Czech Technical University in Prague
Faculty of Electrical Engineering

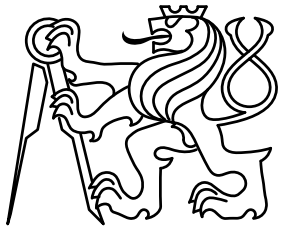
Doctoral Thesis

August 2017

Jan Stiborek

Czech Technical University in Prague

Faculty of Electrical Engineering
Department of Computer Science



Dynamic Reconfiguration of Intrusion Detection Systems

Doctoral Thesis

Jan Stiborek

Prague, August 2017

Ph.D. Programme: Electrical Engineering and Information Technology
Branch of study: Information Science and Computer Engineering

Supervisor: Ing. Tomáš Pevný, Ph.D.
Supervisor-Specialist: Martin Rehák, Ph.D., Ingénieur ECP

Acknowledgments

First and foremost, I would like to express my deep and sincere gratitude to my supervisors Tomáš Pevný and Martin Rehák who guided me in my research pointed me to right directions. My special thanks belong to prof. Michal Pěchouček who provided me the opportunity to joint Agent Technology Center almost ten years ago and who always offered help and advice.

I would like to thank my colleagues and coworkers for their support and inspiring discussion throughout my studies. Namely I would like to thank Karel Bartoš, Martin Grill and Ján Jusko whom I worked with closely for more than seven years. My special thanks belong to Petr Somol for his valuable comments to this thesis.

I gratefully acknowledge the support of funding sources that make my PhD work possible. I was funded by the ITC-A of the U.S Army (contracts N62558-07-C-001, W911NF-08-1-0250, W911NF-10-1-0070 and W911NF-12-1-0028); ONR Global under the Department of the Navy (contract N62909-11-1-7036); Air Force Office of Scientific Research, Air Force Material Command, USAF (grant FA8655-10-1-3016); Czech Ministry of Education (grants ME10051 and MEB111008) and the Czech Ministry of Interior (grants VG2VS/189 and VG20122014079).

Finally, I would like to thank my family, especially my wife Marie and son Kuba. I would not be able to finish this thesis without their support. Thank you.

Abstract

Intrusion detection systems (IDS) used in network security are complex solutions that require precise tuning prior to their deployment. Such tuning, however, is a problem. If done statically, the fixed configuration fails to follow the dynamic trends in the network traffic. On the other hand, configuration which is dynamically optimized using the complete traffic of the monitored network (*background traffic*) is infeasible due to the lack of ground-truth. To tackle these issues, researchers recently proposed to mix prerecorded static traces of labeled network traffic (i.e. *challenges*) into the background traffic, where they serve as evaluation data, and the IDS is dynamically adapted with respect to these challenges.

This thesis extends the challenge-based approach in two steps. In the first step, we adopt techniques from game theory to model the interactions between IDS (defender) and an attacker to make the adaptation process robust against the rational adversaries. We propose a dynamically-defined two-player single stage game with complex utility function to precisely capture incentives of both attacker and defender. Next, we combine the game definition with the challenge-based principle so we can estimate the parameters of the security game online, use traditional game-theoretical solution concept to solve the game, and immediately reconfigure the IDS accordingly. The experimental evaluation proves that this approach outperforms the trust-based baseline solution and thus allows us to improve the performance of the IDS against rational attacker.

However, using fixed database of static challenges for dynamic adaptation of the IDS is still far from optimal as it provides data with only limited variability, and manual updates of the database cannot provide new data fast enough as new trends and techniques used by malware authors emerge literally every day.

To solve these problems, we propose to replace legitimate challenges with dynamic simulation of network behavior based on probabilistic generative model. We experimentally verified that the proposed model generates network traffic similar to the traffic of real users. Next, we automate the updating the database of malicious challenges via emulation of malicious behavior with network traffic observed during execution of malware binaries in controlled environment (*sandbox*). In order to address the lack of labeled malware binaries, we propose novel approach for classification and clustering of unknown binaries based on their interactions with system resources (files, network traffic, mutexes, registry keys and error messages generated by the operating system). Moreover, the proposed model prioritizes the generated clusters to further aid the manual analysis of the threat level required in the definition of the security game. The performance of the classification and clustering of malware binaries is verified on large real-world dataset.

Abstrakt

Systémy pro detekci vniknutí do počítačových sítí (IDS systémy) představují komplexní řešení, která před svým operačním nasazením typicky vyžadují přesné nastavení všech parametrů. To ovšem představuje problém. Statická konfigurace totiž nerespektuje dynamické změny v síťovém provozu. Stejně tak použití dynamické konfigurace optimalizované na základě síťového provozu není možné kvůli jeho neznámé klasifikaci. Proto byl v poslední době navržen třetí postup. Ten je založený na principu vkládání předem oklasifikovaných statických příkladů síťového provozu (*challenges*), na jejichž základě se pak IDS systém dynamicky rekonfiguruje.

Tato práce rozšiřuje poslední zmíněný postup ve dvou směrech. Tím prvním je použití herně–teoretického přístupu, který modeluje interakce mezi obráncem (v našem případě IDS systém) a útočníkem. Navržené řešení spočívá v použití dynamicky definované jednokolové hry dvou hračů kombinované s komplexní účelovou funkcí, která detailně zachycuje záměry útočníka i obránce. Tato hra je poté propojena s principem vkládání příkladů síťového provozu, abychom mohli odhadnout její parametry a nalézt optimální řešení, podle něhož se poté IDS systém rekonfiguruje. Experimentální evaluace ukazuje, že navržený postup je schopen v porovnání s výchozím trust-based řešením lépe optimalizovat IDS systém.

V takovém případě však není pro dynamickou rekonfiguraci IDS systému optimální použití fixní databáze statických příkladů síťového provozu, a to kvůli její omezené variabilitě. To se týká jak provozu legitimního, tak také generovaného malwarem.

Druhou částí navrhovaného řešení je nahrazení fixní databáze. V případě legitimního provozu je nahrazena dynamickou simulací pomocí generativního modelu. Tady experimentální ověření ukazuje, že navržený model simuluje provoz podobný reálnému provozu. V případě provozu generovaného malwarem je databáze automaticky aktualizována pomocí provozu zachyceného v průběhu exekuce malwaru v kontrolovaném prostředí (*sandbox*). Vzhledem k tomu, že klasifikace jednotlivých aplikací obvykle není dostupná, navrhujeme klasifikovat a shlukovat tyto aplikace podle jejich interakce se systémovými prostředky (soubory, síťovým provozem, mutexy, registry, a chybovými hlášeními generovanými operačním systémem) a tím zjednodušit manuální analýzu. Navržený model je navíc schopen seřadit jednotlivé shluky aplikací pro následnou analýzu jejich nebezpečnosti, která je poté použita při definici adaptační hry. Schopnost klasifikovat a shlukovat neznámé aplikace byla ověřena na rozsáhlé množině vzorků.

Contents

Chapter 1

Introduction

Due to the increased frequency and sophistication of cyber-attacks carried out over the Internet, protection of the critical infrastructure has become more important than ever before. Governments, corporations and other institutions deploy multilayered security solutions, with an *intrusion detection systems* (IDS) [1] at the core, to protect their networks. Deployed next to tools for *policy enforcement* (firewall), *intrusion prevention systems* (IPS) or *host-based detection systems* (antivirus software), they often serve as the last line of defense against threats such as *targeted attacks*, *advanced persistent threats* (APTs) or *sophisticated malware*.

One particular type of an IDS that received a lot of attention in research community is an *anomaly-detection-based IDS* [2, 3]. It employs statistical modeling and machine learning techniques to automatically detect threats by monitoring irregularities in the network traffic. The main benefit is its independency on the static database of externally-maintained threat intelligence or hand-designed rules. On the other hand, anomaly-detection-based IDS typically requires precise tuning of its internal parameters before deployment to real-world networks [4] to prevent overwhelming number of false alarms.

Traditional approach to optimize parameters of an IDS relies on labeled static dataset of the network traffic. During the process, the optimal configuration is pre-selected and used across all deployed instances of the IDS. However, such approach does not respect different profiles of individual networks (bank vs. academic vs. retail), the dynamic nature of the network traffic (different profile of the network during the day and night or working days and weekends), or the rapid evolution of the threat landscape (new malware families or their variants are emerging literally every day). This leads to situations when an attack, easily detected during the nighttime, remains hidden in the daytime traffic, or when a new type of attack is completely missed by the IDS.

An alternative approach is to tune the IDS system dynamically as it is deployed. However, such approach requires labeling of the background traffic which is typically not available in online deployment. Moreover, using the background traffic directly makes the configuration process susceptible to manipulation as an adversary has direct access to the data used for tuning. Such attacker can mislead the IDS by insertion of a sequence of attacks that are orthogonal to its actual plan, and that would make the IDS less sensitive w.r.t the actually dangerous attacks.

The approach proposed by Rehák et al. [5] is positioned between completely offline and completely online approach. It is based on mixing known prerecorded examples of both legitimate and malicious traffic called *challenges* [6] into the background traffic. The challenges are then used to tune the system against various types of malicious behavior such as different types

of attacks or command and control channels used by malware. As challenges are completely under operator's control, attacker's options to manipulate with the configuration of the IDS are reduced.

However, for adversary with the complete knowledge of the system it is still possible to attack the adaptation process itself and increase the chances to perform a successful attack [7]. To protect the adaptation process and tune the IDS in environments with such advanced adversaries, *game theory* [8] is a natural choice. It studies interactions of two or more actors, called *players*, with potentially antagonistic goals and provides concepts for solving such interactions. As such it was successfully adopted, albeit offline, for optimization of parameters of IDS in scenarios with rational adversary [9, 10, 11, 12, 13, 14, 15]. However, adopting game theoretical principles in online adaptation of an IDS system with the assumption of a rational attacker is still an open question.

