AIDIS: Detecting and Classifying Anomalous Behavior in Ubiquitous Kernel Processes

Robert Luh, Helge Janicke, Sebastian Schrittwieser

PII: S0167-4048(18)31445-7
DOI: https://doi.org/10.1016/j.cose.2019.03.015
Reference: COSE 1494

To appear in: *Computers & Security*

Received date: 10 December 2018
Revised date: 15 March 2019
Accepted date: 17 March 2019

Please cite this article as: Robert Luh, Helge Janicke, Sebastian Schrittwieser, AIDIS: Detecting and Classifying Anomalous Behavior in Ubiquitous Kernel Processes, *Computers & Security* (2019), doi: https://doi.org/10.1016/j.cose.2019.03.015

AIDIS: Detecting and Classifying Anomalous Behavior in Ubiquitous Kernel Processes

Robert Luh[a,b,*], Helge Janicke[b], Sebastian Schrittwieser[a,b]

[a] *St. Pölten University of Applied Sciences, Josef Ressel Center TARGET*
Matthias Corvinus St. 15, St. Pölten, Austria
[b] *De Montfort University (DMU), Leicester*
The Gateway, Leicester, United Kingdom

Abstract

Targeted attacks on IT systems are a rising threat against the confidentiality, integrity, and availability of critical information and infrastructures. With the rising prominence of advanced persistent threats (APTs), identifying and understanding such attacks has become increasingly important. Current signature-based systems are heavily reliant on fixed patterns that struggle with unknown or evasive applications, while behavior-based solutions usually leave most of the interpretative work to a human analyst.
In this article we propose AIDIS, an Advanced Intrusion Detection and Interpretation System capable to explain anomalous behavior within a network-enabled user session by considering kernel event anomalies identified through their deviation from a set of baseline process graphs. For this purpose we adapt star structures, a bipartite representation used to approximate the edit distance between two graphs. Baseline templates are generated automatically and adapt to the nature of the respective operating system process.
We prototypically implemented smart anomaly classification through a set of competency questions applied to graph template deviations and evaluated the approach using both Random Forest and linear kernel support vector machines. The determined attack classes are ultimately mapped to a dedicated APT attacker/defender meta model that considers actions, actors, as well as assets and mitigating controls, thereby enabling decision support and contextual interpretation of ongoing attacks.

Keywords: intrusion detection, malware, anomaly detection, graph matching, star structure, security model, semantic gap, machine learning, classification, svm

*Corresponding author
Email address: robert.luh@fhstp.ac.at (Robert Luh)

1. Introduction

IT systems are threatened by a growing number of cyber-attacks. With the emergence of Advanced Persistent Threats (APTs), the focus shifted from off-the-shelf malware to multipartite attacks that are tailored to specific organizations or systems. These targeted threats are driven by varying motivations, such as espionage or sabotage, and often cause significantly more damage.

Several cases in recent history have shown that targeted attacks sometimes remain undiscovered by their victims for many months or even years [10, 21, 22, 32, 69]. The prime example, Stuxnet, which targeted programmable logic controllers (PLCs) of sensitive industrial systems, was active for at least 3 years until discovery [65]. According to a Symantec study [15], Stuxnet infected close to 100,000 systems across 115 countries. Its quasi successor, Duqu, also targeted industrial control systems (ICS), gathering sensitive information in at least eight countries [7, 29]. On the espionage side, the Regin Trojan is believed to have been used for global, systematic campaigns since at least 2008 [68]. Other examples include Flame [30], Mahdi [63], and Gauss [31]. These strains are currently used for cyber-espionage in Middle Eastern countries and, depending on the variant, are capable of stealing passwords and cookies, recording network traffic, keystrokes, microphone audio, and even entire Sykpe conversations [46].

APTs are increasingly affecting less prominent targets as well. In 2013 alone, "economic espionage and theft of trade secrets cost the American economy more than $19 billion" [51]. In the 2017 Official Annual Cybercrime Report by Cybersecurity Ventures [50], analysts speak of an estimated $6 trillion of annual damage that will be caused by cyber attacks by 2021.

While APTs use malware like most conventional attacks, the level of complexity and sophistication is usually much higher. This is problematic since defensive measures offered by security vendors often utilize the same detection approaches that have been used for years – with mixed results. The major drawback of these primarily signature-based systems is that the binary patterns required for detection are unlikely to exist at the time of attack, as most APTs are tailored to one specific entity. In addition, meta- and polymorphic techniques are employed to throw off signature-based systems while the multi-stage nature of APTs makes it generally difficult to interpret findings out of context [38]. This increased complexity makes it necessary to explore novel techniques for threat intelligence and malicious activity detection on multiple layers.

Behavior-based approaches are a promising means to identify illegal actions. No matter the stealth techniques employed, the attacker will sooner or later execute his or her action on target – be it data theft, hijacking or sabotage. Anomalies signifying a deviation from a known behavioral baseline can then be used to detect the threat. However, most existing systems do not disseminate the offending behavioral data and contribute little to its interpretation. We argue that closing the resulting semantic gap is a vital next step in holistic IT system threat mitigation.

To achieve this goal, we introduce AIDIS, an "Advanced Intrusion Detection and Interpretation System" capable of dealing with a variety of targeted

attack scenarios. Our approach, which is based on a bare system design published in 2017 [41], seeks to contribute by presenting a semantics-aware, fully transparent graph-based anomaly detection system coupled with the automated interpretation of abstracted kernel events collected from Windows workstation computers. With AIDIS, we widen the focus of analysis from suspicious binaries to ubiquitous kernel processes that might be affected by adversarial action, thereby enabling uniform anomaly detection for known portions of the operating system. Our approach reduces the computational requirements attached to analyzing each and every application launched on a system and presents anomalies in a human-readable way. Identified outliers are ultimately mapped to our gamified APT meta model, which encompasses widely used threat intelligence languages, attack patterns, as well as mitigating controls. The model can be queried for information and possible responses to past and ongoing attacks while always maintaining the link to the data layer beneath.

In its entirety, AIDIS enables detecting attacks and sharing interpreted threat intelligence for whole OS ecosystems. The accessible combination of attack modeling, white box anomaly detection, and real-world system event data provides a novel approach to combating high-impact threats to IT infrastructures.

Specifically, we contribute by:

- Presenting a holistic approach to collecting and analyzing host and network events able to describe and assess all victim-side APT attack stages;
- Introducing a transparent anomaly detection system based on star graph structures;
- Providing optional processing components incorporating features such as event sequence compression through grammar inference and sentiment analysis for identifying expressive operating system processes;
- Classifying anomalies into semantic threat categories based on the CAPEC dictionary [48] using both a Random Forest (RF) and support vector machine (SVM) approach;
- Enabling the interpretation of classified data through a comprehensive targeted attack meta model encompassing actors, assets, as well as hostile and mitigating actions.

In the following, we discuss related approaches and how our system can contribute to current attack detection efforts. Section 2 presents related work in the areas of attack modeling and anomaly detection/classification. In Section 3, the design considerations of our interpretation system are discussed in detail. AIDIS' technical implementation and all its components are elucidated in Section 4, while a full evaluation can be found in Section 5. In conclusion, we discuss the current prototype's properties and drawbacks and outline future work.

2. Related Work

2.1. Attack Modeling

Threat formalisms such as attack/defense models are commonly used by security analysts to share insight, enhance scenario coverage, and help planning and prioritizing threat mitigation by quantifying related system vulnerabilities. Both static and dynamic (behavioral) models are used [56]. AIDIS proposes the use of 'PenQuest' [43, 44], an advanced dynamic approach realized as a full-fledged strategy game. In the following, we discuss existing work similar to our modeling approach.

AIDIS uses an extended variant of the APT kill chain by Hutchins et al. [24], which represents a simple yet useful solution for modeling targeted attacks by depicting attacker activity as stages in the manner of a tiered military campaign. Several such models have been developed in the past – the decision of which variant to use largely depends on personal preference and data exchange requirements: While the cyber kill chain model [24] considers command and control activity and weaponization as separate stages, the model by Giura and Wang [20] is more detailed when it comes to the collection of data. Reconnaissance, exploitation, operation, and exfiltration stages are mostly identical, albeit named differently at times. Both models can be used in conjunction with MITRE's APT-enabled Structured Threat Information eXpression (STIX) data exchange format [49], which was developed to represent threat information in a comprehensive manner. In our work, we built upon the general kill chain approach and added sub-stages as well as interdependencies for a more finer-grained view on targeted attacks. See Section 3.2 for more information.

Similar to our model in much of its terminology, the Diamond Model of Intrusion Analysis [6] establishes the basic elements of generic intrusion activity, called an *event*, which is composed of four core features: adversary, infrastructure, capability, and victim. It extends events with a confidence score that can be used to track the reliability of the data source or a specific event. While some of its premises are comparable to our own work, the Diamond Model does not consider technical or organizational tools and controls. Mechanisms for determining specific actions conducted on the attacker's or defender's side are not offered. While the Diamond Model is a powerful template in its own regard, our approach aims to provide these mechanisms – and more: PenQuest/AIDIS combine threat modeling with a ready-to-use framework for simulation and automated knowledge discovery. In summary, the Diamond Model and our gamified approach share commonalities and could potentially benefit from each other in terms of feature modeling and terminology.

In the work by Syed et al. [67], the authors present a unified cyber security ontology (UCO) extending the Intrusion Detection System ontology by Undercoffer et al. [71]. UCO is a semantic version of STIX [4] with a link to security standards similar to the ones that are used in our work. Real-world knowledge is appended using featured Google searches (Google Knowledge Graph) and various knowledge bases. Syed et al. provide little information about data retrieval mechanisms and general automation. The main use cases emphasized

are the identification of similar software and the association of vulnerabilities with certain (classes of) products. Unlike our research, UCO does not consider temporal information or measurements of uncertainty.

Following Schneier's introduction of attack trees [61], the idea of a tree-like representation of an attack scenario, where the root of an attack tree corresponds to an attacker's goal, has been picked up by several other research groups. For instance, Kordy et al. [33] developed a formalism called attack-defense trees (ADTrees), which are node-labeled rooted trees describing not only the measures an attacker might take to attack a system but also the defenses that a victim can employ to protect it. The authors provide semantics and axiomatic definitions that are relevant for further research. However, ADTrees are primarily designed for visualization and require manual mapping of countermeasures – something our model is seeking to remedy. Further works in the area, and at the same time the foundation for Kordy et al.'s work, include attack and protection trees for physical security [13] and attack trees with a temporal component [28].

Many other approaches can be subsumed as taxonomies designed to help analysts or researchers counter specific threats. For example, Mirkovic and Reiher [47] present two taxonomies for classifying attacks and defenses against DDoS attacks. Their approach is to highlight commonalities and important features of attack strategies, a task that is done manually in the light of the problem's complexity. While a common classification scheme is important, our model goes beyond this vital first step and aims to deliver automation support for mapping general attack tasks as described by pattern repositories like CAPEC [48] to standardized defensive controls. This ultimately serves as a foundation for anomaly interpretation and mitigation planning.

2.2. Anomaly Detection and Interpretation

Anomaly-based malware or intrusion detection systems are found in many a proposed solution. However, it is rare to see it combined with a semantic component that is dedicated to the automated interpretation of the generated traces, logs, or alerts.

2.2.1. General

The shift of focus towards semantic awareness is visible in several, more general works. For example, Anagnostopoulos et al. [2] present a system for the application of semantics to general intrusion scenarios. The authors seek to classify and predict attacker intentions using a Bayesian classifier paired with a probabilistic inference algorithm. Their semantic model includes both legitimate and illegitimate actors, activities in the form of sequential events, concrete commands issued, and an overall state of attack.

Putting a novel spin on anomaly detection in general, Gautam et al. [19] present a multi-kernel learning approach for One-Class Classification (OCC), an approach where data of only one out of n classes in the dataset is used for training. The authors present an alternative to existing One-Class SVM methods [62] and the Multi Kernel Anomaly Detection (MKAD) algorithm [9]

by locally assigning weight to each kernel. While AIDIS relies on binary and multi-class SVM to perform its distinction, the OCC approach will have to be investigated for scenarios that cannot rely on knowledge about classes other than the one defining e.g. benign baseline behavior.

For unsupervised anomaly detection, Zhang et al. [79] introduce a density-based system using the Gaussian kernel function. The objective is to improve outlier detection in non-linear data by assessing the similarity of data points through measuring the local density between a point and a set of its neighbors. In AIDIS, we use three alternative approaches to unsupervised learning for our mostly sequential data, which typically revolve around strings instead of numerals: Grammar inference, heuristic clustering of traces for the generation of template behavior, and text similarity hashing (see Section 4.5.1). Despite the differences in input, Zhang et al.'s take on anomaly detection warrants further investigation as e.g. component in unsupervised trace clustering.

Noble and Cook [54] explore graph-based anomaly detection through the identification of repetitive substructures within graphs as well as by determining which subgraph of interest consists of the highest number of unique substructures and therefore stands out the most. The introduced system is also able to measure the regularity of a graph using conditional entropy. Being mostly formal in nature, the approach does not consider attack semantics. Most other graph-based systems for intrusion detection scenarios discuss attack graphs, which put the focus on (network) vulnerability analysis and the sequence of events leading to a state of compromise [55, 64].

2.2.2. Host Activity

Kruegel et al. [34] describe a classical approach to detecting anomalies in call sequences. Their system is designed to detect attacks against privileged applications. To this end, it analyzes the relation between system call arguments and calling contexts. Among the anomalies considered are string length, character distribution, as well as the occurrence of certain characters. In contrast, AIDIS mainly considers file system activity linked to a central graph node representing the calling process.

Dolgikh et al. [11] conduct dynamic behavioral analysis of applications. Their system is capable of automatically creating application profiles for both malicious and benign samples. It considers recorded API calls that are subsequently transformed into a labeled graph representing a stream of system calls. Graphs are compressed using a genetic data processing algorithm in order to extract behavioral profiles. There is no classification interpretation or interpretation of the resulting data.

Anderson et al. [3] present a detection algorithm based on the analysis of graphs constructed from dynamically collected instruction traces. Working with simplified assembly sequences represented as Markov chain and subsequently transformed into a weighted directed graph, the general level of abstraction is lower than in AIDIS, which primarily uses generalized API calls. For classification, Anderson et al. [3] use SVM on previously created similarity matrices.

The system's specific results are further discussed in Section 5.4, where they are compared to AIDIS' binary classifier.

Touching the network domain, Jacob et al. [26] present Jackstraws, a system designed to identify command and control (C2) communication. Unlike other network-centric approaches, Jackstraws uses host events captured through dynamic analysis. Association of network activity to local processes is achieved through behavior graph modeling of data flows between individual system calls. Graph templates for C2 pattern similarity matching are mined from a known set based on a technique introduced by Yan and Han [78], followed by a clustering stage. While Jackstraws is potentially vulnerable to certain obfuscation and mimicry attacks [74], its unique approach to detecting C2 traffic makes it interesting for both APT detection and knowledge generation. Jacob et al. [26]'s work is further discussed as part of the comparative evaluation in Section 5.4.

2.2.3. Network Domain

On the network traffic side, Münz and Carle [52] present TOPAS, a traffic flow and packet analysis system compatible with NetFlow and IPFIX. The system's algorithm uses threshold-based detection via pre-defined values as well as outlier detection through the comparison of a sample to previously learned, nominal behavior. While this offers a good foundation for traffic anomaly detection, the link to local processes and applications is not investigated.

The system presented by Ambwani [1] focuses on full network traffic dumps associated to DoS, privilege escalation, and other remote threats. It is similar to AIDIS in its use of SVM multi-class classification for the identification of various attacks, underlining the feasibility of such ML approaches for classification tasks. Anomaly detection or threat modeling is not part of the system. We compare the system's accuracy to AIDIS in the evaluation in Section 5.4.

