoded

Listing 3. Pipeline resulting from the TPOT run on encoded data

```
('selectpercentile',
SelectPercentile(
        percentile=34,
        score_func=<function f_classif at 0x7effdbb17e18>)),
('xgbclassifier',
XGBClassifier(
        base_score=0.5,
        booster=None,
        colsample_bylevel=1,
        colsample_bynode=1,
        colsample_bytree=1,
        gamma=0,
        gpu_id=-1,
        importance_type='gain',
        interaction_constraints=None,
        learning_rate=1.0,
        max_delta_step=0,
        max_depth=9,
        min_child_weight=2,
        missing=nan,
        monotone_constraints=None,
        n_estimators=100,
        n_jobs=1,
        nthread=1,
        num_parallel_tree=1,
        objective='binary:logistic',
        random_state=0,
        reg_alpha=0,
        reg_lambda=1,
        scale_pos_weight=1,
        subsample=0.8500000000000001,
```

```
        tree_method=None,
        validate_parameters=False,
        verbosity=None))
```

3.4 TPOT

Listing 4. Pipeline resulting from the TPOT run on non-encoded data

```
('randomforestclassifier',
RandomForestClassifier(
        bootstrap=True,
        ccp_alpha=0.0,
        class_weight=None,
        criterion='entropy',
        max_depth=None,
        max_features=0.4,
        max_leaf_nodes=None,
        max_samples=None,
        min_impurity_decrease=0.0,
        min_impurity_split=None,
        min_samples_leaf=1,
        min_samples_split=5,
        min_weight_fraction_leaf=0.0,
        n_estimators=100,
        n_jobs=None,
        oob_score=False,
        random_state=None,
        verbose=0,
        warm_start=False))
```

REFERENCES

Ali, T., Nauman, M., and Jan, S. (2017). Trust in IoT: Dynamic Remote Attestation through Efficient Behavior Capture. *Cluster Computing*

Arora, D., Ravi, S., Raghunathan, A., and Jha, N. K. (2005). Secure embedded processing through hardware-assisted run-time monitoring. In *Design, Automation and Test in Europe*. 178–183 Vol. 1. doi:10.1109/DATE.2005.266

Arshad, S., Shah, M. A., Wahid, A., Mehmood, A., Song, H., and Yu, H. (2018). SAMADroid: A Novel 3-Level Hybrid Malware Detection Model for Android Operating System. *IEEE Access* 6

Arthur, D. and Vassilvitskii, S. (2007). K-Means++: the Advantages of Careful Seeding. In *Proceedings of the 18. Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)* (ACM)

Azmi, R. and Pishgoo, B. (2013). SHADuDT: Secure Hypervisor-Based Anomaly Detection Using Danger Theory. *Computers & Security* 39

Baum, L. E., Petrie, T., Soules, G., and Weiss, N. (1970). A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *The Annals of Mathematical Statistics* 41

Bidoki, S. M., Jalili, S., and Tajoddin, A. (2017). PbMMD: A Novel Policy Based Multi-Process Malware Detection. *Engineering Applications of Artificial Intelligence* 60

Eikerling, H. (2018). *Prototyping Infrastructure for Automotive Application Intrusion Detection*. Master's thesis, Paderborn University

Elsabagh, M., Barbará, D., Fleck, D., and Stavrou, A. (2018). On Early Detection of Application-Level Resource Exhaustion and Starvation. *The Journal of Systems and Software* 9404, 430–447

Esfahani, N., Yuan, E., Canavera, K. R., and Malek, S. (2016). Inferring Software Component Interaction Dependencies for Adaptation Support. *ACM Transactions on Autonomous and Adaptive Systems* 10

Forrest, S., Hofmeyr, S., and Somayaji, A. (2008). The Evolution of System-Call Monitoring. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC)* (IEEE)

Ganesh, V. and Dill, D. L. (2007). A decision procedure for bit-vectors and arrays. In *Proceedings of the International Conference on Computer Aided Verification* (Springer Berlin Heidelberg)

Gu, Z., Pei, K., Wang, Q., Si, L., Zhang, X., and Xu, D. (2015). Leaps: Detecting camouflaged attacks with statistical learning guided by program analysis. In *Proceedings of the 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks* (IEEE)

Han, Y., Etigowni, S., Liu, H., Zonouz, S., and Petropulu, A. (2017). Watch me, but don't touch me! contactless control flow monitoring via electromagnetic emanations. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (ACM), CCS '17