As we have discussed above, the approach proposed by Rehák et al. [5] relies on traffic traces called challenges and assumes that they are realistic representations of the real-world traffic. However, using static database of fixed traces is not optimal as it provides data with only limited variability and updating process based on manual analysis of the unknown network traffic does not provide sufficient amount of data to cover frequent modification of malware behavior employed by malware authors. This aspect raises a question how to keep the database updated or how to replace the static database and provide labeled data in sufficient quality and volume.

1.1 Research problems

The open questions discussed in previous paragraphs can be summarized into the following research problems:

- *RP1: How to dynamically tune parameters of an IDS system in an adversarial environment?*

  In situations when static configuration fails to capture differences between individual networks, the dynamic character of network traffic, or changes in the threat landscape, the adaptation process based on mixing pre-recorded traces of network traffic offers promising results. However, attacks against the adaptation process may reduce its effectiveness. In this thesis, we propose to tune the configuration of an IDS online such that the damage caused by possible attacker with the complete knowledge of the system is minimized.

- *RP2:How to obtain large amount of labeled data necessary for adaptation/validation of an IDS system?*

  The key component of the adaptation process is the labeled data (challenges). Mistakes in labels, or mismatch of the profile of both legitimate and malicious traffic, or unrealistic artifacts in the data can cause failure of the adaptation process and limit the operational performance of the IDS. However, obtaining high quality evaluation data is difficult as manual labeling of network traffic requires deep understanding of both threat landscape as well as the principles of the network communication itself. Even a highly-skilled expert is often unable to provide accurate labels as they typically depend on the context (e.g. connection to google.com can be either legitimate if an user is using it for search, or can be a sign of malicious infection if it is used by malware as connection check). In this thesis we divide the problem of obtaining evaluation data into two subproblems:

  - *RP2a: How to obtain sufficient amount of benign training data?*

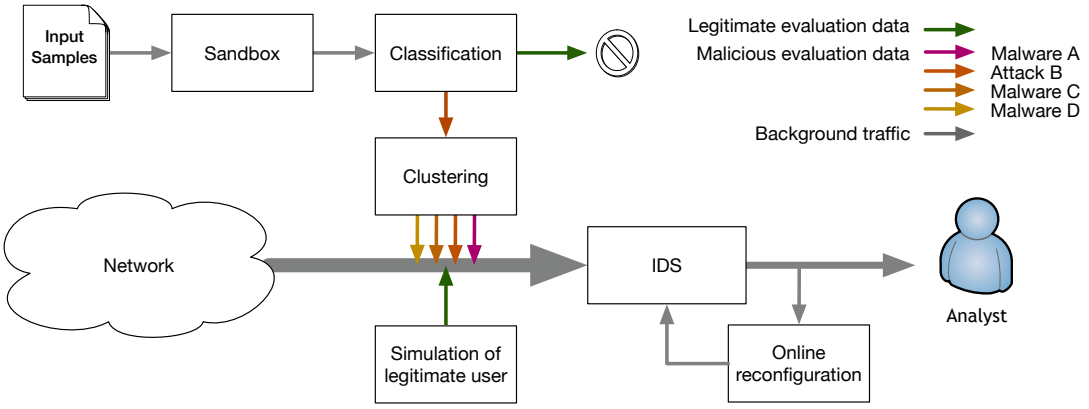


Figure 1.2.1: Schema of the proposed architecture described in the thesis.

– *RP2b: How to obtain fresh examples of malicious behavior?*

It has been recognized by the community [16, 17, 18] that simulation and emulation of the network traffic provides data that can be used for evaluation of an IDS system. In this thesis, we propose a solution that connects both these approaches to obtain evaluation data suitable for the adaptation process.

1.2 Proposed architecture

The research problems defined in the previous section motivate the solution illustrated in Figure 1.2.1. It extends the idea originally proposed by Rehak et al. [5] in two ways. First, we introduce robust approach to online reconfiguration in the environment with advanced adversary based on game-theoretical principles. The proposed solution tunes the system with respect to administrator's preferences for various types of attacks/malware families available to the attacker and at the same time it minimizes the possible damage caused by successful attacks. Second, we extend and improve the original idea of challenges with simulated traffic in the case of legitimate users and emulated traffic in the case of malicious behavior to provide large amount of realistic network traffic suitable for online adaptation of an IDS.

1.2.1 Game-theoretical online reconfiguration

As we have described in *RP1*, the online adaptation process is vulnerable to advanced attacks performed by an attacker with the knowledge about the internal state of the IDS. In Chapter 3 we frame the problem in the game-theoretical framework. We specify the problem as a dynamically defined two-player non-zero-sum game with utility function specifically designed for the purpose of dynamic reconfiguration of the IDS and use standard solution concepts used in game theory to solve this game. Using this formalism, we are able to dynamically select configuration with minimal loss with respect to various types of malicious behavior represented by challenges and thus minimize attacker's gain.

1.2.2 Simulation of legitimate behavior

It has been recognized by the community [19, 16, 20, 21] that it is possible to artificially simulate the legitimate behavior such that the IDS is not able to distinguish simulated and real network traffic. In Chapter 4 we present a generative model designed to simulate realistic traffic of legitimate users in the form of NetFlows [22], lightweight format suitable for anomaly-detection-based IDS [23]. The model is based on the assumption that high-level behavior of a legitimate user is stable in time, meaning that even though there are dynamic changes in the behavior (day vs. night, working days vs. weekends, etc.) his long-term behavioral patterns are stable (e.g. he uses the same e-mail server every day for long time, or visits the same web sites, etc.).

1.2.3 Emulation of malicious behavior

Compared to the behavior of legitimate users, the network behavior of malicious actors changes much more rapidly, as attackers frequently devise new techniques to avoid detection. Since finding new malware, its analysis and building models of its behavior can take days or weeks (or in case of sophisticated malware even months [24]), using complex stochastic model for simulation of malicious network traffic would be impractical. Thus, we propose to employ emulation instead of simulation. We execute various malware samples in a controlled environment (*sandbox*), record their network traffic and use them as challenges. Problem is, that even though there is large number of samples[1] we can use, their classification (malware/legitimate) and categorization into various malware families is unknown and manual analysis does not scale to the amount necessary for the purpose of adaptation.

To address this problem, we provide an approach that instruments all samples in sandbox, classifies them as legitimate or malicious and clusters the malicious ones into coherent groups. Even though the groups are not categorized into malware families, samples in a single group exhibit similar behavior so the complexity of manual analysis is significantly reduced.

The proposed approach is based on assumption that actions of a sample are visible through its interaction with system resources, which in this work includes *(1) interactions with files* (e.g. during encryption of a victim's hard drive), *(2) network communication* (e.g. during data exfiltration or displaying advertisements), *(3) operation with mutexes* (e.g. used to ensure a single instance of malware is running), *(4)* manipulation with *registry keys* (e.g. to ensure persistency after reboot), and *(5) error messages* of the operating system itself. As the number of interactions vary for every sample (every sample interacts with different files, registry keys, etc.), we apply multiple instance learning paradigm [25], designed to model such data, which builds a vector representation of the sample suitable for standard machine learning algorithms. Unknown samples are then classified using *random forest* classifier [26] and the ones considered as malicious are clustered using *probabilistic generative model* (*Bernoulli mixture model* [27]). The benefit of the probabilistic formalism is not only, that it is able to correctly cluster the data, but we are able to derive (1) prioritization schema that promotes clusters better suited for human analysis and (2) we can extract indicators of compromise (IOC), examples of files, registry keys, mutexes etc., that can help an analyst and can be used to identify the malware family in the future.

[1] In this context by sample we mean anything that can weaponized by malware, i.e. PE executables, PDF files, JAR files, documents with macros, etc.

1.3 Key contributions

- **Game-theoretical approach to online reconfiguration of an IDS system [28, 29]** (Chapter 3). We present a self-adaptation mechanism for network intrusion detection system based on the game-theoretical formalism to address *RP1.* The key innovation of our method is a secure runtime definition and solution of the game and real-time use of game solutions for immediate system reconfiguration. Our approach is suited for realistic environments where we typically lack any ground truth and where the significant portion of system inputs may be shaped by the attacker whose goal is to render the IDS ineffective. Therefore, we employ the concept of challenge insertion: we inject a small number of simulated traffic (both malicious and legitimate) into the background traffic and use the system response to the challenges to define the game structure and utility functions. This approach is also advantageous from the security perspective, as the manipulation of the adaptive process by the attacker is far more difficult.

- **Simulation of legitimate behavior [30]** (Chapter 4). We propose a generative model of the user's behavior covering different aspects of the network traffic. It generates of new data and thus it extends the principle of challenges. The possibility to continuously generate examples of network traffic overcomes the key problem of challenge insertion–limitation of variability of static challenges. In experimental evaluation, we demonstrate that the model generates data practically indistinguishable from the real-world samples, which makes it well-suited for the purpose of online reconfiguration and thus addresses *RP2a.*

- **Classification of sandboxed samples [31]** (Chapter 5). We propose a novel approach to model behavior of unknown binaries executed in sandbox based on their interaction with system resources (files on the filesystem, mutexes, registry keys), network communication with remote servers and error messages generated by operating system. Since the number of system resources the binary interacts with vary for every binary, we adopt *multiple instance learning* that is specifically designed to handle such data. The proposed approach was applied in two-class classification scenario necessary for solving the first part of *RP2b,* for preselecting malware binaries. The extensive evaluation on large-scale real-world dataset proves that the proposed approach outperforms current state-of-the-art as it achieves the classification error lower by 30%.