The work by Vance [73] is one of the few approaches that focus directly on APTs: He describes a flow-based monitoring system that uses statistical analysis of captured network traffic data to detect anomalies – instead of event-based deviations. Vance's system uses change detection to identify flows that hint at command & control traffic, data mining, or exfiltration activities.

As above works exemplify, none of the solutions quite manage to bridge the gap between system events (be they function calls or traffic flows) and a truly meaningful representation of an attack in its entirety. Closing this semantic gap is one of the main goals of the system presented in this paper. For more related work in the domain of semantics-aware APT detection, refer to [38].

3. System Design and Model

Our proposed system revolves around the hypothesis that the observation of ubiquitous OS kernel processes is a feasible alternative to sample-focused analysis, where extracted binaries are checked for malicious properties and behavior. Unlike suspicious samples, system processes are present at all times and do not need to be identified prior to analysis. However, it is not entirely clear

which processes deserve our scrutiny the most – something that AIDIS is built to address through its highlighting of expressive system applications.

Furthermore, we hypothesize that classifying identified anomalies is preferable to classifying entire traces of unknown behavior. Instead of processing all captured events associated to a process or a full user session, the system identifies and extracts anomalous events prior to classification. This reduces the amount of data necessary for interpretation and provides the analyst with not only a pattern of potentially harmful activity, but also a more accurate verdict as to which class/stage of attack it may belong to.

Since the computational complexity of any graph-based solution is likely to be high, any such anomaly detection and explication system will have to consider means to reduce processing times. With AIDIS, we opted to utilize compression through grammar inference, which transparently creates rules from sequences of events that are synonymous to nominal behavior – in contradistinction to uncompressed terminals representing outliers. By replacing commonplace activity with rules, we can save both time and computing resources, while at the same time offering advanced threat formalization capabilities.

Lastly, we hypothesize that the interpretation of previously disseminated anomaly data (i.e. events that constitute a deviation) by mapping it to a dedicated attacker/defender model can, in conjunction with automated anomaly classification, be used to explain threats in a comprehensible manner. For this reason, we map all results to PenQuest, the model component of AIDIS, which provides a link to our enhanced APT kill chain as well as various threat vocabularies.

In summary, the proposed system provides the following functionality:

1. Identification of relevant OS kernel processes for continuous, sample-independent monitoring;
2. Compression of event data through grammar inference;
3. Automated detection and classification of anomalies;
4. Mapping of classified anomalies to a model for threat explication and reasoning.

Before a technical implementation can be approached (see Section 4), we have to consider several formal, semantic, and strategic factors. In the following, we discuss these initial premises by following a design checklist (see Table 1) developed for the assembly of semantics-aware systems countering advanced threats to information systems.

3.1. Design Checklist

The design of the system, which has been independently discussed in [41], is based on the roadmap for a conceptual APT defense system introduced in a survey by Luh et al. [38]. In order to fulfill the requirements for the comprehensive detection and analysis of targeted attacks we followed the presented checklist and extended the design with the ability to explain detected anomalies in behavioral data through a combination of classification and threat modeling.

	Prediction	Prevention	Intelligence	Correlation	Detection	Analysis	Response
G met [5/7]		✓	✓	✓	✓	✓	
	Malware	Host intr.	Network intr.		Attribute	Behavior	Context
T countered [3/3]	✓	✓	✓	A used [3/3]	✓	✓	✓
K for T [3/3]	✓	✓	✓	G for A [3/3]	✓	✓	✓
Correlation for T [3/3]	✓	✓	✓	Corr.: A [3/3]	✓	✓	✓
	Threat info	System logs	App logs	Network events	Local events	Binary/raw	Alerts
I incorporated [3/7]	✓			✓	✓		
	Recon	Weaponiz.	Delivery	Exploitation	Installation	C2	Actions
APT stage covered [6/7]	✓		✓	✓	✓	✓	✓

Table 1: Design checklist after Luh et al. [38]. Capabilities and properties of AIDIS are highlighted. Prevention through risk assessment and awareness building is provided by the PenQuest meta model (see Section 3.3) that ties the technical components (intelligence, correlation, detection, analysis) together. While the resulting anomaly reports and scores enable a defender's response, the mitigation of the attack itself is not part of AIDIS. Input data includes kernel events describing local and network events; threat information is brought into the mix by the underlying model. Abbreviations: G...goal, T...threat type, A...analysis technique, K...knowledge generation, I...input data type.

Referencing the design objectives for semantics-aware detection and analysis systems [38], AIDIS fulfills the suggestions for the scope of implementation– and more. Table 1 provides an overview. Specifically, our approach considers both OS and network events, fuses threat detection as well as attack analysis into a behavior-based anomaly detection system able to classify, extract, and interpret hostile activity, and combines threat intelligence and response into a common model. The system's detection methods and analysis techniques encompass anomaly detection on behavioral data, threat context, classification, and reusable (attack) patterns extracted from the original corpus. These unique combination of features helps to make AIDIS an expedient response to the increasing complexity of targeted attacks.

In the following subsections, we talk about the general threat definition as well as the underlying attack/defense model, leading up to the technical implementation.

3.2. Threat Definition

For a definition of high-level threat stages we decided to extend the cyber kill chain model by Hutchins et al. [24]: Every APT stage is further split into subcategories (see Figure 1) that are ultimately linked to concrete attack actions provided by the the underlying attacker/defender model. For APTs, attack stages typically have successor stages that may be executed once its predecessor has been successfully completed. For example, Exploitation attacks require the prior completion of a Delivery action, lest the utilized malicious code cannot be

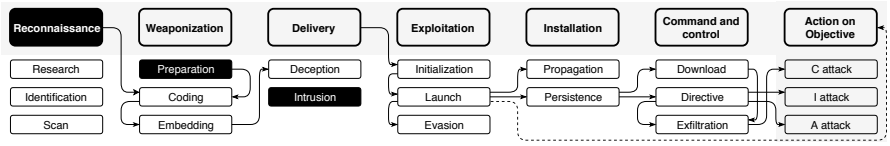


Figure 1: APT stages after [24] and [39]. The arrows represent which stage (i.e. action associated to the stage) needs to be completed before another stage action can be executed. The dashed arrow shows the simplified dependency for non-persistent attacks. Black boxes represent possible start points for kill chain traversal.

executed on the target.

Specifically, the APT kill chain categories we use to define the general threat include:

- **Reconnaissance**: Research into the target and scanning of related assets for information. Subcategories include *Research* using public search engines, *Identification* of systems through e.g. fingerprinting, as well as *Scan* actions, where a victim system is actively probed for weaknesses and topological properties. Successful reconnaissance enables the procurement and weaponization of vulnerabilities.

- **Weaponization**: Preparation of exploits and the malicious misappropriation of code. Weaponization mostly takes place at the attacker's premises and is therefore nigh impossible to detect. Its subcategories are *Preparation*, which includes exploit searches and targeted research, the *Coding* of exploits and tools, as well as *Embedding* the prepared or purchased malware in websites, mail messages, or other, ostensibly harmless media.

- **Delivery**: Delivery actions describe the process of gaining access to a target system or of smuggling payload into the victim's perimeter. Specifically, we differentiate *Deception* attacks that use logical or physical social engineering to fool the victim and straightforward *Intrusion*: Here, the attacker actively tries to penetrate the target's IT infrastructure.

- **Exploitation**: In this stage, a payload or attack code is actively executed on the system. During *Initialization*, malware or an exploit is prepared for launch by abusing a system weakness or causing changes in configuration. *Launch* describes worker processes, threads, services, or modules that are started, marking the point in time where malicious code commences operation. The *Evasion* subcategory encompasses techniques that hinder or prevent the analysis of an ongoing attack.

- **Installation**: This stage covers *Propagation*, which is all about spreading malware infections and the vertical traversal towards the target. *Persistence* attacks, on the other hand, attempt to establish a permanent foothold in a system.

- **Command and Control**: The C2 channel of an APT is responsible for communication between the victim and the malicious controller. The stage consists of the *Download* category, which includes patching and update mechanism that alter or expand the original function of malware or exploits, the *Directive* category, which subsumes commands sent via the C2 channel that potentially alter an attack's original purpose, and the *Exfiltration* aspect, which includes the smuggling out of e.g. gathered information.

- **Actions on Objective**: These actions encompass the actual victim attack task performed after going through some or all of the above kill chain stages. They correspond to the CIA triad of information security [66]: Confidentiality, integrity, and availability. Depending on the type of attack, these actions might include the theft of intellectual property, alteration of financial data, or the unexpected shutdown of a resource.

The adapted APT kill is only one part of the model underlying AIDIS. In the following, we detail the gamified component of PenQuest and how it can be used to explain, simulate, and help mitigate a threat.

3.3. Attack Modeling

For attack modeling and the subsequent interpretation of classified system behavior, we utilize PenQuest [43, 44], our versatile attacker–defender meta model that takes the definition of threat stages and provides concrete actions based on accepted security languages and standards. See Figure 2 for an overview of the model.

Specifically, PenQuest allows for simulating time-enabled attacker/defender behavior as part of a dynamic, imperfect information multi-player game that derives significant parts of its ruleset from established information security sources such as STIX [4], CAPEC[1], CVE[2]/CWE[3], and the NIST SP800-53 security & privacy controls standard [27]. Attack patterns, vulnerabilities, and mitigating controls are mapped to counterpart strategies and concrete actions through practical, data-centric mechanisms. The gamified model considers and defines a wide range of actors, assets, and actions, thereby enabling a detailed assessment of cyber risks while giving technical experts the opportunity to explore specific attack scenarios in the context of their own infrastructure.

In PenQuest, *actions* are at the core of the joint model. They link real-world service and actor behavior to concrete data points such as observable attack patterns or event sequences. Formally, an action X is defined as n-tuple of typical length $n = 11$, whereas the model's flexibility allows for the omission of unneeded elements. Simply put, an action is performed by an actor and further enabled or disabled by equipment, policies, and tools. It is assigned a category

[1] http://capec.mitre.org/
[2] https://cve.mitre.org/
[3] https://cwe.mitre.org/

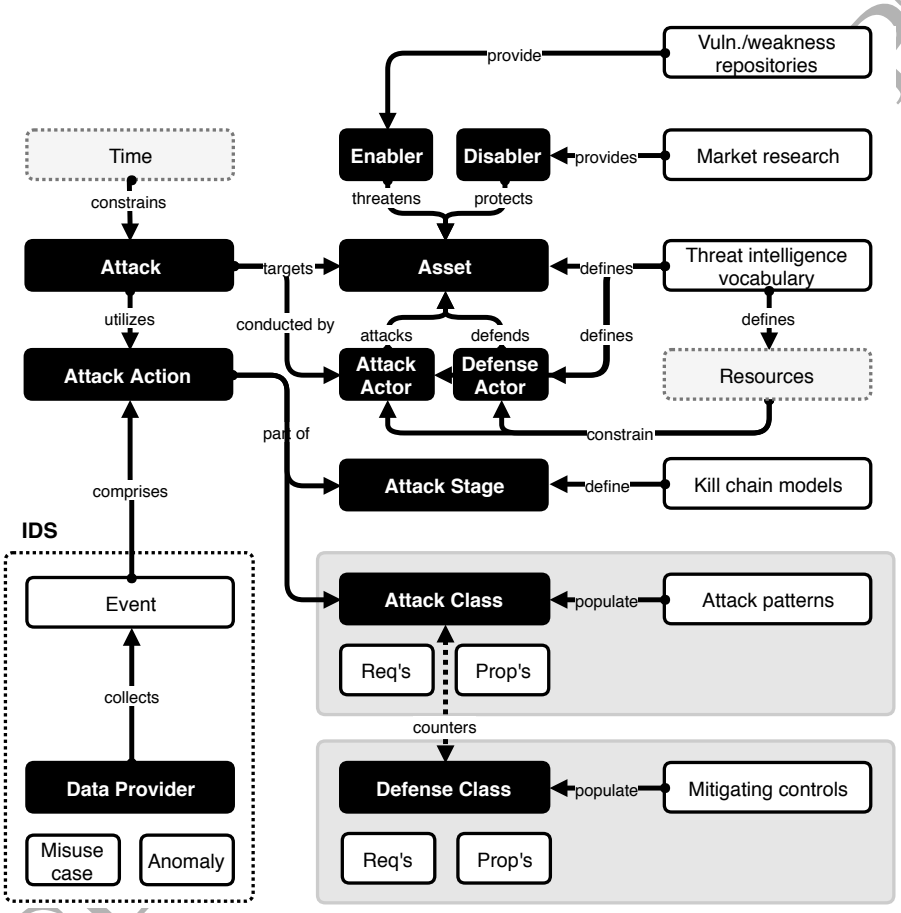


Figure 2: Simplified representation of the PenQuest meta model [43, 44]. The lower left side depicts the AIDIS data provider (agent) monitoring for anomalies or pattern occurrence, while the right sight shows PenQuest's class structure for a generalized Action X (see definition below).

within the model, including target, usage requirements, properties pertaining to its success, placement in an attack vector, as well as a detection probability. X is defined as follows (subclass properties are omitted for legibility):

$$
\begin{aligned}
X = \langle & \\
& \langle AttackActor\langle Class, Motivation, Attributes, Resources\rangle\rangle, \\
& \langle DefenseActor\langle Class, Attributes, Resources\rangle\rangle, \\
& \langle Enabler\langle Type, Effect, Attributes, Name\rangle\rangle, \\
& \langle Disabler\langle Type, Effect, Attributes, Name\rangle\rangle, \\
& \langle Victim\langle Type, Name, Exposure, Parent, VectorParent, \\
& Configuration, Knowledge, Status, Integrity\rangle\rangle \\
& \langle AttackClass\langle Stage, PatternClass, Mode\rangle\rangle, \\
& \langle DefenseClass\langle Category, ControlClass, ActionClass\rangle\rangle, \\
& \langle Requirements\langle *Actor\langle Attributes, Resources\rangle, \\
& Victim\langle Exposure, Integrity\rangle\rangle, \\
& \langle Properties\langle Sophistication, SuccessChance, DetectionChance\rangle\rangle, \\
& \langle AttackPattern\langle Impact, ID\rangle\rangle, \\
& \langle Events\langle Type, Time, Sequence, Parent, Operation, Argument\rangle\rangle \\
& \rangle
\end{aligned}
$$

The central part of the model – and subsequently its playable game component – is synonymous to the process of picking a $\langle Victim\rangle$ and executing an attack $\langle AttackClass\rangle$ corresponding to the CIA triad mentioned above, followed by a valid defensive response. We have modeled this basic building block as a Workflow net (see Figure 3) to identify inconsistencies in the model. A workflow net (WF-net) is a strongly connected Petri net (PN) with two unique input (source) and output (sink) places and a reset transition r.

Following the notation of Esparza et al. [14], a Petri net can be defined as a 5-tuple $N = (P, T, F, W, m_0)$ where P is a set of places or states, T is a set of transitions with $P \cap T = \emptyset, F \subseteq (P \times T) \cup (T \times P)$ is a flow relation, $W : (P \times T) \cup (T \times P) \to N$ is a weight function satisfying $W(x, y) > 0 \iff (x, y) \in F$, and $m_0 : P \to N$ is a mapping called the initial marking. In our case, m_0 equals the initial state where a victim has not yet been picked (called "Start"). A reset transition r leads back to the beginning of the attack process and repeats until the victim has been successfully compromised.