Hofmeyr, S. A., Forrest, S., and Somayaji, A. (1998). Intrusion Detection Using Sequences of System Calls. *Journal of Computer Security* 6

Hu, H., Shinde, S., Adrian, S., Chua, Z. L., Saxena, P., and Liang, Z. (2016). Data-oriented programming: On the expressiveness of non-control data attacks. In *Proceedings of the Symposium on Security and Privacy (SP)* (IEEE), 969–986

Islam, S., Khreich, W., and Hamou-Lhadj, A. (2018). Anomaly Detection Techniques Based on Kappa-Pruned Ensembles. *IEEE Transactions on Reliability* 67

Kephart, J. O. and Chess, D. M. (2003). The vision of autonomic computing. *Computer* 36

Khreich, W., Khosravifar, B., Hamou-Lhadj, A., and Talhi, C. (2017). An Anomaly Detection System Based on Variable N-Gram Features and One-class SVM. *Information and Software Technology* 91

Khreich, W., Murtaza, S. S., Hamou-Lhadj, A., and Talhi, C. (2018). Combining Heterogeneous Anomaly Detectors for Improved Software Security. *Journal of Systems and Software* 137

Lu, S. and Lysecky, R. (2019). Data-driven anomaly detection with timing features. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 24

Mansour, C. and Chasaki, D. (2019). Adaptive security monitoring for next-generation routers. *EURASIP Journal on Embedded Systems*

Masri, W., Abou Assi, R., and El-Ghali, M. (2014). Generating Profile-Based Signatures for Online Intrusion and Failure Detection. *Information and Software Technology* 56

Masri, W. and Halabi, H. (2011). An algorithm for capturing variables dependences in test suites. *Journal of Systems and Software* 84

Masri, W., Nahas, N., and Podgurski, A. (2006). Memoized forward computation of dynamic slices. In *Proceedings of the International Symposium on Software Reliability Engineering (ISSRE)* (IEEE), 23–32

Masri, W. and Podgurski, A. (2009). Algorithms and tool support for dynamic information flow analysis. *Information and Software Technology* 51

Masri, W., Podgurski, A., and Leon, D. (2007). An empirical study of test case filtering techniques based on exercising information flows. *IEEE Transactions on Software Engineering* 33

Muniswamy-Reddy, K.-K., Holland, D. A., Braun, U., and Seltzer, M. (2006). Provenance-aware storage systems. In *Proceedings of the USENIX Annual Technical Conference (ATEC)* (USENIX)

Ragel, R. G. and Parameswaran, S. (2006). Impres: integrated monitoring for processor reliability and security. In *2006 43rd ACM/IEEE Design Automation Conference*. 502–505. doi:10.1145/1146909.1147041

Reeves, J., Ramaswamy, A., Locasto, M., Bratus, S., and Smith, S. (2012). Intrusion Detection for Resource-Constrained Embedded Control Systems in the Power Grid. *International Journal of Critical Infrastructure Protection* 5

Shu, X., Yao, D. D., Ramakrishnan, N., and Jaeger, T. (2017). Long-Span Program Behavior Modeling and Attack Detection. *ACM Transactions on Privacy and Security (TOPS)* 20

Tajoddin, A. and Abadi, M. (2019). RAMD: registry-based anomaly malware detection using one-class ensemble classifiers. *Applied Intelligence*

Xie, Y., Feng, D., Tan, Z., and Zhou, J. (2016). Unifying Intrusion Detection and Forensic Analysis via Provenance Awareness. *Future Generation Computer Systems* 61, 26–36

Yoon, M., Mohan, S., Choi, J., Kim, J., and Sha, L. (2013). Securecore: A multicore-based intrusion detection architecture for real-time embedded systems. In *2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 21–32. doi:10.1109/RTAS.2013.6531076

Zegzhda, P. D., Kort, S. S., and Suprun, A. F. (2015). Detection of Anomalies in Behavior of the Software with Usage of Markov Chains. *Automatic Control and Computer Sciences* 49

Zhang, M., Raghunathan, A., and Jha, N. K. (2014). A Defense Framework Against Malware and Vulnerability Exploits. *International Journal of Information Security* 13

Zonouz, S., Houmansadr, A., Berthier, R., Borisov, N., and Sanders, W. (2013). Secloud: A Cloud-Based Comprehensive and Lightweight Security Solution for Smartphones. *Computers & Security* 37