- **Clustering of sandboxed samples [32]** (Chapter 6). In order to provide complete framework for analysis of unknown samples we propose a probabilistic generative model (Bernoulli mixture model) based on the representation described in Chapter 5, which clusters malware samples according to their behavior. Moreover, the probabilistic model allows human-friendly prioritization of identified clusters and extraction of readable behavioral indicators to maximize interpretability. Using this method human analysts are able to annotate large number of malicious samples that can be used in adaptation process which completes *RP2b.* The quality of the proposed model was evaluated on large-scale experiments using real-world dataset and compared to related state-of-the-art approaches. The evaluation prove that the proposed approach is able to discover the malware families more precisely than state-of-the-art with the added benefit of human-friendly prioritization and extraction of behavioral indicators.

1.4 Outline of this thesis

The thesis is organized as follows:

Chapter 2 summarizes review of relevant prior art. It is divided into three main parts covering game-theoretical configuration of an IDS, simulation of network traffic and analysis of malware binaries.

Chapter 3 describes the principles of the runtime adaptation in the adversarial environment. It provides definition of the security game and its solution concepts, and evaluates its impact on the real-world IDS.

Chapter 4 proposes probabilistic generative model that extends the concept of static challenges used in Chapter 3 so that their shortages are overcome. The quality of the generated traffic is experimentally evaluated on real-world IDS.

Chapter 5 introduces novel approach to model malware's behavior based on multiple instance learning. The performance of proposed model is verified on large-scale dataset and compared to state-of-the-art approaches on the two-class classification problem.

Chapter 6 introduces novel approach to analysis of unknown samples executed in sandbox based on the representation introduced in Chapter 5. This chapter proposes a probabilistic generative model applied to the problem of clustering of malicious binaries as well as prioritization of generated clusters and extraction of high-quality behavioral indicators to simplify the human analysis. The performance of the proposed approach is experimentally verified and compared to the state-of-the-art approaches.

Chapter 7 summarizes the contributions of this thesis and provides list of publications.

Chapter 2

Related work

In this chapter, we will review the recent work related to main topics of this thesis. First, we will discuss the state-of-the-art approaches related to game-theoretical configuration of an IDS system (Section 2.1), followed by review of approaches for simulation of network traffic (Section 2.2) and recent work focused on analysis of malware samples (Section 2.3).

2.1 Game-theoretical reconfiguration

Using game theory in security applications (physical security [33, 34], jamming and eavesdropping of wireless networks, IDS configuration, etc.), is a popular approach for modeling real-world situations where we assume rational attacker [35, 10, 36]. It provides solid mathematical framework for modeling interactions between two (or more) opponents with possibly antagonistic intentions, and provide solution concepts that the optimal decision process can be based on. In this section, we review the work related to configuration of an IDS system under the assumption of rational attacker. The discussed approaches are summarized in Table 2.1.

2.1.1 Reconfiguration of single IDS system

One of the first applications of the game theory in intrusion detection systems was proposed by Alpcan et al. [37]. Authors propose IDS composed of multiple detection nodes distributed in the network and define two-player non-zero-sum game between attacker and defender to optimally assign operational thresholds for the proposed IDS. Authors further extended their work in [38] where they relax the assumption about perfect detection of attacker's actions. They introduce third fictional player that represents the IDS with imperfect detection of attacker's actions. Next, in [39] authors use their previous results and define the interaction between attacker and defender as stochastic game and model the state of the network using finite-state Markov chain. The optimal solution of the game is then find using reinforcement learning (Markov-decision-process value iteration, minimax-Q, naive Q-learning).

Next step in the game-theoretical reconfiguration of an IDS was proposed by Zhu et al. [10]. Authors propose to use *stochastic zero-sum dynamic game* to model dynamic behavior of attacker and defender. They further assume that defender (the IDS system) can be in one of particular states $s \in S = \{s_1, \ldots, s_n\}$, which for $n = 2$ the state can represent whether the system is infected or not. Next, authors assume that attacker has limited set of possible attacks $\mathcal{A} = \{a_1, \ldots, a_M\}$ with predefined damage d_i at his disposal and defender has finite set of libraries $\mathcal{L} = \{l_1, \ldots, l_N\}$ assigned costs of deployment c_i, where every library l_i can detect only

Approach	Game type	IDS	#players
Alpcan [37]	two-player, non-zero-sum game	S	2
Alpcan [38]	two-player, non-zero-sum game	S	2
Alpcan [39]	two-player zero-sum stochastic game	S	2
Liu [9]	two-player static/dynamic Bayesian game	S	2
Reddy [40]	two-player zero-sum game	S	2
Nguyen [41]	two-player non-zero-sum game with imperfect inf.	S	2
Zhu [10]	stochastic zero-sum dynamic game	S	2
Zhu [12]	cooperative game	S	2
Lisý [42]	imperfect-inf. zero-sum extensive form game	S	2
Zhu [43]	two-player zero-sum game	S	2
Moosavi [14], [15]	two-player non-zero-sum discounted stoch. games	S	2
Chen [44]	static cooperative/non-cooperative game	C	$N+M$
Zhu [11]	non-zero-sum stochastic game	C	$N+M$
Jin [45]	two-layer stochastic game	C	$N+M$
Proposed approach	two-player, non-zero sum dynamically defined game	S	2

Table 2.1: Overview of the related work for configuration of single IDS system (S) or cooperative IDS (C) along with details about the adopted game-theoretical concept.

	Monitor	Not monitor
Attack	$(1-2\alpha)w - c_a, (2\alpha-1)w - c_m$	$w - c_a, -w$
Not attack	$0, -\beta w - c_m$	$0, 0$

(a) Defender's opponent is attacker node.

	Monitor	Not monitor
Not attack	$0, -\beta w - c_m$	$0, 0$

(b) Defender's opponent is regular node.

Table 2.2: Utility function for the game defined by Liu et al. [9] where w represents the value of protected asset, α is defender's true positive rate and β represents defender's false positive rate, and $c_m, c_a > 0$ represent costs of monitoring and performing attack respectively.

subset of attacks and nothing else. Actions taken by the defender represent loading/unloading particular library and actions taken by attacker represent performing particular attack. Authors state that optimal policies can be computed either offline or using online learning. Furthermore, authors propose an extension of their approach based on *Q-learning* that can be used to estimate the optimal policy in case that transition probabilities between states are not known, and prove that under mild constraints their iterative algorithm converges to the optimal *Q-function* and yields to optimal policies.

Zhu et al. further extend their previous idea in [12] where they introduce the solution to optimal signature-based IDS system. They relax the condition from [10] that particular library always detect an set of attacks and introduce the probability α_{ij}^P that particular attack a_i can be detected with library l_j, and α^P-weighted detectability of attack that represents the *effectiveness* of an IDS against attack a_i composed of a set of detection libraries. Using cooperative game authors describe influence of individual detection libraries, apply Shapley Values and Banzhaf-Coleman index to solve the optimization problem and provide optimal configuration of an IDS system.

Zhu et al. [43] extend their idea of an IDS system to the whole network and postulate that attacker can gain detailed information about static network using sufficient number of scans. To address this problem, authors propose to dynamically reconfigure the network structure and thus disrupt the attacker's knowledge. The aspect that authors need to optimize is how frequently reconfigure the network infrastructure as frequent changes disrupt attacker's knowledge but the legitimate business usages as well.

Besides traditional area of deployment, wireless sensor networks present important area that requires protection as they are vulnerable to various types of attacks (jamming, eavesdropping, etc.). Problem is that, traditional technologies are typically not applicable here due to limitations of power, storage, or processing capabilities. Reddy [40] applies two-player zero-sum game to solve the problem of allocation of detection mechanisms, where the goal is to keep the network operational and minimize the total energy spent by the nodes in the network with the assumption of the rational attacker.

Liu et al. [9] adopt the two-player Bayesian game for monitoring of wireless ad hoc networks. Authors assume three types of nodes in the network: (1) *defender node* with actions monitor or not monitor, (2) *attacker node* with actions attack or not attack and (3) *regular node.* Authors further assume that defender do not know whether particular node is attacker or regular but rather define prior probability μ_0 that particular node is attacker. The payoff matrix in strategic form given in Table 2.2 formalizes the interaction between defender and two remaining types of node. It can be derived that in case that

$$\mu_0 < \frac{(1+\beta)w + c_m}{(2\alpha+\beta-1)w}$$

the Bayesian game admits single pure strategy that defender plays *Not monitor*, and the opponent plays *Attack* if the node is malicious and *Not attack* if it is regular. In case that

$$\mu_0 > \frac{(1+\beta)w + c_m}{(2\alpha+\beta-1)w}$$

only mixed strategy exists. Authors further extend the proposed game into dynamic Bayesian game where defender's prior probabilities about node being malicious are updated in every step. The approach is then deployed in hybrid IDS system where defender can choose between lightweight IDS and heavy IDS with different costs of monitoring. The benefit of such deployment is that the lightweight IDS with much lower monitoring costs can save significant amount of energy without complete loss of protection.