Specifically, an attacker picks a $\langle Victim\rangle$ that is either exposed or part of the attack vector. Subsequently, a mode of attack (C, I, or A) is chosen. As depicted in Figure 3 the initial impact of such an attack is determined based on the $\langle Enabler\rangle$ resource and various other properties and attributes. Independently from the level of success of the action, the defender first attempts to detect the undesired activity. In the Petri net, this is again modeled as exhaustible resource that corresponds to the $\langle Disabler\rangle$ class, which i.a. contains various monitoring

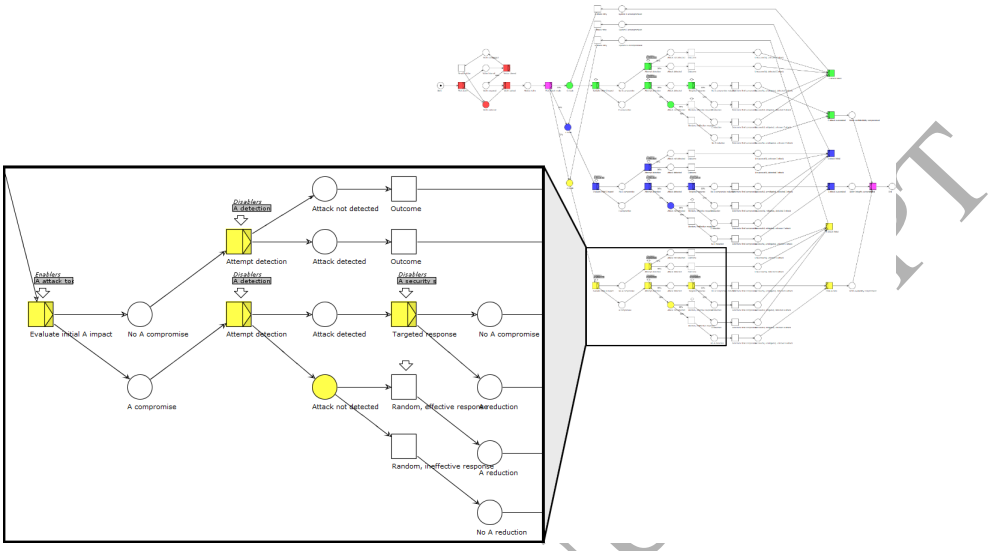


Figure 3: WF-net describing the process of picking and attacking a victim. The net consists of 64 places, 51 transitions, and a total of 133 arcs. Soundness was determined by analyzing the net in WoPeD [17].

systems providing a boost to detection probability. If an attack is spotted, the defending entity can attempt to counter its initial impact by utilizing further $\langle Disablers \rangle$. Ultimately, initial impact and the degree of success in reducing that impact is resolved in PenQuest's only zero-sum game element:

Player 1 chooses action $a \in A$ targeted at victim $v \in V$ ($\langle Victim \rangle$ in our class structure), which is kept secret unless player 2 meets detection requirements (see Figure 4). If action a_v is conducted successfully, player 1 gains a number of points $L(a_v)$ representing the level of victim compromise, which are directly or indirectly deducted from the respective defender's tally (payoff). Independently from the outcome of the detection attempt, player 2 chooses $d \in D$ for victim $v \in V$, attempting to counter the (assumed) action a_v. Points $L(d_v)$ are computed, fully negating $L(a_v)$ in the best of cases. Specifically, this principle applies to victim system integrity and status (see Figure 4), to success chance of hostile attacks and defensive actions, as well as some other attribute or resource bonuses and penalties.

We have again modeled this component as WF-net, describing the process of generating $L(a_v)$ as well as $L(d_v)$, respectively. This second net, with a total of 33 places and 65 transitions, can be used for simulating arbitrary actions generating $L = 0..3$ on both the attacking and defending side. The unveiling of actions is again part of the parent WF-net discussed above.

It is important to bear in mind that the PenQuest's 'victory' conditions are only indirectly affected by these shifts in L, and that an increased numeric distance from the equilibrium of factors other than victim system integrity and

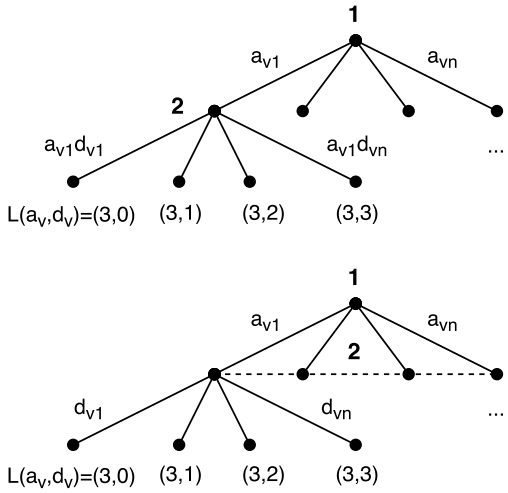


Figure 4: Information set of the defending player for victim system compromise through attacker action a_{v1}, shown in extensive form [36]. In the above case, player 2 successfully unveils a_{v1}, giving him the chance to specifically counteract the gain $L(a_v)$ of player 1 by increasing his own score $L(d_v)$ through defense action d_{v1}. Below tree represents the limited information set for player 2, when a_{v1} remains undetected.

status does not always guarantee one player's domination over the other. In fact, victory is determined by the exhaustion of available (temporal) resources or the successful compromise of the victim asset within an allotted time window.

The link between actions and real-world events (also see Section 4.1) is another integral part of the model and is depicted in Figure 2. Each attack action corresponds to an attack stage and class, the latter of which is populated by concrete attack patterns. *Attack patterns* are identified by an ID and their impact on the CIA triad [66]. Attack patterns link the modeled action to specific hostile activity as described in the CAPEC schema [48], which can be understood as a public database describing specific information system attacks and their technical properties. Ultimately, the model maps attack patterns to a set of recorded *events* with specific underlying operations and arguments, triggers (parents), timestamps, and sequence numbers. The *Pattern* event type describes event sequences that directly represent the action, while *anomalies* contain a behavioral deviation from a baseline. AIDIS primarily focuses on the latter. The modeling of unique events closes the gap to the data layer and allows us to lower the level of abstraction to the actual systemic representation. For more information about PenQuest and the model-to-data mapping, see [43] and [44].

4. Core Components

AIDIS is composed of several components enabling the underlying anomaly detection and knowledge explication process. The initial tasks encompass the

Component	Sentiment analysis	Grammar inference	Graph anomaly det.	Anomaly classific.
Section	4.3	4.4	4.5.1	4.5.2
Original paper	Luh et al. [40]	Luh et al. [42]		
Optional	Yes	Yes	No	No
Algorithm(s)	LLR [12, 40]	Sequitur [53, 42]	Kuhn-Munkres [23]	RF [37], SVM [8]
Knowledge gen.	✓✓	✓✓	✓	✓
Anomaly det.	✓✓	✓✓	✓✓	✗
Classification	2 classes	✓	2 classes	n classes
Interpretation	✗	✓	✗	Yes
Learning	Supervised	Unsupervised	Supervised	✗
Complexity	$O(n)$	$O(n)$	$O(n^3)$	$O(k * nlog(n))$, $O(n^2)$

Table 2: List of AIDIS components beyond collection and preprocessing. Knowledge generation describes the extraction/inference of information about event sequences and anomalies in general. Anomaly detection capabilities allow for the identification of anomalous behavior through a score, statistical analysis, or visual assessment. Classification enables the separation of the result into malicious, benign, or more granular threat categories, while further (anomaly) interpretation is enabled through the link to our PenQuest model. Different learning modes are identified, as are the component's computational requirements. Legend: ✓✓...fully supported/key feature, ✓...limited support/byproduct, ✗...not supported or not part of the primary purpose.

acquisition of data on a number of monitored machines and/or network devices as well as the transmission and translation of kernel events to a clean database format. In stage 2, we extract OS processes suitable for observation through sentiment analysis. Following optional data compression using grammar inference, we link events by their contextual parent and construct traces in the form of star structures, simple graphs that describe the operations conducted by each process within a specific time range. From a baseline of benign system behavior we then extract one or several process-unique templates that are subsequently used to check new activity for anomalies by measuring the edit distance between the simplified graphs.

Our approach not only calculates deviations but also returns a human-readable list of actions that constitute the identified anomaly. This list is ultimately classified using both a Random Forest and SVM-based approach. The resulting behavioral patterns are mapped to the aforementioned PenQuest model, thereby linking each anomalous action to an APT attack stage and semantic description. Figure 5 and Table 2 provide a full overview of the system components. The following subsections detail each stage and provide technical specifics.

4.1. Data Collection

AIDIS works with *events* collected directly on the host (endpoint). Event traces (i.e. ordered lists) are typically defined as descriptions of operating system kernel behavior invoked by applications and, by extension, a legitimate or illegitimate user. Individual events are abstractions of raw system or API calls that yield information about the general behavior of a sample [75]. API calls may include wrapper functions (e.g. `CreateProcess`) that offer a simple interface to the application programmer, or native functions (e.g. `NtCreateProcess`)

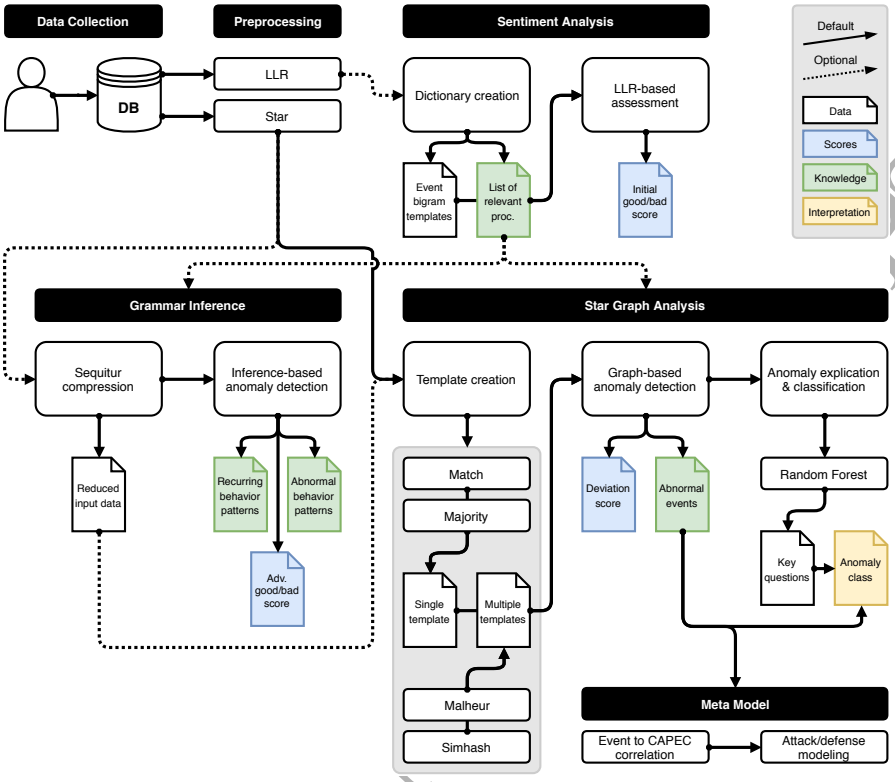


Figure 5: AIDIS system overview. Optional sentiment analysis is used for extracting kernel processes that deserve special attention, while the grammar inference component offers data reduction and unsupervised anomaly detection. The core knowledge extraction and anomaly detection component utilizes star structures for template generation and matching. Event interpretation is realized through RF and SVM classification applied to the resulting anomaly reports. The link to our meta model (see Figure 2) semantically enriches the information and helps plan an appropriate response.

that represent the underlying OS or kernel support tools. In its abstracted form, a contextual event trace might look like this:

With AIDIS, process and network event data is collected directly from the Windows kernel. We employ a driver-based monitoring agent designed to collect and forward a wide range of events to a database server. This gives us unimpeded and fast access to events depicting various OS operations [45]:

- **Process events** – Whenever a process is started or stopped, the monitoring system registers a new event. Next to PID and paths, we record parent and contextual information such as ownership data. Process events are at the heart of our system: Every other type of event is ultimately associated to a process through its PID and timestamp (see below for more information).

ProcessEvent	Start: 'shell.exe' (PID 220)
ImageLoadEvent	Load: 'library.dll'
RegistryEvent	Open: 'HKLM/Software/.../Run'
RegistryEvent	Add: REG_SZ ('evil.exe')
FileEvent	Create: 'evil.exe'
ProcessEvent	Start: 'evil.exe' (PID 224)
ProcessEvent	Terminate: 'shell.exe' (PID 220)

Table 3: Example trace of events as chronological list.

- **Thread events** – Some events are triggered by individual threads instead of processes. The information logged by the agent is largely similar to process events; the main identifier for threads is the thread ID (TID).
- **Image load events** – Most processes load additional resources (functions) stored in various program libraries (DLLs). The nature of a DLL can give a good indication as to which behavior the binary executable will exhibit during its lifetime.
- **File events** – File events are logged when a file is read, created, accessed, modified, or deleted. Logging file interaction is important since processes can interact with virtually every file stored on the disk. Attack-related file events can e.g. help identify dropped executables or data theft.
- **Registry events** – Applications use the registry to save user and program settings while other hives contain startup programs or file type settings. Since it is a common target for espionage and system manipulation attacks, monitoring registry events is critical for any Windows-based detection solution.
- **Network events** – Network events encompass the handling of inbound and outbound connections as well as the access to general OS networking resources. Depending on the nature of the process, network events can be used as indicator for malicious behavior, since many malware variants contact remote systems for e.g. command & control purposes.

The relative ease of monitoring as well as the semantic expressiveness of kernel events and network operations make such traces ideal for dynamic software and, by extension, malware analysis as well as application classification. The system introduced in this paper uses this rich repository of behavioral data to compile sentiment dictionaries as well as graph-like star structures of event sequences that can describe not only a single application, but a system session in its entirety. This approach is detailed in the following subsections.

4.2. Preprocessing

All previously collected events are linked through their parent process in order to establish a semantic connection between action and cause. This is real-

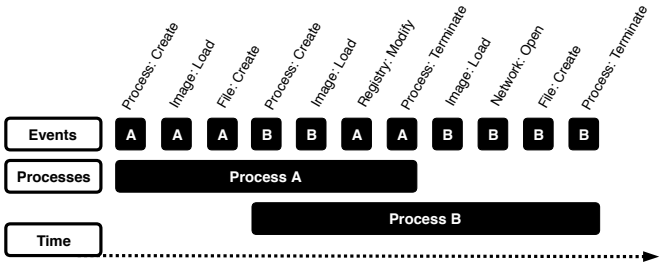


Figure 6: Orphan events in a context-unaware trace [40]. Smart traces counter the issue of intermixing events that belong to different processes or threads by rearranging them into a chronology by context (here: A followed by B).

ized through two attributes that are present in all the data collected by the host monitoring agent: Creation time, and the PID that forms a unique identifier for each process. Threads work in a similar fashion. Like PIDs, thread IDs (TIDs) are appended to other event types (e.g. registry events). Ultimately, each process or thread created by the respective event can incorporate an arbitrary number of child events, depending on its nature and run time.

Both process and thread events can be used to construct an event tree depicting the flow of file system activity that helps to determine specific dependencies between processes and general events. Concatenated into a full system graph, the sequence of events constituting a monitored session are assembled without orphan entries (depicted in Figure 6) interrupting the process flow by grouping them by their associated process and thread. These pseudo-chronological *smart traces* [40, 45] are the basis for all follow-up computation.

Further preprocessing includes the normalization of non-uniform IDs such as user names, security identifiers, and temporary folder names. This is done to make data more comparable across systems and to prepare the traces for future anonymization.

4.3. Sentiment Analysis

AIDIS uses an approach akin to sentiment analysis [18] for generating initial knowledge about relevant OS processes. In this optional stage, we determine the most expressive process candidates for later investigation. At the same time, this kick-off stage computes a first benign/malicious score that provides us with a tendency towards general harmfulness for the provided dataset. The resulting verdict can be used as additional feature in the final classification stage of the AIDIS process. Figure 5 and Table 2 show how this component fits into the big picture of AIDIS.