Similarly, Moosavi et al. in [14] and [15] address the problem of protecting wireless ad hoc or sensor networks. Authors assume that due to the nature of these networks continuous monitoring is not possible as individual nodes have only limited resources (battery, bandwidth, etc.) and IDS can monitor only one wireless channel at the time. Therefore, in this work authors adopt *discounted stochastic games* to model the situation when attacker tries to disrupt wireless ad hoc network by compromising critical number of nodes, whereas defender's goal is to prevent such disruption by protecting clean nodes and recovering nodes that were already compromised. Individual states of the game then represent number of compromised nodes in the network. Authors verify the game definition on simulated scenario and analyze critical number of compromised nodes under various configurations. Such analysis then can serve as a guideline for deployment and configuration of various security solutions in wireless ad hoc networks.

Slightly different scenario is studied by Lisy et al. [42] where authors propose to use game-theoretical approach for adversarial plan recognition. Authors define the problem of plan recognition as an imperfect-information zero-sum game in extensive form between the attacker and the detector, and provide novel generalization of Monte-Carlo tree search to approximate the optimal solution (*Nash equilibrium* [8]). The experimental evaluation proves that using game-theoretical approach outperforms various naive approaches used for plan recognition.

2.1.2 Networked and collaborative IDS

Networked and collaborative IDS systems are natural extension of single IDS system deployed to multiple nodes in the network. The earliest works assume that IDS systems are collaborating honestly without selfish intentions [13, 46].

One of the first work that introduces the game theory in the field of cooperative IDS is proposed by Chen et al. [44]. Here authors apply the work proposed by Liu et al. to the network with multiple targets, each of them with different value, and use the *Nash equilibrium* [8] as solution concept to provide optimal strategies for attacker and defender. They further extend the definition of their problem to the situation with multiple attackers and multiple defenders and study two principal cases: (1) each node is monitored by one defender at most and (2) multiple defenders monitor single node. Authors provide theoretical lower bounds for minimal numbers of defenders in each scenario and provide optimal strategy that can be used for configuration of networked IDS.

Zhu et al. [11] apply their results derived for optimal configuration of single IDS [10] into the scenario with multiple IDS systems and multiple attackers. Authors define stochastic nonzero-sum dynamic game with $N + M$ players and provide iterative algorithm that converges to ϵ-*Nash equilibrium* [47]. Authors further define *security capacity* as the "largest payoff achievable under Nash equilibrium" which provides upper bound to the value of the target that can be compromised and specifies targets that can be realistically compromised.

More recent work proposed by Jin et al. [45] addresses the problem of collaboration of fully connected network of multiple IDS systems for the detection of various attacks. Authors propose two-layer architecture where the first layer consists of two-player stochastic game in which each IDS learns its own optimal strategy using *Q-Nash learning algorithm* [48]. The second layer than collects all strategies learned by individual IDS and employ Vickrey-Clarke-Groves auction to optimally distribute the learned information such that the amount of available resources for every IDS system is not exceeded.

Name	Type	Output[1]	IDS eval.	Network eval.
α, β-profiles [50]	C	P,N	X	
Swing [21]	C	P	X	X
LESS [19]	C	N	X	
cnaf [17]	C	P	X	
FLAME [51]	P	N	X	
SSH brute-force attacks [52]	P	N	X	
NS-3 [53, 54]		P		X
NeSSi2 [55, 56]		P	X	X
OMNet++ [57]		P		X
Mininet [58]		P		X
Proposed approach	P	N	X	

Table 2.3: List of discussed simulation techniques with type (C=complete, P=partial) and output (N=NetFlows, P=packets).

2.2 Simulation of network traffic

This section provides review of work focused on simulation/emulation of network traffic. It has been recognized in the research community [20, 49, 16], that simulation of network traffic provides a viable alternative to manually labeled real-world traffic and proves to be useful for tuning/evaluation of an IDS system. It overcomes the main problems of real-world traffic, such as the insufficient scalability and high cost of manual labeling, uncertainty of labels, or privacy issues with sharing real network traffic. Moreover, using simulated traffic is typically the only option for exhaustive evaluation of an IDS as the data with necessary variety is often difficult or even impossible to obtain on real network.

2.2.1 Simulation for evaluation

In the first part, we will review the work solely focused on evaluation of an IDS. It can be divided into two principal groups. The first group focuses on generating traces of complete traffic composed of both legitimate and malicious traces which allows sharing of complete data necessary for tuning or evaluation of an IDS. The major concerns with such approach are the realism of generated data and complexity of the simulation. To overcome the problems of the simulation of complete traffic, the second group of works deals with the simulation of only malicious traffic and mix it with prerecorded traces of background traffic. Such approach provides the best possible realism of background data as it is gathered on real network but limits the possibility to publicly share the complete dataset. The complete list of discussed approaches is listed in Table 2.3.

Complete traffic simulation

In order to generate complete traffic, Shiravi et al. [50] propose approach that mixes network traffic generated according to two types of descriptions (*profiles*). The first type, *α-profile*, describes malicious behavior either in an exploit language (ADeLe [59]) or as prerecorded traces of network communication. The second type, *β-profile*, is used to describe statistical properties of legitimate (background) traffic using histogram of various statistical properties of network traffic as no well-known distributions capture them correctly (authors considered Normal, Beta,

Wei-bull, Erlang, Triangular, Gamma, Exponential, Uniform, and Lognormal). Every instance of β-profile then characterizes behavior of a single user using a single protocol (HTTP, FTP, SSH, SMTP, IMAP or POP3) in a single day, extracted from training data. Authors argue that these profiles contain only statistical information or malicious behavior, and thus there are no privacy concerns that prevent sharing. The main drawback of this approach is its complicated adaptation for the purpose of online evaluation of IDS since the generation of legitimate traffic is performed as connections over real network.

Vishwanath et al. [21] propose *Swing,* "a closed-loop, network-responsive traffic generator". It uses simple and semantically meaningful underlying model of the transmitted packets populated from prerecorded network traces. Authors argue that such model is able to reflect changes in structure of the network, covers changes in application layer (e.g. changes in application behavior generating packets) and generates traffic based on current state of the network with slightly different parameters. The model is divided into four basic categories: *users* (how often users are active, thinking time, etc.), *sessions* (e.g. number and target of individual connections within a session), *connections* (destination of the connection, size of request and corresponding responses, wait time before generating response, etc.) and *network characteristics* (loss rate, latency, etc.). Authors claim that different applications such as FTP client, P2P client, or e-mail client exhibit different characteristics (*signatures*) and therefore have to be considered separately. The model with trained parameters is then used to generate packet traces. In the simulation scenario the number of *generators* and *listeners* is specified using application specific signatures. Furthermore, the simulation scenario contains specification of topology of the emulated network along with specific level of bandwidth, latency and loss rate that are extracted from the source trace. The simulated trace is then recorded as packets transmitted over the simulated network. This approach is well-suited for creating datasets with precise network characteristics but the overhead of the emulated network complicates its application in online adaptation.

Sonchack et al. [19] propose *Large-scale Evaluation for Security Simulation* (LESS), an agent-based simulation approach that generates traffic based on tunable stochastic processes. It uses similar approach to Swing and α, β-profiles as it extracts description of the legitimate traffic from static traces and combines it with model of malicious behavior. The key difference is that LESS is specifically designed for evaluation of large-scale security systems. The simulation framework first analyzes static traces, detects applications used on both client-side and server-side, and extracts necessary statistics (number of applications per host and server, ratio of hosts that used particular application). Next, it instantiates predefined number of agents and assign them detected applications (both clients and servers). The network traffic is then simulated by communication between individual agents. In order to validate their approach, authors used different evaluation criteria than authors of Swing and α, β-profiles. In this paper authors compare the generated traces with specific attack scenarios with real ones using four large-scale security systems, namely *entropy-based anomaly detection* [60], *Highly Predictive Blacklisting System* [61], *Peer-to-peer Botnet Detection* [62] and *Collaborative Anomaly Detection Boggs* [63]. Their results prove that it is possible to generate traffic traces that are for security systems indistinguishable from the real ones.

Abt et al. [17] (*cnaf*) argue that tools used for simulation of network traffic are typically too complex and too difficult to work with. Therefore, instead of using statistical models of the traffic, *cnaf* defined behavior in user-defined scripts easily understandable to human analyst. The simulation process is built on virtual machines (VMs) grouped into virtual networks that executes various applications to simulate different types of network behavior (web browsing, instant messaging and email communication). The generated traffic is then recorded on the level of VM hypervisor connected to the Internet. Authors argue that such approach ensures

high scalability and extensibility. However, using user-defined scripts limits the variability of the generated traffic as it requires manual definition of every new behavior.

Simulation of malicious behavior

Brauckhoff et al. [51] present simulation tool based on *anomaly injection* principle designed for evaluation of anomaly-based IDS systems. Authors use clean background NetFlow data and mix it with a simulated malicious activity. The malicious activity is then labeled as an anomaly that should be detected by the IDS system. Authors categorize anomalies into three main classes according to their effect on the background traffic: (1) *additive anomalies* that add NetFlows without affecting the background traffic (e.g. network scan or bot activity), (2) *subtractive anomalies* that remove specific NetFlows from the background traffic (e.g. outage events or shifts to other AS peerings), (3) *interactive anomalies* that add NetFlows and alter the background traffic (e.g. DDoS attacks that consume network bandwidth or consuming large portion of the processing power). Authors acknowledge that models of anomalies that should be injected into the background traffic requires expert knowledge and large insight into behavior of modeled anomaly. In order to create corresponding models, authors propose to use large archive of anomalies [64] that provides sufficient baseline for correct definition of large number of different anomalies.