The details of the sentiment analysis component have been previously discussed as stand-alone anomaly detection solution based on inferred dictionaries containing 'malicious' vocabulary [40]. While the evaluation found in this article instead focuses on the selection of relevant processes for continuous monitoring

Application	Purpose
Compression	Lossless reduction of input data size
	Reduction of processing complexity (follow-up stages)
Anomaly detection	Detection and extraction of deviating behavior
Baselining	Identification of common patterns in traces
Visualization	Visual presentation of inferred rules through KAMAS [76]
Discovery	Interactive filtering and extraction of terminals/rules
	Rule labeling, storing as part of a knowledge base
	Highlighting of known rules

Table 4: Applications of SEQUIN. With AIDIS, we primarily use compression and baselining to reduce processing complexity and to generate behavioral templates of processes. (Visuals-assisted) anomaly detection is used in scenarios where supervised learning is not feasible or possible, mandating manual investigation.

(see Section 5.3.1), the technical foundations of the core component remain the same. Please refer to [40] for more information.

4.4. Grammar Inference

Grammar inference through Sequitur compression is the second optional stage in the AIDIS process. It is used to losslessly reduce the amount of input data for the more computationally expensive final stages, while providing a semi-supervised approach to identifying potentially interesting portions in arbitrary event sequences and smart traces.

For the purpose of compression we utilize prior work, SEQUIN [42], a grammar inference system based on the Sequitur algorithm, which constructs a context-free grammar (CFG) from string-based input data. Specifically, Sequitur is a greedy compression algorithm that creates a hierarchical structure from a sequence of discrete symbols by recursively replacing repeated phrases with a grammatical rule [53]. The algorithm creates this representation through two essential properties, which are called *rule utility* and *bigram uniqueness*. Rule utility checks if a rule occurs at least twice in the grammar, while bigram uniqueness observes if a bigram occurs only once. A bigram in this context describes two adjacent symbols or terms.

The full rule extraction and evaluation process is detailed in [42]. The article describes the application of our adapted Sequitur system on smart traces of kernel events associated with arbitrary processes and other security-relevant data, proving a full example grammar. In short, SEQUIN has a wide variety of applications that go beyond AIDIS: Table 4 provides an overview.

The reduction of input data in particular can be helpful to decrease the complexity of lengthier analysis tasks, such as the graph-based approach discussed in this paper (see Section 4.5.1): By using SEQUIN, it is possible to slim down the input corpus to only relevant n-grams ($n \geq 2$), instead of working with the full, unfiltered set of event or code snippet unigrams. SEQUIN's grammar transformation mechanism [42] also enables us to work with an automatically generated placeholder variable instead of several compound terminals.

See Section 5 for an evaluation of the compression component in the context of AIDIS. Refer to [42] for more information about this and other SEQUIN components.

4.5. Star Graph Analysis

Whether or not relevant processes have been identified using sentiment analysis and input data has been compressed, the key analysis component of AIDIS can be executed at this point: We utilize *star structures* to create a by-process representation of event sequences that encompass single process launch behavior, its full run time, or even entire multi-process system sessions. Star structures are a means to reduce the complexity of a known NP problem to polynomial complexity [23]. Instead of searching entire system session graphs for matching patterns, the star structure approach breaks down the computation into a triplet of nodes (vertices) connected by a labeled edge, denoted as $G = (U, V, E)$, where U and V are nodes and E is the respective edge. The attached label is used as basis for minimal cost calculation of same-size star structures. Specifically, we utilize bipartite graph matching based on the Hungarian (Kuhn-Munkres) algorithm [35], where every star is processed as a matrix. Graph edit distance calculation determines the minimal costs of relabeling the nodes and edges of a graph G to match a target graph H. The edit path $P_{G,H}$ can be understood as a sequence of transformation operations σ. The final graph edit distance is determined by the cheapest of all edit paths between G and H.

Compared to full graph matching, this approach is typically considered to be a faster, but less precise approximation, as it only matches the immediate neighborhood of one node at a time. In our system, we use an adaptation of the Hu et al. [23] approach that combines n bipartite graphs into one star representing a single process. This makes the effect on result accuracy far less pronounced: With a focus on individual processes, our input data can already be reduced to star structures without significantly compromising trace semantics. This is due to the fact that we anchor every event to a trigger (parent) process (see Section 4.2) that actively invokes respective actions, making this process the natural center vertex of a star-shaped graph. In our system, elemental operations for determining the minimal cost graph edit distance between individual elements are not limited to relabeling nodes, but consider the connecting edges as well. There are two operations that contribute to the edit score:

Vertex edit operations encompass single vertex relabeling σ_{RV} as well as both an insert vertex σ_{IV} and a delete vertex operation σ_{DV}. Semantically speaking, each vertex is akin to an unspecified system event (event type plus parameter, sans event type) as introduced in Section 4.1. Depending on the type of operation, the respective event in H is either new (insert), missing (delete), or has been altered (relabel) from the baseline G.

Edge edit operations, on the other hand, primarily consider the edge relabeling cost σ_{RE}. We opted to dynamically assign individual relabel costs based on the type of event considered, making the approach fully capable of assessing event similarities. For example, σ_{*V} will drastically increase in cost when

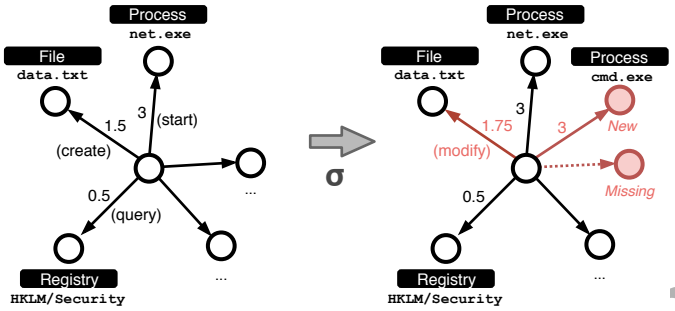


Figure 7: Example event representation for process `svchost.exe` (central node). Target graph H (right) differs from the baseline graph G (left) by several additional or missing events, depicted as red nodes. Mere changes to the edge label (different operation type applied to the same object) are considered as well. Graph transformation σ is derived using the Kuhn-Munkres algorithm [35].

e.g. converting a semantically inexpensive file 'read' event to a relatively high-impact 'delete' event. The type of operation (numerical representations of e.g. create, modify, delete, start, and stop operations) considered by σ_{RE} determines the final cost of edge relabeling. See Table 7 for a list of event types and their experimental labels. Combined with a vertex operation, all possible changes to a process can be quantified.

Figure 7 shows a simplified example. In the depicted case, the base graph consists of various vertices representing events such as the creation of a file, the start of a process and an open/read operation conducted in the `HKLM/Security` hive of the Windows registry. When comparing the baseline graph G to a target H, the introduced Hungarian graph edit distance approach will use σ to determine the minimal cost of transforming G to H. In case of the exemplary file event interacting with `data.txt`, this edit distance is a mere 0.25, since a single σ_{RE} operation is sufficient to transform the bipartite graph $G(svchost.exe, 1.5, file.txt)$ to $H(svchost.exe, 1.75, file.txt)$.

This method for determining the minimal edit distance between two star-shaped graphs is used as the foundation for context-aware anomaly detection utilizing supervised learning on a per-process basis.

4.5.1. Star Anomaly Detection

The required transformation operations and, by extension, the minimal graph edit distance between two star structures can be used to determine the event-level deviation between instances of the same process. In order to automatically determine thresholds for each observed process, we first need to create a template from a benign environment. Only then can we match base to target graphs and disseminate their differences.

We have implemented the generation of baseline templates in 4 different ways (also see Table 5), two of which generate a single template, while the other two produce any number of templates ranging from 2 to n, depending on the

Method	Perfect match	Majority	Prototype	Sim. hashing
Algorithm(s)	String comp.	String comp.	Malheur [59, 70]	MinHash [5] Jaccard sim. [25]
Deterministic	✓	✓	✓	✗
Template count	1	1	n	n
Reduction	✗	✗	✗	✓
Complexity (extraction)	$O(n)$	$O(n)$	$O(k * n)$	$O(n^2)$
Complexity (matching)	$O(n)$	$O(n)$	$O(k * n)$	$O(k * n)$

Table 5: Overview of star graph template creation methods. Single template approaches are well suited for simple processes with little semantic variance. Multiple templates are needed for complex, multifaceted processes. Similarity hashing is the only method that supports the reduction of Malheur-derived templates but is less accurate when used for extraction due to its non-deterministic nature. It is computationally advantageous for later template matching to reduce Malheur templates using similarity hashing.

complexity and versatility of the process/session in question. In each case we take a set of benign process graphs and extract an optimal representative using one of the below methods:

Perfect match – Here, we extract identical events found in each iteration of a process (i.a. the 'smallest common denominator') and assemble an entirely new graph. This creates a sleek template that enables the analyst to primarily focus on hitherto unobserved events. However, there is an performance-for-accuracy trade-off that results in higher mean edit distance values.

The approach proved to be best suited for background processes with a single purpose and little user interaction, such as certain device drivers or task bar tools providing static context menus.
Use-case examples: Quick Access for Intel Grapics (`igfxtray.exe`), Office Telemetry Agent (`msoia.exe`), Windows Power Management (`powercfg.exe`).

Majority – This method picks the most common base graph from the input set and converts it to a template without altering its contents. While slightly more accurate than 'perfect match', this approach struggles with processes that show greater variety in their benign instances due to their multifaceted nature.

Majority extraction is generally similar to the perfect match approach. It is best utilized for processes where major deviations from the baseline are not tolerated. Examples would include applications that exhibit behavior following a set schedule or isolated processes with little system interaction. Despite its limitations, majority matching is still the best choice for complex processes where a multi-template approach is not desired for e.g. performance reasons. Certain concessions regarding accuracy have to made, however, as is discussed in the Evaluation chapter below.
Use case examples (in addition to the above): Google Updater for Chrome (`GoogleUpdater.exe`), Windows Activation Client (`slui.exe`), Java Launcher (`java.exe`).

Prototype extraction – Especially useful for diverse processes, this approach uses the Malheur algorithm [59, 70] to extract not one, but several prototypes representative of the various aspects of a single process. This promises significantly improved accuracy when assessing more complex OS applications. However, the resulting number of templates sometimes negatively impacts performance, which is why we added a second component that intelligently merges templates using similarity hashing.

Prototype extraction is best used for complex processes which control a wide range of OS functions and that are not necessarily similar in their behavior. In a best-case scenario, each functionality is automatically assigned a template. If newly logged behavior does not correspond to at least one of them, the observed activity is treated as an anomaly.
Use case examples: Windows Generic Host Process (`svchost.exe`), Generic Host Process for Libraries (`taskhost.exe`), Registry Editor (`regedit.exe`).

Similarity hashing – Usable as both standalone alternative for the Malheur approach as well as a reduction mechanism for the same, this take on multi-template creation is based on the MinHash algorithm [5], which builds upon the mathematical concepts of resemblance and containment to measure document similarity. Specifically, we measure the differences between original traces or previously Malheur-extracted templates by their Jaccard distance [25]:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}, where(A,B) \subset U$$

In short, the MinHash algorithm converts a set of tokens from U into n randomly selected and hashed tokens, which are then broken down into bands and compared [58]. Documents are considered similar if the resulting Jaccard distance threshold is exceeded. In AIDIS, the individual document similarity values are mapped to a graph, where representative prototypes are determined by their betweenness centrality (see Figure 8). If betweenness centrality is equal, node in-degree is used instead. The most significant traces, determined by a Pareto score of $\geq 90\%$, are ultimately kept as templates.

Similarity hashing has proven to be a feasible second stage to the prototype extraction approach. For complex processes, it reduces the number of Malheur prototypes to a more workable number without negatively impacting accuracy. Additionally, similarity hashing is well suited to processes that are versatile in nature but similar in their behavior.
Use case examples: File Ownership Tool (`takeown.exe`), Task Scheduler (`schtasks.exe`), Session Manager (`smss.exe`).

Before any anomaly detection can be implemented, we need to set threat thresholds. In our case, they are determined by comparing the generated template(s) to the remainder of the benign input graphs using the Hungarian distance. This yields a minimum, mean, median, and maximum lower-bound edit distance for each process. Depending on the level of scrutiny, each of these

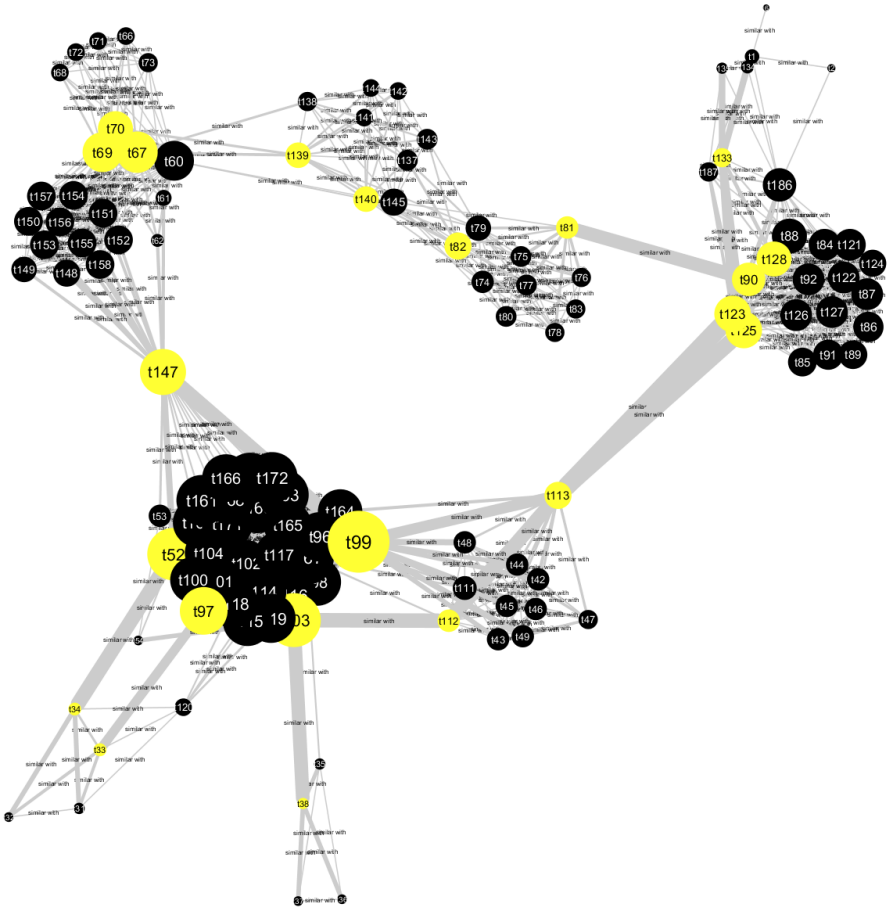


Figure 8: Example template extraction through similarity hashing. Similar templates are determined by their Jaccard distance and graph betweenness centrality. The yellow nodes were determined to have the greatest betweenness centrality score. All matches with a Pareto score of $\geq 90\%$ are ultimately chosen as templates.

distances can be used as anomaly threshold. In our case, the default (mean) threshold was dynamically derived from a high number of test runs.

Armed with one or several templates for each process, we can now check unknown graphs against the predetermined thresholds and extract events responsible for the deviation.

4.5.2. Star Anomaly Classification

As our system focuses on the classification of anomalies instead of unknown system traces in their entirety, the amount of data processed in this explication stage is drastically reduced. Specifically, we seek to explain why the anomaly detection routine has identified a star structure as significantly deviating from the template, thereby disseminating the in-depth knowledge gained in the process. Only afterwards can we commence with anomaly classification.