Sperotto et al. [52] propose different approach to generate evaluation/tuning data than papers we have discussed above. In this paper authors focus only on single specific attack scenario—penetration of SSH servers using brute-force attack (using specific dictionary or randomly generated passwords), since it is, according to their analysis, the most common attack scenario carried out over the Internet. Authors divide the SSH brute-force attack into three phases: (1) *scanning phase* during which attacker performs sequential scan looking for target servers, (2) *brute-force phase* during which the attacker performs the actual SSH brute-force attack to selected subset of targets gathered in the first phase and (3) *die-off phase* that corresponds to the residual traffic generated shortly after the attack. All three phases of the attack are modeled with *discrete time Markov chain* with transition probabilities estimated from training data. The validation of the proposed approach prove that the probabilistic model is able to correctly capture all aspects of this attack scenario and generate realistic data.

2.2.2 Network Simulators

Network simulators are frequently used in the field of general network research where they are used to verify new low-level network protocols such as routing protocols, transfer protocols or new network topologies [54, 65, 58].

Typically, network simulators employ discrete event simulation (DES) [66, 67] that models the evolution of the simulated network in discretized time events (e.g. start or end of packet transmission, beginning of communication, etc.). The discrete event simulation further assumes that no important change, that affects the simulated network, happens between two consecutive time windows and the traffic is then emitted in these time windows.

One of the first network simulators, NS simulator [49, 68], was originally developed as a replacement of the REAL project [69]. Its latest version, NS-3 [54], is designed as high fidelity discrete event simulator that allows users to simulate both wired and wireless networks with large number of different devices (routers, switches, etc.) and topologies. However, the primary focus of the NS-3 simulator is on evaluation of low-level network protocols rather than evaluation of an IDS system.

OMNet++ [57] is another example of discrete event simulator which designed as framework rather than complete simulation environment. Therefore, in its basic form, OMNet++ does not contain simulation models that can be used for simulation of network traffic. Note that different simulation models and different simulation tools using OMNet++ such as INET [70] or OverSim [71] were developed by different research groups for different evaluation purposes.

Mininet [58] adopts OS-level virtualization instead of DES. It simulates individual hosts in the network as processes encapsulated in separate network namespaces connected via virtual Ethernet pairs. This design allows efficient simulation of large-scale networks which makes the Mininet popular tool for design and simulation of software defined networks.

NeSSi [55] and its ancestor NeSSi2 [56] adopts different idea than simulators discussed above. Their primary purpose is to generate realistic traffic for packet-based IDS systems such as SNORT or Bro, which is the main difference to NS-3 or OMNet++ simulators. The NeSSi2 simulator is divided into simulation backend using the JIAC framework [72] and the graphical frontend that allows users to intuitively create different simulation scenarios. The agent-based approach allows to distribute the simulation and thus easily increase performance of the simulator.

2.3 Malware analysis

Since the analysis of malicious binaries and recommending them for further analysis has important practical applications, there exists a rich prior art. Although it is frequently divided into three main categories, static analysis, dynamic analysis and hybrid analysis, the boundaries between them are blurred since techniques such as analysis of the execution graph are used in both static and dynamic analysis.

2.3.1 Static malware analysis

Static malware analysis treats a malware binary as a data file from which it extracts features without executing it. The earliest approaches [73] looked for a manually specified set of specific instructions *(tell-tale)* used by malware to perform malicious actions but not used by legitimate binaries. Problem is that such simple approach is typically not able to detect polymorphic and obfuscated malware that changes its code and structure. Since reversing obfuscation and polymorphic techniques are in theory NP-hard [97], most of the recent state of the art [74, 80, 98] moved to a higher-level modeling of sequences of instructions/system calls and estimating their action or effects on the operating system. The rationale behind such shift in the paradigm is that higher-level actions are more difficult to hide. In following paragraphs, we will discuss two of the most prevalent higher-level representations used in related work.

Call graphs One of the most popular high-level representation is *control flow graph (CFG)*. It captures the flow of a binary as a graph where nodes represent basic blocks (sequence of instructions without any jump) and edges represent dependencies between these blocks. The control flow graph can be further extended to *call graph* where each node represents a function (both internal and external) and edges represent dependencies between these functions.

Christodorescu et al. [74] propose to use annotated control flow graph extracted from individual instructions and map it to *an automaton* used for classification. Authors propose several techniques to prune the graph in order to remove randomization and obfuscation frequently used by malware. Problem is that such approach fails to detect packed or encrypted binaries.

Approach	T	Goal	Method	#samples
Lo [73]	S	C	Static set of instructions (*tell-tale*)	-
Christodorescu [74]	S	C	static signatures of CFG	7
Cesare [75]	S	C	static signatures of CFG	-
Kinable [76]	S	L	function call graphs	194
Kong [77]	S	M	function call graphs	526 179
Santos [78]	S	C	n-grams	2000
Reddy [79]	S	C	n-grams	500
Ahmadi [80]	S	M	n-grams, metadata, images, entropy, etc.	21 741
Naval [81]	D	C	syscalls (Ordered System-Call Graph)	3751
Wuchner [82]	D	C	syscalls (quantitative data flow graph)	7507
Kolbitsch [83]	D	M	syscalls (behavior graph)	300
Park [84]	D	M	syscalls (behavior graph)	380
Pfoh [85]	D	C	syscalls (String kernels)	4 565
Lanzi [86]	D	C	syscalls (n-grams)	242
Canzanese [87], [88]	D	M	syscalls (n-grams)	$\sim$ 76 000
Rieck [89]	D	C	syscalls (cluster prototypes)	33 698
Bayer [90]	D	L	syscalls	14 212
Pirscoveanu [91]	D	C	syscalls + counts of selected files, mutexes, registry keys, DNS	42 068
Mohaisen [92]	D	C/M/L	features extracted from files, registry keys, network	115 157
Mohaisen [93]	D	M	n-gram model of behavioral traces	2699
Rieck [94]	D	M	higher-level malware actions	10 072
Bailey [95]	D	L	files, registry keys, processes, network communication	$\sim$ 3700
Anderson [96]	H	C	dynamic instructions, static instructions, n-grams of bytes, opcodes, etc.	22 492
Proposed approach	D	C/L	files, mutexes, reg.keys, network traffic	250 527

Table 2.4: List of discussed approaches for two-class classification (C), multi-class classification (M), clustering (L) of malware samples using dynamic analysis (D), static analysis (S) and hybrid analysis (H).

To address this problem, Cesare et al. [75] extends the idea of Christodorescu et al. and propose an approach for automatic analysis of packed binaries and extraction of approximative function call graph. Then, instead of annotating the graph per instruction, authors match individual sequences of extracted code to static signatures stored in database to detect malicious code.

Kinable et al. [76] propose to cluster call graphs extracted from the binaries using k-medoids and DBSCAN clustering algorithms with the similarity between two graphs defined by *graph edit distance*. The assumption is that malware binaries from the same malware family have similar structure captured in call graph and therefore the clustering reconstructs individual malware families.

In more recent work, Kong et al. [77] propose to extract call graph in order to capture the structure of the execution tree of the binary and then for each node in the call graph (i.e. a function in the binary code) define set of static features (histogram of instructions in function, number of memory readings or writings in the function, etc.). Authors then optimize distance function between two graphs using *maximum margin principle*. It states that two binaries from the same malware family should close to each other whereas two binaries from different malware families should be separated with large margin. The experimental evaluation indicates that the trained distance correctly separates large portion of selected malware families.

n-gram models Another approach, inspired by text analysis, is based on n-gram models of binaries and instructions within. Each n-gram is a sequence of n consequent bytes, instructions, etc., extracted from the binary. Note that the length of the n-gram has to be specified in advanced. Typically, n-gram model is represented using *bag-of-words* representation where each n-gram is considered as separate feature. Such approach is proposed by Santos et al. [78] where authors extract n-gram from the binary itself without any preprocessing and the binaries are then classified with k-nearest neighbor approach.

The main problem with n-gram models is the exponential growth of number of features with the length of the n-gram. To address this issue, Reddy et al. [79] select only limited number of the most frequent n-grams in both legitimate and malicious binaries. Using such approach, they are able to limit the number of extracted n-grams and thus improve the performance and scalability of their approach.

In more recent work, Ahmadi et al. [80] propose to combine n-gram model with other features extracted from the binary such as metadata, image representation, frequency of symbols, frequency of API calls, etc. The concatenated feature vector is pruned with *forward feature selection* in order to select the most important features for the classification. Authors use XGboost [99] to classify unknown binaries into individual malware families.

2.3.2 Dynamic malware analysis

Typically, approaches based on static analysis struggle to analyze binaries that are packed or encrypted. An alternative solution to overcome these problems is the execution of a binary in a controlled environment (sandbox) and analyzing its interactions with the operating system and system resources.

A large portion of the work related to dynamic malware analysis utilizes system calls, since in modern operating systems system calls are the only way for applications to interact with the hardware and as such the system calls can reveal malware actions. Another source of data are the higher-level actions executed by the malware (writing into file, modification of registry keys, starting new processes, etc.). In this section, we will discuss the most relevant works that employs both of these approaches.

System calls The simplest methods identifying malware samples from sequences of syscalls rely on n-grams [86, 87, 88]. However, simple model based on bag-of-words representation of n-grams of syscalls is not able to correctly separate malware and benign software on practical level [100]. In order to improve the performance, Lanzi et al. [86] extend the information extracted from n-grams of syscalls with details about files that were executed, modified or read in specific directories, and with information about operations with registry keys performed by the analyzed binary. The presented results indicate that the extended model is capable to correctly detect the malicious samples. Another problem with n-gram-based models is their size. As we have discussed in previous section, the number of features defined by the n-grams grows exponentially with the length of the n-grams which make the use of the raw feature space is impractical. To address this principal problem, Canzanese et al. [87, 88] propose to transform the original feature space into feature space with much lower dimension using various approaches (*singular value decomposition, linear discriminant analysis*) and thus improve the classification performance. Another extension of the n-gram model is proposed by Pfoh et al. [85] where authors adopt *string kernels* rather than similarities based on bag-of-words representation to measure similarity between two system call traces.