Knowledge dissemination–One of the advantages of our anomaly detection system lies in the fact that the star depiction of a graph allows for the comprehensive dissemination of semantic information that depicts each and every anomaly in a simple fashion. The analyst is presented a report detailing the events that constitute the respective deviation. Below snippet shows an example `svchost.exe` process being checked against one of its extracted prototype templates:

```
==> svchost.exe [Deviation (threshold):
    300.5 (123.3) --> ANOMALY]
svchost.exe spawns 13 additional threads
svchost.exe terminates 18 additional threads
svchost.exe loads 54 additional images
=> atl.dll
=> bcrypt.dll
=> cfgmgr32.dll
    (...)
svchost.exe sets 6 additional registry entries
=> /HKLM/Software/Microsoft/...
=> /HKLM/System/ControlSet001/...
    (...)
svchost.exe modifies/deletes 6 additional files
=> /Windows/system32/acctres.dll
    (...)
svchost.exe opens 7 additional network sockets
=> 192.168.100.100
    (...)
```

As mentioned in Section 4.2, AIDIS allows for varying levels of granularity. Registry paths can either be normalized to hive names or be processed in their entirety. Abstraction of ID numbers, memory addresses, user IDs is implemented as well – as is pseudonymization of file names, IP addresses, and other personally identifiable information.

Knowledge interpretation–Knowledge dissemination offers interesting information to the analyst but does not yet automate its interpretation. One of the key components of our system is the classification of certain combinations of anomalous events by mapping them to both APT attack stages contained in the model (discussed in Section 3.2), as well as CAPEC attack patterns describing even more concrete adversary behavior (Section 3.3).

The initial version of our system explores event combinations using Random Forest and linear kernel support vector machines (SVM). In the first step, we use the disseminated knowledge to answer over 200 competency questions that are expected to aid in the decision of whether a factor contributes to a malicious objective of a certain kind. These questions include simple Boolean queries into the presence of events over another event (e.g. if the number of thread terminations exceed the number of thread spawns) as well as decisions based on the presence of certain activity tags describing the base functionality of a loaded image (e.g. networking, authentication, user interface, kernel, etc.). The latter is enabled by intelligent tagging (categorization) of more than 1,700 known Windows function libraries, which has been done in advance by parsing both the Windows API section of the MSDN library[4] as well as community sources[5]. Pattern checks determining the use of certain system directories for file events or the assessment of IP addresses are technically possible, but were not implemented at this point: The goal of the prototype system was to avoid fixed patterns as much as possible, as they require constant tuning effort and might be circumvented by a malware's analysis evasion routines.

In order to support the development of expressive competency queries we apply the Random Forest algorithm to determine the mean decrease in accuracy/Gini for each feature, thereby selecting the most significant questions for the respective scenario. For the discrimination of anomaly traces, both binary 'benign' vs. 'malicious' and multi-class classification is used: Based on the response to the competency questions, we get a probability describing the graph's affinity towards a certain kill chain stage or attack pattern. The process was double-checked using a linear kernel SVM with and without hyperplane optimization. See Section 5 for the list of assigned classes as well as detailed evaluation results.

Ultimately, AIDIS maps the resulting verdict (e.g. 'anomaly belongs to class CAPEC-112') and the anomaly report itself to our PenQuest model [43, 44], thereby building our knowledge base of labeled attacks that can then be associated a goal, stage, likely actor, possible countermeasure, and more. In a simplified fashion, star graph events $G = (U, V, E)$ contributing to an anomaly could look like this:

Start node (U)	**End node** (V)	**Edge** (E)
process-shell.exe	process-drop.exe	start (3)
process-drop.exe	image-library.dll	load (1.5)
process-drop.exe	registry-HKLM/Software/.../Run	open (0.25)
process-drop.exe	registry-REG_SZ(evil.exe)	add (0.75)

Table 6: Example event sequence describing the process of making a dropped executable persist through a system restart (CAPEC-564: "Run Software at Logon"). Time stamps and full paths omitted for legibility.

[4]https://msdn.microsoft.com/en-us/library/
[5]https://undocumented.ntinternals.net/

Once appended to the model, the PenQuest action definition [43, 44] would transform the data to a simple instance of the $\langle Event \rangle$ class, which can be easily added to e.g. an ontology representing the model. For example:

$$
\begin{aligned}
Event = \langle \\
&\langle Type = Pattern \rangle \\
&\langle Time \langle Start = 16.53.661, End = 16.53.729 \rangle \rangle, \\
&\langle Sequence = 1 \rangle, \\
&\langle Parent = shell.exe \rangle, \\
&\langle Operation = process\_start \rangle, \\
&\langle Argument = drop.exe \rangle \\
&\langle Sequence = 2 \rangle, \\
&\langle Parent = drop.exe \rangle, \\
&\langle Operation = image\_load \rangle, \\
&\langle Argument = library.dll \rangle \\
&... \\
&\rangle
\end{aligned}
$$

From now on, the data is part of the model and can be viewed in context, presenting analysts with classification and interpretation of hitherto unknown events. At the same time, the umbrella attack causing the anomaly becomes part of the risk assessment process enabled by the PenQuest model. See 5.3.5 for an example mapping based on real-world data.

5. Evaluation

In this section we discuss the experimental setup as well as the individual steps of the AIDIS system, beginning with the identification of relevant processes through LLR sentiment analysis. Figure 5 and Table 2 provide an overview of how the individual components play together. In summary, AIDIS provides the following: 1) Data collection, 2) tree and trace construction with data cleanup, 3) LLR-based sentiment analysis for relevant process identification, 4) optional compression through grammar inference, 5) star graph anomaly detection, 6) anomaly classification (core component), as well as 7) a mapping mechanism of anomaly data to the PenQuest meta model for additional threat semantics. Stages 3 and 4 provide anomaly scores that result in a binary malicious/benign classification, which can be used as additional feature in star graph anomaly classification (stage 6). Multi-class classification in preparation for model mapping is also performed in stage 6.

In this evaluation, anomaly classification is based on star structure anomaly reports. While it is also possible to use the computed anomaly traces of LLR and SEQUIN as training set, the respective components have proven to be better

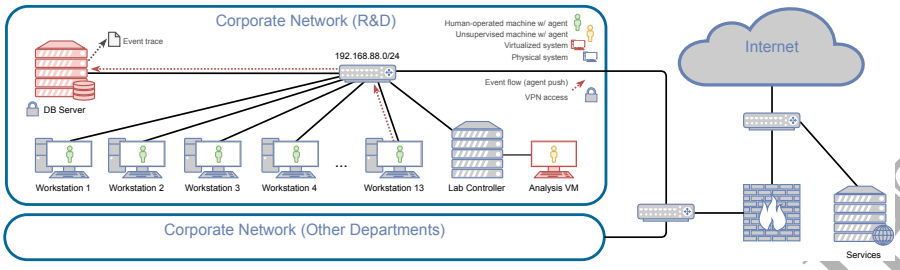


Figure 9: Topology of the testbed network. Event data is pushed to the database server in 3-second intervals, where it is converted to smart traces and made available for AIDIS processing.

suited to process extraction and compression in the context of AIDIS. Nevertheless, LLR sentiment dictionary matching and SEQUIN's unsupervised anomaly detection capabilities have been evaluated as individual systems. Detailed results can be found in [40] and [42], respectively. In the following, we focus on the compound stages of the process and compare AIDIS to 3 similar solutions.

5.1. Experimental Setup

The prototype of the system was implemented in a test-bed environment consisting of 13 physical Windows 7 and Windows 10 computers used on and off by developers and IT personnel of a medium business over the course of half a year. The company, a local security solutions provider, performed regular checks to ensure that the machines in question were not affected by undesired software. One additional virtual Windows 7 instance was utilized for dynamically monitoring malicious software and automated targeted attacks on demand. All machines at least provided common user applications such as Microsoft Office, Adobe Reader, various browsers, as well as widely used OS extensions such as Java SE and the .NET framework. The required host and network event data was collected by a specifically developed kernel driver agent outlined in Section 4.1. Figure 9 provides an overview of AIDIS' testbed topology.

While the system is capable of collecting all the discussed event types, we omitted several of them (e.g. file reads) for practical reasons. Available event classes are listed in Table 7, represented as columns. The respective arguments considered were process, image, file, and registry path/key names, as well as accessed IP addresses denoted as hash values. The type of operation (table rows) was internally processed as numeric value ranging from 0.1 (registry read) to 3.5 (process termination).

The kernel monitoring agent logs all the event types to a central listener that in turn writes the events to a Postgres database server. SQL is used to query the database and to construct the star structures that are the basis for all further processing, which is handled by AIDIS' individual program components (see Section 5.2 for code specifics). Our approach is able to selectively retrieve entire system sessions or pick out individual processes, whereby any temporal

	E	Process	Thread	Image	File	Registry	Network
Start	3.0	✓					
Stop	3.5	✓					
Spawn	0.9		✓				
Terminate	1.1		✓				
Load	1.5			✓			
Read	0.2				✗		
Create	0.75				✓		
Modify/Del.	1.25				✓		
Read	0.1					✗	
Set Key	0.5					✓	
Edit Value	0.25					✓	
Open Socket	2.0						✓

Table 7: Types of events collected by the agent and evaluated by AIDIS. The values for edge E were assigned manually for mapping purposes and in accordance to their approximated impact on the system. Operations marked with an ✗ are supported by the agent but were not considered in the evaluation.

range can be specified. For example, we can process only the first n seconds after an application's launch or extract data from a specific point within its lifetime – which is exactly what was done for our initial PoC evaluation ($n = 10$).

The repository of data included a total of 125 GiB of traces with more than 1.3 billion individual events across all monitored processes, with an event type distribution as depicted in Figure 10. Another 4.3 million (4.5 GiB) events were recorded on the aforementioned analysis VM. For these malicious traces, we executed a total of 1,995 APT malware samples and attack software, ranging from DarkComet [77] and other, unnamed Remote Access Trojans (RATs) to various crypto-miners and tools such as ShoulderSurfer[6], which is used for stealing information from Microsoft Exchange databases. Since AIDIS is not primarily used for malware classification but considers behavior independently, most monitored attack activity is not attributed to specific sample families. We instead use a CAPEC-based classification [48] to describe patterns for e.g. reconfiguring the system or disabling security mechanisms. This ensures that shared behavior is prioritized over family designation, which is typically not available for hitherto unknown samples. See Figure 11 for a distribution of classes used in the evaluation.

We performed only minimal cleanup of the input data by normalizing certain file paths and IDs (see Section 4.1). Largely idle or possibly faulty malware was retained, since such samples are likely to be found in real-world datasets. The data was labeled from the get-go, assigning 22 CAPEC classes in addition to the 'benign', 'idle', and 'crash' categories:

1. BENIGN: Non-malicious execution of the process
2. CAPEC-112: Brute Force
3. CAPEC-131: Resource Leak Exposure

[6] https://wikileaks.org/ciav7p1/cms/page_524353.html

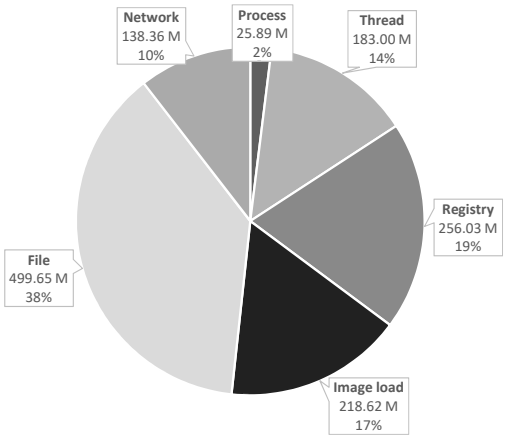


Figure 10: Types of events found in the full benign dataset. While 'process' events generally describe applications launched in the OS, the remainder represent actions triggered by said processes.

4. CAPEC-136: LDAP Injection
5. CAPEC-159: Redirect Access to Libraries
6. CAPEC-169: Footprinting
7. CAPEC-185: Malicious Software Download
8. CAPEC-203: Manipulate Registry Information
9. CAPEC-207: Removing Important Client Functionality
10. CAPEC-242: Code Injection
11. CAPEC-251: Local Code Inclusion
12. CAPEC-389: Content Spoofing Via Application API Manipulation
13. CAPEC-442: Malicious Logic Inserted Into Product Software
14. CAPEC-510: SaaS User Request Forgery
15. CAPEC-549: Local Execution of Code
16. CAPEC-557: Schedule Software To Run
17. CAPEC-564: Run Software at Logon
18. CAPEC-568: Capture Credentials via Keylogger
19. CAPEC-578: Disable Security Software
20. CAPEC-629: Unauthorized Use of Device Resources
21. CAPEC-68: Subvert Code-signing Facilities
22. CAPEC-75: Manipulating Writeable Configuration Files
23. CAPEC-94: Man in the Middle Attack
24. CRASH: Process crashed within 10 seconds
25. IDLE: Process shows insufficient activity for labeling

The numbering of the list corresponds to the class IDs seen in the confusion matrix below (see Table 11). Refer to the CAPEC repository [48] for more information about these particular attack patterns.

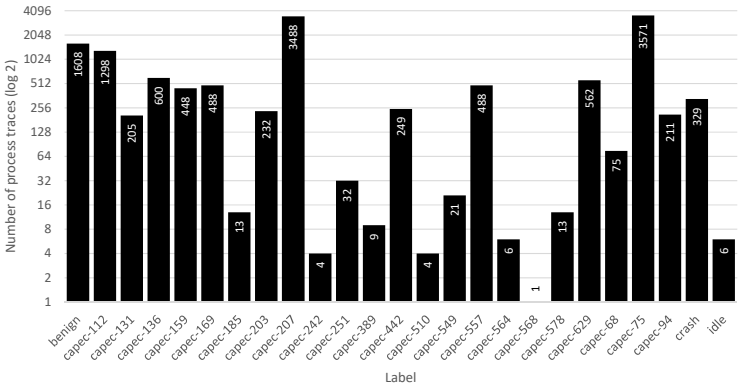


Figure 11: Number of process traces associated to a specific (CAPEC) class. The majority of data was labeled in accordance to its observed behavior, not malware family affiliation.

5.2. Code Implementation

The CSV-formatted graphs as well as the smart traces used by the sentiment component are preprocessed and converted into matrices using Bash and Python scripts. LLR-based sentiment analysis is implemented in R [57]. The optional grammar inference component of AIDIS is based on our own SEQUIN tool [42], which utilizes parts of a Java Sequitur implementation by Eibe Frank[7]. The Hungarian distance (Kuhn-Munkres), which is the basis for all graph distance computations, is determined using the `solve_LSAP` function[8] available in R. Knowledge dissemination and the answering of competency questions is currently done via Linux on-board tools (see output in Section 4.5.2). For prototype-based template generation, we utilize a local Malheur [59] installation configured to accept non-MIST [70] input data. Similarity hashing is based on a Python tool coded by Chris McCormack[9]. Decision trees are computed in R using the `randomForest` function[10], while our SVM implementation utilizes `svm_Linear` and `svm_Linear_Grid`. The mapping of resulting anomaly reports and scores to the PenQuest [43, 44] model is currently done manually.

5.3. Results

5.3.1. Relevant Process Identification

The identification of relevant processes happens in the optional 'sentiment analysis' stage of the AIDIS system (see Figure 5). If omitted, the selection of

[7] https://github.com/craignm/sequitur/tree/master/java
[8] https://www.rdocumentation.org/packages/clue/versions/0.3-55/topics/solve_LSAP
[9] https://github.com/chrisjmccormick/MinHash
[10] https://www.rdocumentation.org/packages/randomForest/versions/4.6-14/topics/randomForest

processes to be considered during anomaly detection and classification has to be conducted manually.

The data investigated contained close to 2,000 unique processes (`svchost.exe` being one of them) that were launched during the lifetime of more than 10,000 benign and close to 2,000 malicious system sessions. In our benign testbed environment alone, 1,260 unique processes were observed across all workstations. Upon infection with APT malware, the test machine interacted with a total of 876 distinctive processes. In both cases, processes with identical names but deviating directory locations were counted individually, as malware often reuses common application names to evade detection.