Another drawback of methods based on n-grams is that malware can mask its true behavior by executing meaningless system calls, and thus avoid detection. To capture more complex relations between individual system calls, large number of related work encodes the malware's behavior into graph structure. Park et al. [84] propose to generate behavioral graph from the sequence of system calls and estimate their distance between using *maximal common subgraph*. Kolbitsch et al. in [83, 101] propose to extract the behavioral graph similar to Park. However, the key difference is that instead of estimating the distance between two graphs, Kolbitsch et al. search for key points in the code (*slices*) that are then recorded and stored in database. Behavior of an unknown binary is then compared to these slices and if a match is found, the binary is considered as malicious.

In contrast to the previous approaches, Wuchner et al. [82] propose to model the malware behavior using *quantitative data flow graphs (QDFGs)*. This approach provides abstraction of malware's behavior and captures the interactions between individual components of the operating system using the data flow rather than temporal coincidence. Similar approach is proposed by Naval et al. [81] where authors model the sequence of executed system calls using *Ordered System-Call Graph (OSCG)* and extract the most relevant execution paths using *Asymptotic Equipartition Property*. The most relevant paths are then used to construct the vector representation used for classification.

Different approach is proposed by Rieck et al. [89]. In this paper, authors use normalized histograms of n-grams as feature vectors, which effectively embeds syscall sequences into Euclidean space endowed with L_2 norm. In this space the algorithm extracts prototypes $Z = \{z_1, \ldots, z_n\}$ using hierarchical clustering. Each prototype captures behavior of the cluster, which should match corresponding malware family. An interesting feature is that if a cluster has less than a certain number of samples, the prototype is not created.

Vast majority of approaches for analysis of unknown binaries focus on classification (two class, multi-class). Bayer et al. [90], on the other hand, propose an approach for malware clustering based on modeling of system calls. Authors taint certain portions of memory, such as output arguments and output values of system calls, and tracks all operations with the tainted memory to generate traces of system calls. This allows to uncover dependencies between individual system calls even when they are interleaved with unrelated ones and provides information necessary for creating behavioral profile of the analyzed binary. These profiles are then clustered with an algorithm based on locality sensitive hashing.

Recently, Pirscoveanu et al. [91] proposed an approach that combines analysis of system calls with analysis of higher-level actions. Along with system calls, authors extract DNS requests, accessed files, mutexes and registry keys that the binary interacted with. This data is then filtered using manually defined whitelists in order to ensure that the data contains only behavior related to malware.

Higher-level actions Since the popularity of the system calls has already triggered the development of evasion techniques such as shadow attacks [102], system-call injection attacks [103], or sandbox detection [104], researchers explore different source of information suitable for malware analysis.

AMAL proposed by Mohaisen et al. [92] uses custom sandbox to intercept and log interactions of the malware binary with files and registry features and its communication over the network. From these interactions AMAL extracts high-level numeric features such as counts or sizes of created, modified or deleted files, counts of created, modified or deleted registry keys, counts of unique IP addresses, etc., and uses single-linkage clustering to identify similar binaries. Unlike AMAL, the work presented in this thesis uses resource names instead of their numerical properties to construct its features. Moreover, the generative model allows to prioritize founded clusters and extract typical characteristics of each cluster. Another approach proposed by Mohaisen et al. [93] called CHATTER, models the dynamic behavioral traces produced by their custom sandbox using n-grams.

Rieck et al. [94] creates a representation of analyzed samples without manually defined conversion of the input data, which consists of the names of system calls and its parameters. The calls are treated as words, specifically each system call name together with all its parameters corresponds to one word. To allow generalization, Rieck et al. creates $n + 1$ additional words from a syscall with n parameters by iteratively removing its last parameter. This causes explosion of the number of features, for example in our experiments to represent 6 000 samples needs about 20 million features. Although this representation is sparse, it is still difficult to work with and limits the scalability. To prevent this explosion and to allow scaling, the work presented in this thesis clusters resource names as described in Chapter 5. Also, Rieck et al. models actions triggered by the malware (writing into a file, communication with remote server, reading data from registry keys, starting new thread, etc.), whereas the proposed approach models only affected resources. This enables to deploy the proposed approach in environments without direct access to or low visibility of low-level actions (VMs without such access, user machines without API hooking). Another key aspect is that Rieck et al. proposes a prioritization of syscalls to aid the manual analysis. However, their approach is tailored to supervised scenario when labels are available whereas the proposed approach is able to extract behavioral indicators directly from the unknown samples without any labels.

Bailey et al. [95] propose idea similar to the one proposed in this thesis as authors model the malware's behavior based on its external manifestations. Authors record *process names*, *modified registry keys*, *modified file names*, and *network connection attempts* and use them to define malware's behavioral profile. To evaluate the similarity between two behavioral profiles X and Y authors use *normalized compression distance* defined as

$$\mathrm{NCD}(X, Y) = \frac{C(X + Y) - \min(C(X), C(Y))}{\max(C(X), C(Y))},$$

where $X+Y$ represents concatenation of behavioral profiles and $C(X)$ represents zlib-compressed length of profile X. In contrast to this approach, in this thesis we define similarities specifically designed to capture different aspects of individual sources of information (files, registry keys,

network connections) and project the data into numerical vector rather than defining similarity between complete behavioral profiles. Using this approach, we are able to prioritize generated clusters and extract humanly readable behavioral indicators.

2.4 Hybrid analysis

Anderson et. al [96] propose to combine approaches from static analysis with the data obtained using dynamic analysis in order to counter techniques frequently used by malware authors to avoid detection, e.g. packing or execution stalling. Authors propose six different types of input data, three based on techniques from static analysis: (1) features extracted from raw binary modeled as n-grams, (2) opcodes extracted from disassembled binary and (3) *control flow graph*—a graph of all possible execution paths; two based on dynamic analysis: (4) instruction traces [105] and (5) system call traces; and (6) one based on various information extracted from the binary itself such as packer identification, entropy of the binary, number of instructions in disassembled file, etc. For every type of input authors define a kernel which are then combined using *multiple kernel learning [106]* to obtain optimal combination. The optimized kernel combination is then used with SVM classifier.

Chapter 3

Runtime optimization

This chapter presents a game theoretical model of local adaptation processes inside an autonomic, self-optimizing Intrusion Detection System [6]. Our goal is first and foremost to analyze the risks related to opponent's manipulation of system internal state and configuration in order to reduce its effectiveness. This addresses the existing concern with expected increase in malware sophistication—theoretical models for distributed learning in malware exist [107], and strategic manipulation of Intrusion Detection Systems by shaping of the input data has been demonstrated, albeit offline [108]. This behavior corresponds to wider context of targeted attacks on learning processes, studied in the fields of adversarial machine learning and adversarial classification [109].

Therefore, if we want to introduce a new layer of environment-driven adaptation into the intrusion detection system [6], we need to ensure what is the extent to which can the opponent misuse the reconfiguration layer to reduce system's effectiveness.

The principal question this chapter investigates is simple: What is the cost of preventive IDS resistance to the attackers with access to internal state information and outputs of an IDS, in terms of suboptimal False positives/False Negative values? In other words, we measure whether and by how much will the IDS reconfiguration reduce its performance against the "worst case", highly sophisticated attacks with insider access compared to the "standard", relatively unsophisticated attackers with no knowledge of IDS existence, nominal effectiveness and current internal state.

In order to answer the above question, we use the methods from the field of game theory [8] and decision theory. These concepts, introduced in Section 3.1 are mapped to IDS structure in Section 3.1.1. They conceptualize the relationship between the attacker and the defender as a two player, non-zero-sum game, where the attack/defense actions of both players correspond to strategies in the game-theoretical model of their interaction.[1]

[1]Note that this chapter is based upon work supported by the ITC-A of the US Army under Contract W911NF-12-1-0028 and by ONR Global under the Department of the Navy Grant N62909-11-1-7036 and work supported by Czech Ministry of Education grant AMVIS-AnomalyNET: MSMT ME10051 and MVCR Grant number VG2VS/189. Parts of this chapter were originally drafted for the final report of project W911NF-12-1-0028, next were used in master thesis [110] and were subsequently published in [28] with extension to [29].

3.1 IDS Game Model

In this chapter, we will use the simplest model available in the field of the game theory, a single stage game of two players [111]. Each such game is defined as a three tuple:

$$G = (P, S, U) \tag{3.1.1}$$

- where P is a set of **players** traditionally indexed as $P = \{1, 2\}$, in our case denoted $P = \{d, a\}$, where the player a is the attacker (the column player) and the player d is the defender (the row player),
- S is a set of **strategies** available to all players. In our case, as the strategies are disjunctive, we impose simply $S = \{d_1, ..., d_i, ..., d_m, a_1, ..., a_j, ..., a_n\}$, here the strategies d_i are those of the defender and the strategies a_i are available to the attacker, and
- U denotes **utility function** of the form: $U : S \times S \rightarrow \mathbb{R} \times \mathbb{R}$, or less formally: $U : d_i \times a_i \rightarrow (u_d, u_a)$. Utility function returns the game payoff of the defender u_d and the attacker u_a when these invoke the strategies d_i and a_i respectively. Payoffs are real valued, and are frequently negative. Note that the negative value of payoff signifies the loss for the player, and unlike in the case of zero-sum games, this loss does not become other player's gain. The game structure can be alternatively defined by two matrices that link the attacker's and defender's strategies with the payoff functions for each player[2]. Such alternative definition is defined in Equation 3.1.2 for defender and 3.1.3 for attacker.

$$u_d = \begin{pmatrix} Def./Att. & a_1 & a_2 & a_3 & \dots & a_n \\ d_1 & u_d(d_1, a_1) & u_d(d_1, a_2) & u_d(d_1, a_3) & \dots & u_d(d_1, a_n) \\ d_2 & u_d(d_2, a_1) & u_d(d_2, a_2) & u_d(d_2, a_3) & \dots & u_d(d_2, a_n) \\ \vdots & \vdots & \dots & \dots & \ddots & \dots \\ d_m & u_d(d_m, a_1) & u_d(d_m, a_2) & u_d(d_m, a_3) & \dots & u_d(d_m, a_n) \end{pmatrix} \tag{3.1.2}$$

$$u_a = \begin{pmatrix} Def./Att. & a_1 & a_2 & a_3 & \dots & a_n \\ d_1 & u_a(d_1, a_1) & u_a(d_1, a_2) & u_a(d_1, a_3) & \dots & u_a(d_1, a_n) \\ d_2 & u_a(d_2, a_1) & u_a(d_2, a_2) & u_a(d_2, a_3) & \dots & u_a(d_2, a_n) \\ \dots & \dots & \dots & \dots & \dots & \dots \\ d_m & u_a(d_m, a_1) & u_a(d_m, a_2) & u_a(d_m, a_3) & \dots & u_a(d_m, a_n) \end{pmatrix} \tag{3.1.3}$$

The gameplay of this game is very simple in our case: both players simultaneously select their strategies from the set S and the combination of these strategies determines the payoffs to attacker and defender, as defined by their respective utility functions. Note that due to the inherent nature of the IDS problem, the game is not a zero sum one as for example investigation of a security incident (an attack) typically costs the defender more than is the cost of performing the attack for the attacker. Therefore, combination of strategies affects not only the distribution of payoffs between the players, but also the total sum of payoffs.

3.1.1 Intrusion Detection Game Specification

The game structure introduced above allows us to reason about the outcome of the interaction between the attacker and the defender, and potentially identify the likely outcomes of the game. Below, we will present the details of player's strategies, utility functions and solution concepts that will influence the outcome of the game.

[2]These two matrices can be collapsed into a single one in case of the zero sum game, where the gain of one of the player is directly translated into the equivalent loss of the other player.

Strategies

The pure strategy sets of both players form the following set

$$S = \{d_1, ..., d_i, ..., d_m, a_1, ..., a_j, ..., a_n\}$$

as defined in Eq. 3.1.1. The strategies $d_1, ..., d_i, ..., d_m$ are the strategies of the defender (row player), while the strategies $a_1, ..., a_j, ..., a_n$ are available to the attacker.

The defender's **pure strategies** are defined as a selection of one system configuration from a finite number of available configurations - playing the game is therefore functionally equivalent with the trust-based optimization described in [112], where we have introduced an online regret-minimization mechanism suitable for dynamic and unstable environments.

When the defender plays a **mixed strategy**, it constructs a probability distribution over the set of pure strategies $d_1, ..., d_i, ..., d_m$, assigning a probability in the $[0, 1]$ interval to each pure strategy. The sum of probabilities (weights) of individual strategies must be 1. In practice, the mixed strategies will typically have a restricted support, with roughly 2-5 pure strategies with non-zero probability.

The attacker's pure strategies are defined even more easily. Each attacker's strategy is defined by performing one attack such as horizontal scan, vertical scan, host fingerprinting, buffer overflow, denial of service and others. Mixed strategies are defined similarly to defender, as a probability distribution on the support of attack actions. In practice, attackers also execute their strategies in a particular, logical orderings (plans), but the identification and use of this behavior is outside of the scope of this chapter. For more detailed discussion we refer reader to [42] where authors propose game-theoretical approach to identify of attacker's plan.

Utility Functions

In contrast to previous work in IDS modeling [37, 39, 9, 44], the utility functions that we use to represent player's gains and losses are not simplified, but are designed to provide a realistic model of incentives in real IDS system. This is made possible by the fact that we don't attempt to perform a formal analysis of the problem (even if we are still able to identify and verify several key properties of the system), but rather concentrate on online discovery of game parameters and runtime solution of the game in the context of specific threat environment and network traffic situation.

The form of the utility functions determines the characteristics of the game – if the game is a **zero sum game**, i.e. the sum of utilities of all players is constant over any combination of played strategies, it is relatively easy to identify stable Nash equilibria and other solutions, as we will discuss in Section 3.1.2. However, in our case, the game is not zero sum which is a natural corollary of the criminal character of the activities [113].

The utility functions in our game are relatively complex, as they need to reflect the complexity of the problem. The utility function of the defender has three principal components: the first term deals with successfully detected attacks, the second term represents the loss associated with undetected attacks and the third term describes the overhead of the monitoring, which consists of the costs of false alarms (false positives) and the fixed cost of monitoring. Individual utility functions are defined as follows. Defender's utility is:

$$u_d(d_j, a_i, t) = \alpha_{ij}(D_d(a_j) - C_{TP}) + (1 - \alpha_{ij})\gamma_j P_d(a_j) \quad (3.1.4)$$

$$-\beta_i V(t) C_{FP} - C_M \quad (3.1.5)$$

Attacker's utility can be described as:

$$u_a(d_j, a_i) = \alpha_{ij} D_a(a_j) + (1 - \alpha_{ij})\gamma_j P_a(a_j) - C_a(a_j) \tag{3.1.6}$$

The parameters of both utility functions are:

- α_{ij} denotes the probability that the attack strategy a_j is detected when the defender selects the defense strategy d_i. Intuitively, in our case, it estimates the probability that the given detection strategy (i.e. IDS configuration), combined with current status of internal models of the IDS and the background traffic will be able to successfully raise an alarm upon the occurrence of an attack from the class corresponding to a_j.
- β_i denotes the probability that a given detection strategy i, combined with current system state and background traffic, will result in a false positive. Note that this element is only present in defender's utility matrix, and its manipulation can be used by the attacker to launch denial-of-service attacks on detection mechanisms [114].
- γ_j denotes the probability of attack success. It discounts the value of undetected breach from both the attackers and defenders standpoint, and typically features relatively low values.
- $V(t)$ denotes the background traffic volume that is used to estimate the number of false positives in combination with the parameter β_i.
- $P_d(a_j)$ denotes the defender's payoff/loss on attack success. It is most often a negative value, except for multi-tier honeypot systems or very particular situations where the defender can gain knowledge from the attacker bypassing the IDS. The loss can be relatively low in case of exploratory activities (scan, fingerprinting), but is relatively high when the attacker actually breaches a system.
- $P_a(a_j)$ denotes the expected utility the attacker receives upon successful realization of given attack action from the attack class corresponding to strategy a_j.
- $D_a(a_j)$ denotes the attacker's payoff/loss on detection, which is typically a negative value. The value of this parameter can vary widely, as it can be very high for last stages of elaborate attacks executed inside defender's perimeter (e.g. cleaning exploited machine), or can be almost zero in case of internet attacks.
- $D_d(a_j)$ denotes the defender's payoff for attack detection. This parameter value is the main cause for the game not being a zero sum game in a general case, as the payoff is typically zero internet settings following the similar reasoning as in the $P_d(a_j)$ case – damages from the attacking party are almost impossible to seek (not even considering the problems related to the root attacker identification and burden of proof).
- $C_a(a_j)$ denotes the cost of the attack performance on the part of attacker. Typically very low for internet-originating attacks.
- C_{TP} - denotes the (average) cost of processing of each detected incident (true positive) for the defender.
- C_{FP} - denotes the average cost of a false alarm for the defender, used in conjunction with β and $V(t)$ to estimate the false positive cost.
- C_M - denotes the fixed cost of monitoring infrastructure, independent on attack or traffic intensity.

Utility function terms. The first situation that we represent corresponds to attack detection. The term in the defender's utility function describing this situation (without the fixed costs of monitoring and false positives, which will be discussed below) is:

$$\alpha_{ij}(D_d(a_j) - C_{TP}) \tag{3.1.7}$$

We can see that the defender may get some payoff from attack detection, but globally, the value of the term $D_a(a_j)$ would be zero or negative due to the reinstallation and recovery costs. The term C_{TP} represents the immediate cost of incident detection, investigation and processing.

On the attacker's side, this situation is described by the term:

$$\alpha_{ij} D_a(a_j) \tag{3.1.8}$$

We can see that the loss of the attacker depends almost entirely on the impact the detection has on attacker's plans – the value can be relatively high in the last stages of complex attack plans deep within the protected perimeter, but is next to zero for malware propagation on Internet due to the lack of effective enforcement.

The second term of the utility function covers the situation when the attacks are not detected.

In the defender's case, the term is:

$$(1 - \alpha_{ij})\gamma_j P_d(a_j) \tag{3.1.9}$$

The first factor corresponds to non-detection probability, while the factor γ_j describes the likelihood of attack/exploit success, which then amortizes the value of the successful execution $P_d(a_j)$. The impact of the factor γ_j is crucial. When there is an attack, the actual *defender's optimum in most situations is that the attack is both undetected and unsuccessful.* Reasoning behind this analysis is straightforward: Equation 3.1.7 typically has the term $D_d(a_j)$ zero or negative[3], the term $-C_{TP}$ is also negative and the best strategy is therefore to avoid detection of attacks with low loss/likelihood product $\gamma_j P_d(a_j)$.