The reason for the discrepancy between total events and the lower process counts lies in the nature of our data: Each process can trigger an arbitrary number of events during its lifetime, which ranges from minutes to weeks, depending on the frequency of system reboots. This is markedly pronounced for kernel processes that run for as long as the OS is active. Ultimately, we used only a fraction of the available data for anomaly detection to demonstrate AIDIS' feasibility in a single-process scenario. To get there, such a process had to be identified:

As a first step, all processes that occurred in only the benign or malicious domains were discarded, leaving a total of 120 executed binaries that are possibly ubiquitous. The reason for this is that with AIDIS, we want to reduce the reliance on any prior knowledge about the name or location of malicious software, focusing our anomaly detection efforts on omnipresent processes that will exist no matter the current state of compromise. Additionally, we argue that not all APT attacks will utilize dropped binaries that can be seen in the operating system; especially when campaigns involve manual intrusion activity or techniques such as process injection. Full paths were ignored during relevant process identification in order to make fake system services comparable to their genuine counterparts.

In step 2, we used LLR sentiment scoring [40] to determine the likelihood of a process occurring as part of an event bigram exhibiting malicious tendencies. With the threshold set to $s_t = -0.05$ (see Section 4.3 for more information) and a tolerance bandwidth of 0.1, we extracted all processes with a mean sentiment rating of $s \leq 0.05$. This resulted in a reduced list of 84 processes. Next, we eliminated likely false positives that stem from the technical differences of physical and virtual machines, such as graphics drivers or first-run wizards that were initialized every time the virtual environment was started. With 72 processes left, it became apparent that user apps like word processors and programming environments need to be considered separately. While there is malware in the wild that utilizes such programs or their associated file types as injection targets or carrier medium, these attack scenarios were deemed too specific for the initial evaluation. Any baseline created for these processes would be highly dependent on user interaction which, in this case, might deviate significantly from benign instance to benign instance. At the same time, we hypothesize that even a compromised user application will eventually utilize a kernel process to perform a portion of its adversarial task – something that was corroborated by our tests.

Process (.exe)	Description	$\wedge$	$\overline{x}$	$\widetilde{x}$	$\vee$	$\sigma_{\overline{x}}$	k Events
conhost	Console window host process	-0.895	0.016	0.008	0.607	0.104	3,691.0
csrss	Win32 user-mode subsystem	-1.000	0.039	0.044	0.836	0.171	1,000.7
explorer	Explorer shell and file manager	-1.000	0.003	0.003	1.000	0.008	29,227.0
searchindexer	Windows search and indexing	-1.000	0.002	0.007	1.000	0.047	4,346.3
smss	Session manager subsystem	-1.000	-0.236	-0.332	1.000	0.416	6.3
svchost	Generic host process	-1.000	0.005	0.005	1.000	0.012	126,120.1
taskhost	Generic host process for libraries	-1.000	0.022	0.011	0.983	0.048	1,871.7

Table 8: Shortlist of relevant processes identified through sentiment analysis. While `smss.exe` produced the most malign numbers, the amount of process data for a meaningful follow-up analysis was simply too small. We opted for the generic host process as something of a 'semantic worst case' with the highest amount of data (number of events in the database) available.

With 57 processes remaining, we matched the OS-centric shortlist against two lists[11] [12] of ubiquitous kernel processes found in modern versions of Windows. This yielded a match for 7 processes that were determined to be relevant by the sentiment analysis component. In order to perform meaningful data mining, the process with the most recorded events (126.1 million) was chosen for all subsequent analyses: `svchost.exe`, the Windows generic host (service host) process [60].

See Table 8 for details about the 7 primary investigation candidates. Our results highlight that any event-based anomaly detection system might benefit from focusing on one or several of these processes.

Discussion – The multi-purpose `svchost.exe` process is involved in many operating system tasks and has existed since Windows 2000 [60], making it ideal for in-depth observation. At the same time, it arguably represents a worst case scenario for anomaly detection systems, as it is very difficult to create a behavioral baseline for such a versatile application. We argue that it is unlikely to find a Windows process that will produce more questioning results in an evaluation scenario, underlying the efficacy of AIDIS in adverse situations. At the same time, it needs to be emphasized that future iterations of the system will include additional processes for reasons of diversification and accuracy, starting with the remaining 6 processes determined to be relevant by LLR. Additional applications such as user programs will be considered in corresponding scenarios. For example, malware delivery (deception) detection as per the APT kill chain is more likely to focus on processes like `winword.exe` and `acrord32.exe` while many installation, persistence, and launch tasks can be captured by observing `cmd.exe`, `regedit.exe`, or `net.exe`. Our initial evaluation focused on the

[11] https://social.technet.microsoft.com/wiki/contents/articles/4485.windows-7-default-system-processes.aspx

[12] https://www.andreafortuna.org/dfir/forensics/standard-windows-processes-a-brief-reference/

generic host process because of its versatility but also due to data availability and time constraints. Refer to Section 5.3.4 for a closer look on computational performance.

5.3.2. SEQUIN Compression

Data compression is part of the 'grammar inference' component of AIDIS (see Figure 5). Like process identification, this stage is optional.

The standalone version of the SEQUIN component is evaluated in detail in [42]. In summary, event trace compression resulted in a 97.2% reduction of data for the 51.3 million events that comprise the investigated (benign) `svchost.exe` traces. However, with 47.6% processing time reduction, the average speed-up of the star anomaly detection process was less than we have seen for smaller corpora, where the average speed increase for AIDIS-type data was around 73%.

Discussion – There are two factors that limit the use of SEQUIN as recommended stage in the AIDIS process: Firstly, memory consumption is significant for corpora above a certain size. Initial experimentation had us reach a 64 GiB ceiling at around 6 million events [42]. While the more extensive experiment in the context of AIDIS consumed only 120 GiB RAM as opposed to the expected > 500 GiB determined by earlier linear regression [42], the memory demands on the analysis system remains a limiting factor.

Secondly, the effective reduction of data may prevent the creation of star anomaly templates, as the amount of events remaining to compute meaningful prototypes is simply too small. In our specific case, both Malheur and MinHash failed to produce templates because of insufficient data.

This points to the conclusion that SEQUIN, while very effective in compressing star graph data, is better suited as visualization-assisted knowledge extraction system tasked with analyzing event data that cannot be accurately classified by AIDIS. By highlighting deviating events, SEQUIN helps to spot anomalies without relying on supervised learning. Because of the specific approach used by Sequitur [53], SEQUIN has proven to be best suited for processes that are less eclectic than e.g. `svchost.exe` in their default behavior, such as driver software, error handlers, and on-demand applications for specific user or system tasks.

In future research, we will also investigate SEQUIN's suitability as an additional feature in the anomaly classification process itself. Furthermore, the component's ability to extract common sequences warrants investigation into its suitability as alternative to the 'perfect match' approach to template generation. In the meantime, SEQUIN will be used to aid in analyzing outliers and non-ubiquitous processes.

5.3.3. Star Anomaly Detection

The anomaly detection process based on extracted star structures fulfills two major purposes: Firstly, it scores unknown process traces against one or several templates for a numeric anomaly score, and secondly, it provides a human readable report explaining the deviation from a baseline graph. These reports

Mode	Templ.	Thresh.	TP	TN	FP	FN	Accuracy
Single	1	$\overline{m}$	93.86%	37.04%	62.96%	6.14%	89.34%
Single	1	$\overline{m_o}$	99.99%	29.56%	70.44%	0.01%	94.38%
Multi	n	$\overline{m}$	93.86%	75.68%	24.32%	6.14%	92.42%
Multi	n	$\overline{m_o}$	99.99%	52.76%	47.24%	0.01%	96.23%

Table 9: Accuracy of the star anomaly detection component in standalone mode, for both single and multiple ($n = 17$) templates. We use 'majority' mode and 'prototype' plus 'similarity hashing' mode (reduction), respectively. While optimizing the threshold increases overall accuracy for the dataset, a more balanced approach to reducing the false positive rate is recommended (multi $\overline{m}$). Note that this AIDIS component is not typically used without subsequent classification, which boosts accuracy significantly.

are then used in the anomaly classification component discussed in the next Section (5.3.4). In the following, we evaluate anomaly detection accuracy for the process `svchost.exe` using a single template produced by the 'majority' mode approach as well as a multi-template experiment utilizing 'prototype' mode followed by further reduction through similarity hashing (see Table 5 for an overview of graph template creation methods). Instead of analyzing traces in their entirety, we focus on the initial startup behavior, namely the first 10 seconds of execution of each process instance. In all cases, half of the available data was used for validation.

Results show that, while a single template is well suited for processes that exhibit stable behavior, it is difficult to accurately classify a versatile process like `svchost.exe` as benign or malicious with just one baseline to compare to. We ultimately achieved an accuracy of 89.34% using the mean benign score $\overline{m}$ as threshold separating the two classes. The optimum threshold was determined to be $\overline{m_o} = \frac{\overline{m}}{3}$. Using this value increased accuracy to 94.38%. Template generation took a negligible amount of time for each of the 13,961 malicious and 2,202 benign process instances, while matching required an average of 51 seconds per trace. See Table 9 for detailed results.

In a second experiment we automatically created a number of templates using the Malheur prototype approach, which resulted in 186 baseline traces computed in around 30 hours. To reduce this number to more practical dimensions we used similarity hashing in 'reduction' mode, bringing this number down to 17 within a few seconds. Using similarity hashing without prior heuristic clustering did not prove feasible: With a processing time of 47 hours and a resulting 589 templates, it was less computationally effective and yielded too high a number of templates for practical use.

To evaluate the multi-template approach, we considered the same dataset as before. If any of the 17 benign templates deemed a trace as within tolerance, the process run was classified as non-malicious. This increased the overall accuracy to 92.42% when using the arithmetic mean of benign scores $\overline{m}$ as threshold, while $\overline{m_o}$ accuracy was boosted to 96.23%. The false positive rate was significantly reduced from 63% (70.4%) to 24.3% or 47.2%, respectively. For detailed results, see Table 9.

Classifier	Classes	OOB error	Accuracy	Kappa	C-value	Time (s)
RF	2	0.26%	99.77%	-	-	142.1
RF	n	4.96%	91.37%	-	-	224.8
SVM	2	-	99.82%	99.24%	1	70.4
SVM grid	2	-	99.83%	99.28%	0.25	1899.8
SVM	n	-	95.53%	94.67%	1	412.0
SVM grid	n	-	95.73%	94.87%	1.75	8180.4

Table 10: Classification accuracy ($n = 25$) of the RF and SVM approach. Support vector machines generally proved to be more accurate in our scenario. For Random Forest, we tried 1000 trees with 100 variables on each split. For SVM, we used 10-fold cross validation with 3 repeats to reduce overfitting.

Discussion – Above results show the 'worst-case' accuracy of the star anomaly detection process when used as an individual system. Despite the better overall numbers, we recommend the mean or median benign score from the training set as threshold between the benign and malicious classes, as it more drastically reduces the false positive rate.

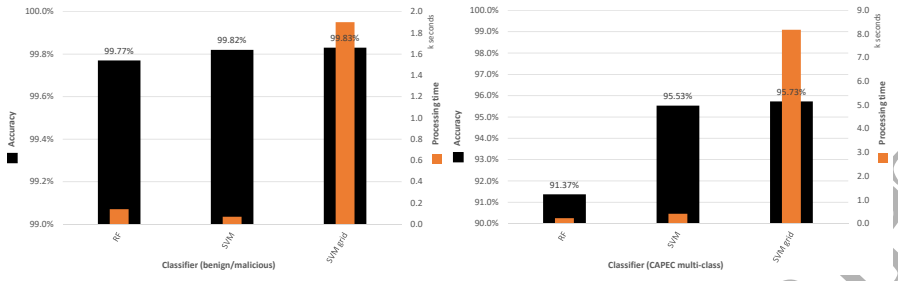
The key outcome of this evaluation stage isn't stand-alone component accuracy, but the degree of *process coverage*: 88.48% of all malicious activity created events attributed to `svchost.exe`. Considering single-template matching using $\overline{m}$, this resulted in a 83.05% accuracy in attack detection when observing only the generic host process for a total of 10 seconds. Combined with subsequent anomaly classification (see Section 5.3.4 below), this number was pushed to 88.31%. This finding is especially promising as it might help eliminate the need to incipiently identify malware binaries and to observe more that a few key OS processes such as the ones identified in Table 8) for system-wide attack detection.

In conclusion, it stands to mention that threshold-based anomaly detection it not AIDIS' core purpose. While the results are workable, the overall false positive rate is still too high to trust this purely number-based decision. As initially argued, considering event semantics is key to improving detection rates and eventual interpretation, which is why the actual classification of anomalies identified in this stage is considered the system's main component, discussed hereafter.

5.3.4. Star Anomaly Classification

With the previous star anomaly detection stage providing a purely threshold-based score, anomaly classification takes the resulting reports (as shown in Section 4.5.2) and asks a number of competency questions, the answers of which are used as classification features. We tested two technical approaches to classification: Random Forest, and linear support vector machines with and without variable C-score (hyperplane optimization).

In the first experiment, we used previously generated single-template anomaly reports from the anomaly detection stage and classified the data into a benign and malicious category: Both methods returned near-perfect results,



(a) Binary classification accuracy (b) CAPEC classification accuracy

Figure 12: Classification accuracy and processing times overview for binary and multi-class RF/SVM. The use of hyperplane optimization through the alteration of the C-value slightly improved the results, but significantly increased processing times, making common linear kernel SVMs the most sensible choice for our dataset.

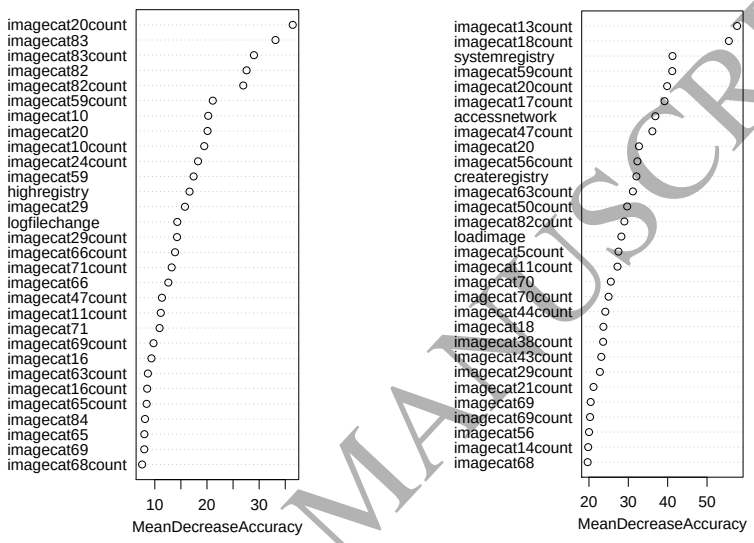
with a ROC accuracy of 99.77% for RF and 99.82% accuracy for SVM (see Table 10 and Figure 12a for details).

Finally, we repeated the process with the labeled data of the aforementioned 13,961 malicious and 2,202 benign process instances in order to test classification into 22+3 CAPEC-determined behavior categories. Linear kernel SVM with a C-value of 1.75 achieved the best result with an accuracy of 95.73%, closely followed by a constant-C ($C = 1$) SVM with 95.53% and Random Forest with 91.37% multi-ROC accuracy and an out-of-bag (OOB) error rate of 4.96%. See Table 10 and Figure 12b for a detailed overview of classification accuracy as well as processing times. Table 11 shows the class confusion matrix of the most accurate approach.

Discussion – Above results show the advantage of semantics-enabled classification over purely threshold-based approaches. Even with 25 classes, the accuracy was much higher than with the default ($s_t = \overline{m}$) benign/malicious distinction used in the previous stage. There is still room for improvement, however. The main source of misclassification were CAPEC patterns 75 [13] ('Manipulating Writeable Configuration Files') and 207 [14] ('Removing Important Client Functionality'). Here, between 4 and 8% of the associated traces were misclassified as the respective other. The reason can be found in the ambiguous nature of the pattern as well as the difficulty of clearly labeling data as one or the other: The removal of client functionality (e.g. firewall or user authentication measures) is often done by manipulating configuration files, leading to similar star graph elements and by extension, anomalous events.

Key features as per mean decrease in accuracy/Gini turned out to be the count of error function libraries imported, the use of Windows user management

[13] https://capec.mitre.org/data/definitions/75.html
[14] https://capec.mitre.org/data/definitions/207.html



(a) Benign/malicious classification (b) CAPEC category classification

Figure 13: Overview of features linked to the answer of a corresponding competency question. Features prefixed by 'imagecat' correspond to observed image load ('loadimage') activity (or the count thereof) in one of the 87 Windows library (DLL) categories parsed from various Microsoft and developer sources as part of the initial project stages. 'createregistry' and 'high-registry' determine the existence of anomalous operations that insert data into the Windows registry and the presence of a large number (> 35) of create/change/delete operations in general. 'systemregistry' is one of the few fixed pattern questions that are set to 'true' when keys within the `HKLM\System` registry hive are interacted with. 'logfilechange' does the same when `*.log` or `*.evt(x)` files are modified. If the parsed anomaly report contains network interaction, the 'accessnetwork' feature value is set to 'true'.
Image categories in the top 10 features: COM (10), Data Access (13), DHCP/DNS (17), Diagnostics (18), Error Handling (20), File System (24), Remote Desktop (56), Security (59), Windows User Management (82), Windows Universal App (83). A full list of categories and libraries within is available on request.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
1	1148	0	0	0	0	0	0	0	0	0	0	0	0	4	1	0	0	0	0	0	0	0	0	0	0
2	0	380	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	70	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	1	0	0	150	0	0	0	0	3	0	0	0	0	0	0	9	0	0	0	0	0	8	0	0	0
5	0	0	0	0	136	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
6	0	0	0	1	1	130	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	2	0	0	0
7	0	0	0	0	0	0	2	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0
8	0	0	0	0	0	0	0	62	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
9	1	0	0	1	0	1	0	3	1032	0	0	1	0	0	0	3	0	0	0	0	0	78	0	0	0
10	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	3	1	0	0	0	0	0	0	0	0	0	0	1	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	81	0	0	0	1	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	2	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	128	0	0	0	0	0	2	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	3	0	0	0	0	1	0
20	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	154	0	2	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	24	0	0	0	0
22	0	0	0	1	0	2	1	1	38	1	1	0	0	0	0	1	0	1	0	5	0	977	0	0	1
23	0	0	0	2	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	2	60	0	0
24	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	3	1	94	3
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 11: Confusion matrix for SVM with linear kernel, C-value of 1.75. The validation resulted in a 95.73% accuracy and a Kappa statistic (comparing observed to expected accuracy) of 94.87%. Classes with the highest misclassification rate were 9 (CAPEC-75) and 22 (CAPEC-207).

and universal app functions, high (system) registry interaction, operations related to log files, network activity in general, as well as the import of data access and diagnostics functions. See Figure 13 for a list and explanation of the most impactful features.

Summed up, many of the most relevant features are related to image load and registry operations. The reason for this can be found to a degree in the selection of data used in the experiment: With a focus on the initial 10 seconds of activity, it is expected to see numerous events pertaining to the dynamic linking of libraries [16], which is generally more widely used than static or runtime linking in both malware and benign software. The concept of dynamic linking requires applications to search and load library (DLL) resources at launch, resulting in a spike of corresponding 'image load' events. Registry events typically represent initialization tasks or changes to certain settings, something that is often seen in the early stages of operation as well. Interestingly, file events were found to be generally underrepresented during the start-up of compromised process instances in particular: Only 108 events in the selection described malicious file operations, as opposed to 40,538 events in the benign `svchost.exe` corpus. As a result, the lack of specific file operations might be a strong indicator of manipulation – something that has to be considered in future feature selection.

Currently, only the general existence of such events is assessed.

While the importance of image and registry operations constitutes an interesting finding in itself, upcoming experiments will increasingly focus on file and network events typically triggered during a process's entire lifetime. This includes defining new features corresponding to questions about the nature of any created or modified files, as well as to the fact that compromised system processes fail to display a level of file interaction commonly seen in their benign cousins.

5.3.5. Model Mapping

Armed with the automated classification of anomalies as belonging to a specific CAPEC pattern, it now becomes possible to link our data with the PenQuest meta model [43, 44] for further semantic enrichment, interpretation, and mitigation planning.

For our evaluation, we use the class with the most events while boasting a low misclassification rate. Disqualifying classes 9 and 22 (see Table 11), we take a look at class 2 (CAPEC-112 [15], 'Brute Force'), with a total number of 380 process anomaly reports and a misclassification rate of 0%.

For data mapping, we use PenQuest's $\langle Event \langle Type = Anomaly \rangle \rangle$ notation of X, as specified in Section 3.3 and [43, 44]. With a time range of $\langle Time \langle Start = 0, End = 10 \rangle \rangle$, each $\langle Operation \rangle + \langle Argument \rangle$ pair with $\langle Parent = svchost.exe \rangle$ will be appended in sequence, resulting in a simple description of X that can be easily converted to other threat definition languages or shared directly with others.

The link to CAPEC provides us with additional semantic information, namely that 'Brute Force' refers to activity where the "attacker attempts to gain access to this asset by using trial-and-error to exhaustively explore all the possible secret values in the hope of finding the secret (or a value that is functionally equivalent) that will unlock the asset."[48]

According to the meta model and Figure 1, CAPEC-112 can be an APT kill chain 'Delivery–Intrusion' as well as an 'Installation–Propagation' or 'Installation–Persistence' support action which is typically used in combination with other attack activity. Categorized as the identically named 'BF' (Brute Force) attack action, PenQuest also defines appropriate primary controls countering the threat, namely 'Authentication protection' (AP): This controls group is primarily concerned with managing authenticators such as passwords, tokens, and biometric information. Associated defense actions include the categories 'Remote Access' (REA) and 'Authenticator Management' (AUM), with a range of controls directly out of NIST SP 800-53 [27]. Specific countermeasures therefore include remote access control and encryption, access point management, password/PKI/hardware/biometric authentication, as well as controls related to cache expiration settings.

[15] https://capec.mitre.org/data/definitions/112.html

The information gleaned from the model can now be used to plan appropriate defensive measures to prevent this particular attack. PenQuest's gamified nature also allows us to play through the attack and test various controls and systems that may reduce threat impact and probability. While the efficacy of the suggested countermeasures need to be evaluated on a case-by-case basis using real world infrastructure, 9 interviewed security practitioners of intermediate, professional, and expert level strongly (3 points) or rather agree (2 points) that the model is 'applicable to real world scenarios', resulting in 21 out of 27 possible points. Almost all testers strongly agree that using PenQuest increases general security awareness when used (22/27 points). Refer to [44] for more information on the model's in-depth evaluation.

Discussion – It stands to reason that this final mapping stage of AIDIS is difficult to quantitatively evaluate. Future work will investigate crafted attack scenarios that are then mitigated by specific defense measures suggested by the respective NIST categories in order to determine the effectiveness of the controls. Furthermore, we will design a full simulation component of the gamified model using reinforcement learning to automate the process of 'playing through' a large number of attacks for strategy optimization purposes.

On the modeling side, not all of the 517 CAPEC classes and only a portion of the defensive controls are currently part of PenQuest. Around 12% of the available patterns have been used to populate the model to date, whereas 'detailed' technical patterns referring to specific software attacks (as opposed to 'meta' and 'standard') are not currently included. Control-wise, we prototypically implemented 70 out of 224 controls specified by NIST. With the model itself ready for use, the remainder of the data can be added at will. However, it needs to be stated that the CAPEC repository itself is missing some of the required information needed for successful automated mapping, which increases the effort required to add the remaining patterns. For future iterations, we will therefore consider alternative vocabularies such as MITRE ATT&CK[16].

5.4. Comparison

In this final evaluation section we take a look at 3 systems that share technical aspects with AIDIS' core components. Please note that it is generally difficult to compare our approach to any alternative solutions, since the data basis used for training and validation is not the same for reasons of both availability and compatibility. Furthermore, none of the identified works provide semantic enrichment through a model such as PenQuest.

In the following, we pit AIDIS against a general intrusion detection system based on similar SVM multi-class classification [1], as well as against two graph-based threat detection systems [3, 26] using binary classification. Refer to the 'Related Work' section for additional information about the discussed works. Figure 14 provides an overview of the respective accuracy scores.

[16] https://attack.mitre.org/

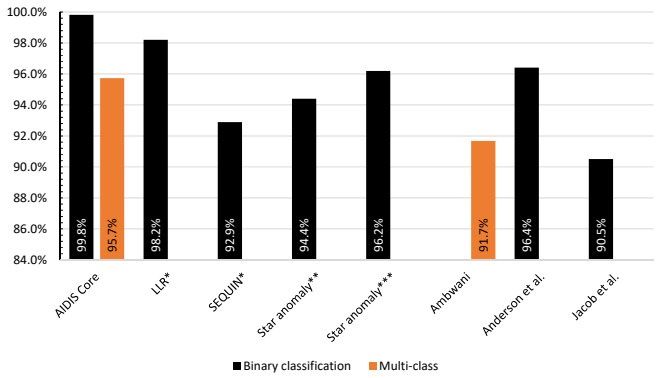


Figure 14: Classification accuracy of AIDIS components (both core and stand-alone [40, 42]) compared to three similar systems [1, 3, 26].

5.4.1. SVM Multi-Class Classification for Network Traffic

With its SVM-based multi-class classification based on the KDD99 dataset [72], Ambwani [1] presents a network-based intrusion detection system attempting to distinguish 4 different attack categories: denial of service (DoS), probing, remote to local (R2L), and user-to-root (U2R). With a total of 23 inherent attacks considered, the classification problem is comparable to AIDIS in scope. Its focus on full traffic dumps makes Ambwani [1]'s solution a direct network equivalent to our host-based approach, which uses both SVM and RF in its classification of predetermined anomalies.

The authors observed an optimized 91.67% detection accuracy when discriminating 23 classes. AIDIS achieves 95.73%, which marks a significant improvement over a purely network-traffic based system. While this does not make Ambwani [1]'s approach obsolete in any way, it gives a strong indication that assessing endpoint events is at least as feasible as using traffic dumps when it comes to ML-powered intrusion detection. Coupled with AIDIS' anomaly detection stages, our system additionally provides means to spot unknown behavior not covered by labeled datasets.

5.4.2. Graph-based Attack Detection

Since AIDIS utilizes a graph-based approach, we compare it to two such systems in our evaluation. Previously outlined in Section 2, the work by Anderson et al. [3] encompasses a detection algorithm based on the analysis of graphs constructed from dynamically collected instruction traces. The utilized SVM classifier is tested in a binary scenario distinguishing benign from malicious software. With a combined kernel (best-case) accuracy of 96.41% compared to AIDIS' 99.82%, the solution fares notably worse. In terms of performance, however, our system is at a disadvantage, as it requires an additional ∼50 seconds per trace instance to compute the anomaly graphs needed for classification.

Jackstraws [26] offers host-side C2 traffic identification through dynamic

analysis of individual malware samples. Not unlike AIDIS, it models data flows between API calls as graph. The resulting patterns are used as templates for subsequent detection. This is also the key difference to our solution: Under the hood, Jackstraws extracts graph templates not as baseline for anomaly detection, but uses them as de-facto signatures. This results in a high number (75%) of unclassified network connections. The remainder was categorized with workable accuracy, but still scores over 9 percentage points lower than AIDIS core in a binary classification setting.

6. Limitations and Future Work

In this section, we want to highlight current limitations of the AIDIS system and outline planned future work. We discuss three main areas: Automation, performance, and accuracy.

Regarding *automation*, our approach currently relies on the manual definition of competency questions that provide the features for RF/SVM classification. Especially thresholds (e.g. what constitutes a 'high number' of file or registry events) stem from software analysis experience rather than statistical evaluation. Here, future work will improve and automate both the creation of the competency questions as well as the process of parsing anomaly reports, which is currently realized through conventional text processing scripts. Another area that would benefit from additional automation is the conversion and mapping of classified anomalies to the PenQuest notation. This will help to ultimately map monitoring data to an ontology describing the meta model, which is currently a work in progress based on earlier efforts such as TAON [39]. Future research will also include the automated simulation of attack scenarios aimed at discovering optimal defense strategies through reinforcement learning.

A second aspect in need of improvement is *performance*. Currently, the creation of graph templates and their matching to target star structures is implemented as an early prototype that does not significantly optimize these computationally complex operations. Future iterations of AIDIS will therefore focus particularly on runtime reduction to open the door for close to real-time applications. This will also include optional stages such as sentiment analysis and Sequitur compression, where the resolving of rules is responsible for much of its processing overhead [42].

Lastly, there is the matter of *accuracy*. While the general results are promising, star structure anomaly detection by itself is in need of further fine-tuning to bring down false positive rates for multifaceted processes such as the investigated Windows host process. However, acknowledging that threshold-based systems are unlikely to achieve the same level of accuracy as semantic approaches, most effort will be invested into improving star structure classification for multiple categories. Firstly, we will investigate potentially less ambiguous and more complete vocabularies than CAPEC to reduce the risk of misclassification. Secondly, we will develop new competency questions utilizing the insight gained from evaluating AIDIS – especially the determined decrease of accuracy for various question types (Figure 13) and the observed peculiarities of (malicious) file

events. This will help replace some of the less relevant questions with more expressive ones. Future versions of AIDIS will also see the inclusion of other detection system scores/results as additional classification features, which is expected to further improve accuracy.

7. Conclusion

We presented the components of AIDIS, a star structure-based "advanced anomaly detection and interpretation system' able to detect and explain anomalous deviations in operating system process behavior. The returned output of detailed state changes as well as a tendency towards a specific APT stage and attack pattern is expressed through the mapping of semantic key factors to a dedicated attacker–defender model. At the same time, the model suggests specific measures intended to counter any observed attack.

The process was prototypically implemented and successfully tested using real-world process data captured on more than a dozen company workstations. Ultimately, 99.8% of all star structure anomalies were correctly identified as benign or malicious, with a solid 95.7% accuracy in multi-class scenarios that seek to associate each anomaly with a distinct CAPEC attack pattern. Furthermore, we have shown that 88.3% of close to 2,000 attacks could be accurately identified by observing and classifying just one generic Windows process for a mere 10 seconds, thereby eliminating the necessity to monitor each and every (unknown) process existing on a system.

Further research will be conducted into the automation of the model mapping process and the means to simulate and assess advanced attacks at scale. Ultimately, an anomaly detection and explication system based on the AIDIS approach will offer invaluable aid to malware analysts and security operators alike.

Acknowledgements

The financial support by the Austrian Federal Ministry of Science, Research and Economy and the National Foundation for Research, Technology and Development is gratefully acknowledged.

References

References

[1] T. Ambwani. Multi class support vector machine implementation to intrusion detection. In *Proceedings of the International Joint Conference on Neural Networks, 2003.*, volume 3, pages 2300–2305 vol.3, July 2003. doi: 10.1109/IJCNN.2003.1223770.

[2] Theodoros Anagnostopoulos, Christos Anagnostopoulos, and Stathes Hadjiefthymiades. Enabling attack behavior prediction in ubiquitous environments. In *Pervasive Services, 2005. ICPS'05. Proc.. Int. Conference on*, pages 425–428. IEEE, 2005.

[3] Blake Anderson, Daniel Quist, Joshua Neil, Curtis Storlie, and Terran Lane. Graph-based malware detection using dynamic analysis. *Journal in computer Virology*, 7(4):247–258, 2011.

[4] Sean Barnum. Standardizing cyber threat intelligence information with the Structured Threat Information eXpression (STIX^{TM}). *MITRE Corporation*, 11:1–22, 2012.

[5] Andrei Z Broder. On the resemblance and containment of documents. In *Compression and Complexity of Sequences 1997. Proceedings*, pages 21–29. IEEE, 1997.

[6] Sergio Caltagirone, Andrew Pendergast, and Christopher Betz. The diamond model of intrusion analysis. Technical report, Center for Cyber Intelligence Analysis and Threat Research, Hanover, 2013.

[7] Eric Chien, Liam OMurchu, and Nicolas Falliere. W32. Duqu: the precursor to the next Stuxnet. In *Proc. of the 5th USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, 2012.

[8] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[9] Santanu Das, Bryan L Matthews, Ashok N Srivastava, and Nikunj C Oza. Multiple kernel learning for heterogeneous anomaly detection: algorithm and aviation safety case study. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 47–56. ACM, 2010.

[10] Jelle De Vries, Hans Hoogstraaten, Jan van den Berg, and Semir Daskapan. Systems for Detecting Advanced Persistent Threats: A Development Roadmap Using Intelligent Data Analysis. In *Intl. Conference on Cyber Security*, pages 54–61. IEEE, 2012.

[11] Andrey Dolgikh, Tomas Nykodym, Victor Skormin, and Zachary Birnbaum. Using behavioral modeling and customized normalcy profiles as protection against targeted cyber-attacks. In *Computer Network Security*, pages 191–202. Springer, 2012.

[12] Ted Dunning. Accurate methods for the statistics of surprise and coincidence. *Computational linguistics*, pages 61–74, 1993.

[13] Kenneth S Edge, George C Dalton, Richard A Raines, and Robert F Mills. Using attack and protection trees to analyze threats and defenses to homeland security. In *Military Communications Conference, 2006. MILCOM 2006. IEEE*, pages 1–7. IEEE, 2006.

[14] Javier Esparza, Martin Leucker, and Maximilian Schlund. Learning workflow petri nets. In *International Conference on Applications and Theory of Petri Nets*, pages 206–225. Springer, 2010.

[15] Nicolas Falliere, Liam Murchu, and Eric Chien. W32.Stuxnet.Dossier. URL `https://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf`. Accessed 2015-09-18.

[16] M. Franz. Dynamic linking of software components. *Computer*, 30(3): 74–81, March 1997. ISSN 0018-9162. doi: 10.1109/2.573670.

[17] Thomas Freytag. Woped–workflow petri net designer. *University of Cooperative Education*, pages 279–282, 2005.

[18] Michael Gamon. Sentiment classification on customer feedback data: noisy data, large feature vectors, and the role of linguistic analysis. In *Proc. of the 20th international conference on Computational Linguistics*, page 841. Association for Computational Linguistics, 2004. URL `http://dl.acm.org/citation.cfm?id=1220476`.

[19] Chandan Gautam, Ramesh Balaji, K Sudharsan, Aruna Tiwari, and Kapil Ahuja. Localized multiple kernel learning for anomaly detection: One-class classification. *Knowledge-Based Systems*, 165:241–252, 2019.

[20] Paul Giura and Wei Wang. A context-based detection framework for advanced persistent threats. In *Cyber Security (CyberSecurity), 2012 Int. Conference on*, pages 69–74. IEEE, 2012.

[21] Adam Greenberg. Russians fingered for 'Uroburos' spy malware campaign, went undetected for years - SC Magazine. URL `http://www.scmagazine.com/russians-fingered-for-uroburos-spy-malware-campaign-went-undetected-for-years/article/336570/`. Accessed 2015-07-29.

[22] Liona Herman. Malware Attack at US Health Organization Went Undetected for 2 Years. URL `http://www.hackbusters.com/news/stories/187232-malware-attack-at-us-health-organization-went-undetected-for-2-years`. Accessed 2015-10-20.

[23] Xin Hu, Tzi-cker Chiueh, and Kang G Shin. Large-scale malware indexing using function-call graphs. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 611–620. ACM, 2009.

[24] Eric M. Hutchins, Michael J. Cloppert, and Rohan M. Amin. Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. *Leading Issues in Information Warfare & Security Research*, 1:80, 2011.

[25] Paul Jaccard. Étude comparative de la distribution florale dans une portion des alpes et des jura. *Bull Soc Vaudoise Sci Nat*, 37:547–579, 1901.

[26] Gregoire Jacob, Ralf Hund, Christopher Kruegel, and Thorsten Holz. Jackstraws: Picking command and control connections from bot traffic. In *USENIX Security Symposium*, volume 2011. San Francisco, CA, USA, 2011.

[27] Joint Task Force Transformation Initiative. SP 800-53 rev. 4. Recommended Security Controls for Federal Information Systems and Organizations. Technical report, Gaithersburg, MD, United States, 2015.

[28] Aivo Jürgenson and Jan Willemson. Serial model for attack tree computations. In *International Conference on Information Security and Cryptology*, pages 118–128. Springer, 2009.

[29] Kaspersky Lab. Duqu: Steal Everything. URL `http://www.kaspersky.com/about/press/major_malware_outbreaks/duqu`. Accessed 2015-07-29.

[30] Kaspersky Lab. What is Flame Malware | Definition and Risks | Kaspersky Lab, 2012. URL `http://www.kaspersky.com/flame`. Accessed 2015-07-29.

[31] Kaspersky Lab's Global Research & Analysis Team. Gauss: Abnormal Distribution - Securelist. URL `https://securelist.com/analysis/36620/gauss-abnormal-distribution/`. Accessed 2015-07-29.

[32] Shawn Knight. Sophisticated malware dubbed 'The Mask' went undetected for the past seven years - TechSpot. URL `http://www.techspot.com/news/55640-sophisticated-malware-dubbed-the-mask-went-undetected-for-the-past-seven-years.html`. Accessed 2015-07-29.

[33] Barbara Kordy, Sjouke Mauw, Saša Radomirović, and Patrick Schweitzer. Attack–defense trees. *Journal of Logic and Computation*, 24(1):55–87, 2014.

[34] Christopher Kruegel, Richard Lippmann, and Andrew Clark, editors. *Recent advances in intrusion detection*. Number 4637 in Lecture notes in computer science. Springer-Verlag, Berlin ; New York, 2007. ISBN 978-3-540-74319-4.

[35] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[36] Harold William Kuhn. *Lectures on the Theory of Games*. Princeton University Press, 2009.

[37] Andy Liaw, Matthew Wiener, et al. Classification and regression by randomforest. *R news*, 2(3):18–22, 2002.

[38] Robert Luh, Stefan Marschalek, Manfred Kaiser, Helge Janicke, and Sebastian Schrittwieser. Semantics-aware detection of targeted attacks: a survey. *Journal of Computer Virology and Hacking Techniques*, pages 1–39, 2016.

[39] Robert Luh, Sebastian Schrittwieser, and Stefan Marschalek. TAON: An ontology-based approach to mitigating targeted attacks. In *Proc. of the 18th Int. Conference on Information Integration and Web-based Applications & Services*. ACM, 2016.

[40] Robert Luh, Sebastian Schrittwieser, and Stefan Marschalek. LLR-based sentiment analysis for kernel event sequences. In *Advanced Information Networking and Applications (AINA), 2017 IEEE 31st International Conference on*, pages 764–771. IEEE, 2017.

[41] Robert Luh, Sebastian Schrittwieser, Stefan Marschalek, Helge Janicke, and Edgar Weippl. Design of an anomaly-based threat detection & explication system. In *ICISSP*, pages 397–402, 2017.

[42] Robert Luh, Gregor Schramm, Markus Wagner, Helge Janicke, and Sebastian Schrittwieser. SEQUIN: a grammar inference framework for analyzing malicious system behavior. *Journal of Computer Virology and Hacking Techniques*, pages 1–21, 2018.

[43] Robert Luh, Marlies Temper, Simon Tjoa, and Sebastian Schrittwieser. APT RPG: Design of a gamified attacker/defender meta model. In *Proceedings of the 4th International Conference on Information Systems Security and Privacy (ICISSP 2018)*. SCITEPRESS, 2018.

[44] Robert Luh, Marlies Temper, Simon Tjoa, Sebastian Schrittwieser, and Helge Janicke. PenQuest: A gamified attacker/defender meta model for cyber security assessment and education. Preprint, 2018.

[45] Stefan Marschalek, Robert Luh, Manfred Kaiser, and Sebastian Schrittwieser. Classifying malicious system behavior using event propagation trees. In *Proc. of the 17th Int. Conference on Information Integration and Web-based Applications & Services*. Association for Computational Linguistics, 2015.

[46] Elinor Mills. A who's who of Mideast-targeted malware. URL `http://www.cnet.com/news/a-whos-who-of-mideast-targeted-malware/`. Accessed 2015-09-18.

[47] Jelena Mirkovic and Peter Reiher. A taxonomy of ddos attack and ddos defense mechanisms. *ACM SIGCOMM Computer Communication Review*, 34(2):39–53, 2004.

[48] MITRE Corporation. CAPEC - Common Attack Pattern Enumeration and Classification (CAPEC), . URL `https://capec.mitre.org/`. Accessed 2015-09-22.

[49] MITRE Corporation. STIX - Structured Threat Information Expression | STIX Project Documentation, . URL `https://stixproject.github.io/`. Accessed 2015-09-22.

[50] Steve Morgan. 2017 Cybercrime Report. Technical report, Cybersecurity Ventures, 2017.

[51] Christopher Munsey. Economic Espionage: Competing For Trade By Stealing Industrial Secrets. URL `https://leb.fbi.gov/2013/october-november/economic-espionage-competing-for-trade-by-stealing-industrial-secrets`. Accessed 2015-09-15.

[52] Gerhard Münz and Georg Carle. Real-time analysis of flow data for network attack detection. In *Integrated Network Management, 2007. IM'07. 10th IFIP/IEEE Int. Symposium on*, pages 100–108. IEEE, 2007.

[53] Craig G. Nevill-Manning and Ian H. Witten. Identifying hierarchical strcture in sequences: A linear-time algorithm. *J. Artif. Intell. Res.(JAIR)*, 7: 67–82, 1997.

[54] Caleb C Noble and Diane J Cook. Graph-based anomaly detection. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 631–636. ACM, 2003.

[55] Cynthia Phillips and Laura Painton Swiler. A graph-based system for network-vulnerability analysis. In *Proceedings of the 1998 workshop on New security paradigms*, pages 71–79. ACM, 1998.

[56] Ludovic Pètre-Cambacédès and Marc Bouissou. Attack and defense modeling with bdmp. In Igor Kotenko and Victor Skormin, editors, *Computer Network Security*, pages 86–101, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. ISBN 978-3-642-14706-7.

[57] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2008. URL `http://www.R-project.org`. ISBN 3-900051-07-0.

[58] Anand Rajaraman and Jeffrey David Ullman. *Mining of massive datasets*. Cambridge University Press, 2011.

[59] Konrad Rieck, Philipp Trinius, Carsten Willems, and Thorsten Holz. Automatic analysis of malware behavior using machine learning. *J. Comput. Secur.*, 19(4):639–668, December 2011. ISSN 0926-227X.

[60] Mark E Russinovich, David A Solomon, and Alex Ionescu. *Windows internals*. Pearson Education, 2012.

[61] Bruce Schneier. Attack trees. *Dr. Dobb's journal*, 24(12):21–29, 1999.

[62] Bernhard Schölkopf, Robert C Williamson, Alex J Smola, John Shawe-Taylor, and John C Platt. Support vector method for novelty detection. In *Advances in neural information processing systems*, pages 582–588, 2000.

[63] Seculert. Mahdi - The Cyberwar Savior? URL `http://www.seculert.com/blog/2012/07/mahdi-cyberwar-savior.html`. Accessed 2015-07-29.

[64] Oleg Sheyner, Joshua Haines, Somesh Jha, Richard Lippmann, and Jeannette M Wing. Automated generation and analysis of attack graphs. In *Security and privacy, 2002. Proceedings. 2002 IEEE Symposium on*, pages 273–284. IEEE, 2002.

[65] Aditya K. Sood and Richard J. Enbody. Targeted cyberattacks: a superset of advanced persistent threats. *IEEE security & privacy*, (1):54–61, 2013.

[66] Gary Stoneburner, Alice Y. Goguen, and Alexis Feringa. SP 800-30. Risk Management Guide for Information Technology Systems. Technical report, 2002.

[67] Zareen Syed, Ankur Padia, Tim Finin, M Lisa Mathews, and Anupam Joshi. UCO: A unified cybersecurity ontology. 2016.

[68] Symantec. Regin: Top-tier espionage tool enables stealthy surveillance. URL `http://www.symantec.com/connect/blogs/regin-top-tier-espionage-tool-enables-stealthy-surveillance`. Accessed 2015-09-15.

[69] The Hacker News. Harkonnen Operation — Malware Campaign that Went Undetected for 12 Years. URL `http://thehackernews.com/2014/09/harkonnen-operation-malware-campaign_16.html`. Accessed 2015-07-29.

[70] Philipp Trinius, Carsten Willems, Thorsten Holz, and Konrad Rieck. A malware instruction set for behavior-based analysis. Technical report, University of Mannheim, 2009.

[71] Jeffrey Undercoffer, John Pinkston, Anupam Joshi, and Timothy Finin. A target-centric ontology for intrusion detection. In *18th International Joint Conference on Artificial Intelligence*, pages 9–15, 2004.

[72] University of California. KDD Cup 1999 Data. URL `http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`. Accessed 2015-07-29.

[73] Andrew Vance. Flow based analysis of Advanced Persistent Threats detecting targeted attacks in cloud computing. In *Infocommunications Science and Technology, 2014 First Int. Scientific-Practical Conference Problems of*, pages 173–176. IEEE, 2014.

[74] David Wagner and Paolo Soto. Mimicry attacks on host-based intrusion detection systems. In *Proc. of the 9th ACM Conference on Computer and Communications Security*, pages 255–264. ACM, 2002.

[75] Markus Wagner, Fabian Fischer, Robert Luh, Andrea Haberson, Alexander Rind, Daniel Keim, Wolfgang Aigner, Rita Borgo, Fabio Ganovelli, and Ivan Viola. A Survey of Visualization Systems for Malware Analysis. In *Eurographics Conference on Visualization (EuroVis) State of The Art Reports*, pages 105–125. EuroGraphics.

[76] Markus Wagner, Alexander Rind, Niklas Thür, and Wolfgang Aigner. A knowledge-assisted visual malware analysis system: Design, validation, and reflection of kamas. *Computers & Security*, 67:1–15, 02/2017 2017. ISSN 0167-4048. doi: 10.1016/j.cose.2017.02.003.

[77] C Wilson. Exterminating the RAT Part I: Dissecting dark comet campaigns, 2012.

[78] Xifeng Yan and Jiawei Han. gspan: Graph-based substructure pattern mining. In *Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on*, pages 721–724. IEEE, 2002.

[79] Liangwei Zhang, Jing Lin, and Ramin Karim. Adaptive kernel density-based anomaly detection for nonlinear systems. *Knowledge-Based Systems*, 139:50–63, 2018.

Robert Luh is researcher at the Josef Ressel Center for Unified Threat Intelligence on Targeted Attacks as well as at the Institute of IT Security Research at St. Pölten University of Applied Sciences. His research into targeted attacks includes threat modeling, adversary and malware behavior, intrusion detection, as well as machine learning. Robert is currently working towards his PhD at De Montfort University Leicester (DMU).

Helge Janicke is professor in computer science and Head of School of Computer Science and Informatics, as well as Head of the Cyber Technology Institute (CTI) at De Montfort University Leicester. He is mainly involved in research supervision and post-graduate teaching in Computer Security and Computer Forensic related subject areas. Helge's research interests are in the area of computer security, in particular access control and policy-based system management. He was awarded his PhD in 2007 from DMU.

Sebastian Schrittwieser is a permanent professor (FH) at St. Pölten University of Applied Sciences. From 2010 to 2014, he worked as a researcher and project manager at SBA Research, the Austrian competence center for information security. Sebastian was awarded his doctorate at TU Wien in 2014. Since 2015, Sebastian heads the Josef Ressel Center for Unified Threat Intelligence on Targeted Attacks, which explores novel techniques for detecting and mitigating targeted attacks on corporate IT infrastructures.