From the attacker's perspective, the second term is a straightforward amortization of success payoff by success likelihood and non-detection probability. Failure to exploit (with probability $1-\gamma_j$) is also preferable to detection for the attacker, but only by the slight margin of the term $D_a(a_j)$, which is typically low for external attacks:

$$(1 - \alpha_{ij})\gamma_j P_a(a_j) \tag{3.1.10}$$

The conclusion that some (i.e. unsuccessful) attacks are better left undetected may seem surprising, but it actually corresponds to very natural equilibria, due to the costs associated with any detected attacks. The problem in this case is therefore how to optimize the sensitivity of the intrusion detection system, so that it will only detect the relevant threats. We attempt not only to remove the false positives, but also discount the value of true positives with little relevance to the actual system. This behavior ensures more effective monitoring with little or no impact on security.

[3]The only situations where this term is actually globally positive are those where an efficient counter-attack mechanism (in tactical/military problems) or intelligence-processing mechanism allows the defender to counter-attack the attacker's resources or to deduce attacker's goals, plans or at least intentions. From the other side of the problem, the attacker needs to structure its actions in such a way, that their eventual compromise would not give away disproportionally high volume of information about its goals or resources. This consideration is integrated in the value of the term $D_a(a_j)$.

The last component of both utility functions captures the costs related to cyber-attack or defense. Attacker's side lost utility can be trivially described as the cost associated with attack performance:

$$-C_a(a_j) \tag{3.1.11}$$

The defender's utilities depend on two principal factors: cost of the monitoring infrastructure and the cost of the processing of false positives, which can be significant for real world systems:

$$-\beta_i V(t) C_{FP} - C_M \tag{3.1.12}$$

The first of the two components estimates the number of false positives ($\beta V(t)$) and the total cost of their assessment ($\beta V(t) C_{FP}$), while the second term captures the fixed cost of monitoring, such as the infrastructure cost and fixed operation costs.

The size of these two terms is non-negligible – the number of false positives can rival the number of real incidents in open networks (see the Experimental section 3.3 for more details), and false positives would typically significantly outnumber the true positives on internal, well-managed networks. These two terms are also the main reason why the IDS game is not a zero sum game, as they introduce a fundamental non-efficiency into the system.

It is important to note that in typical cyber-attack scenarios, the game is actually highly asymmetric, as the attacker's costs $C_a(a_j)$ and potential losses are almost zero in the individual attack case, and the defender's much higher losses are amplified by relatively high cost of monitoring and false positives processing as specified in Equation 3.1.12.

3.1.2 Solution Concepts

The definition of the game alone does not allow the player to identify the optimal behavior. There are several well-accepted solution concepts, based on different criteria of optimality. The ones that we have considered are the most commonly used ones:

- **Max-min rule.** This solution concept (similar to Minmax rule) selects the strategy with the highest minimal payoff for the player. This solution concept is a security strategy, and is especially relevant in the situations where we suspect that the opponent has access to the part of the system's internal state or even to strategy selection decision. Playing max-min (for the defender) covers the risk of the opponent playing the most damaging action:

  $$d* = \arg\max_{d_i} \min_{a_j} u_d(d_i, a_j). \tag{3.1.13}$$

  While being a relatively strong solution concept, it shall be noted that the max-min criteria does not require any knowledge of the opponent's utility function—the selected strategy depends only on the defender's utility functions alone which is not true for the last concept introduced below.

- **Nash equilibrium.** The Nash equilibrium is a strong solution concept that identifies a stable combinations of player's strategies. It is defined as a state where no player can improve his payoff by unilaterally changing his strategy. Formally, the equilibrium, defined as a pair of strategies (either pure or mixed) of both players needs to fulfill the following condition:

  $$\begin{aligned} &(d_i, a_j) \text{ is a NE iff } \forall d_l, l \neq i : u_d(d_l, a_j) \quad \leq u_d(d_i, a_j) \\ &\text{and } \forall a_k, k \neq j : u_a(d_i, a_k) \leq u_a(d_i, a_j), \end{aligned} \tag{3.1.14}$$

where d_i and a_j are to be considered as mixed strategies. The major difference with the max-min rule is the number of equilibria solving the condition 3.1.14. It can be shown that when we admit solutions in mixed strategies, the IDS game as specified in this section always has at least one Nash equilibrium. However, we are typically able to identify more than one equilibrium in the game, and the players are then confronted with the problem which one to select (by playing the corresponding d_i or a_j strategy).

The performance of solution concepts will be analyzed in Section 3.3, where we will compare them on both the challenge data[4], and also on their ability to handle a real world instance of an actual attack scenario. To understand the results, we need to understand the difference between the various types of optimality criteria in the system [115]:

- **Optimal strategy** is the player's best **pure** strategy from the set S in the time t. This is an **a-posteriori** concept, which can be only determined after the execution was completed and the system results against the **actual** network attacks were determined. This value is independent of the solution concept used.

- **Optimal decision** is the strategy (mixed or pure) identified by the player as optimal a-priori, given the inputs available **a-priori** in the moment that the decision is taken. Making the optimal decision does not guarantee actually selecting the optimal strategy, principally for two reasons: information about the state of the system is incomplete/limited/biased (making the optimal strategy not present in the optimal decision, or decreasing its selection probability in the mix), or the pure strategy selected stochastically from the optimal decision was not actually the optimal strategy. Optimal decision can be obtained by the use of any of the four solution concepts described above (or any other solution concepts, such as trust-based mechanism), as each of the solution concepts introduces a slightly different bias into the optimality criteria.

The utility difference between the two concepts is called **regret**, and reflects the quality of the model, randomness of the environment, strategic behavior of the opponent and the cost of hedging against such strategic behavior. In the experiments presented in Section 3.3, the regret of different solution concepts is evaluated and compared.

The solution concept also tightly connects the security of the IDS system and the quality of the decisions it is able to achieve. The first concept—max-min rule—does not require any knowledge of opponent's plans intentions or goals, as they only consider the information about player's own decision function. On the other hand, reaching Nash equilibria requires that both players either interact over a longer period of time, or have at least some knowledge of opponent's utility function. Otherwise, they would not be able to identify the equilibria and can gain less profit (or rather more loss) than when playing max-min.

Under some circumstances, it might be even rational to disclose some information about the system to the attackers, in order to avoid the solutions which leave both players worse off. However, the practical implementation of this concept may be challenging, and our original intuition regarding the usefulness of the Nash equilibria as a solution concept was skeptical. This was to some extent disproved by the results of the experiments from Section 3.3, where it performs on par with other concepts even without explicit information transfer.

[4]Challenges are prerecorded sets of network traffic that are manually labeled as legitimate or malicious and can be seen as training samples.

3.2 Game Integration for Runtime Reconfiguration

In this section, we will describe the integration of the game-theoretical model with the adaptation process of a particular IDS. This integration consists of several steps: dynamic parameter estimation in the system, game definition, game solution and integration of results back into the system.

There are two existing integration options addressing the problems from the opposite sides of the spectrum that reflect the two traditional approaches for tuning parameters of an IDS:

- *Off-line integration*, when the game is defined and solved analytically and the system parameters are configured according to game results[116]. This is the most traditional way of using the game theoretical methods, as their use ensures that the system parameters are set to force the adversary into the selection of less damaging (or more rational) strategies. The advantage of this approach is relatively easy solution identification and low technical difficulty, but the disadvantage, similarly to the offline tuning of an IDS, is the fact that the game solutions identify the behavior that is advantageous on average, and do not reflect the dynamic changes of assumptions, threat characteristics and background traffic. This is sufficient for systems deployed in stable environments, but most IDS need to cope with dynamic environments, where the background traffic and other factors change frequently.

- *Direct on-line integration*, when the game uses presumed adversary actions in the observed network traffic to define the game. The game is being defined by the actual actions of real-world attackers executed against the monitored system. This approach addresses the problem with game definition relevance by using the actual attacks and traffic background to define the game at runtime. The game is then solved as an optimization problem, but with several drawbacks. Direct interaction between the adversary and the adaptation mechanism makes the system potentially vulnerable to attacks on machine learning and adaptation algorithms [109], making the whole IDS potentially less secure than without the use of game-theory driven adaptation. Motivated attacker can easily mislead the IDS by insertion of a sequence of attacks that are orthogonal to its actual plan, and that would make the IDS less sensitive w.r.t the actually dangerous attacks.

The approach adopted in this thesis, named *indirect online integration*, combines the above approaches and provides interesting security properties desirable for real-world deployment. The solution uses the concept of challenges to mix a controlled sample of legitimate and adversarial behavior with actually observed network traffic and is a compromise between the above approaches (see Figure 3.2.1). In this case, the real traffic background (including any possible attacks) is used in conjunction with simulated hypothetical attacks (challenges) on IDS input. The system's response to the simulated traffic is used as an input for game definition as it is used to estimate probabilities of detection of individual attacks α_{ij} and probability of false positive β. The major advantage is higher robustness w.r.t strategic attacks on adaptation algorithms, and lower system configuration predictability by the adversary, as the simulation runs inside the system itself and its results cannot be easily predicted by the attacker.

This approach offers the optimal mix of situation awareness and security against engineered inputs. In this case, we actually play against an abstract opponent model inside the system, and expect that the moves that are effective against this opponent will be as effective against the real attacks. The advantage of this approach is not only in its security, but also in better model characteristics in terms of strategy space coverage (less frequent, but critical attacks can be covered), robustness and relevance—the abstract game can represent the attacks and utility combinations that would be obvious only for insider attackers.

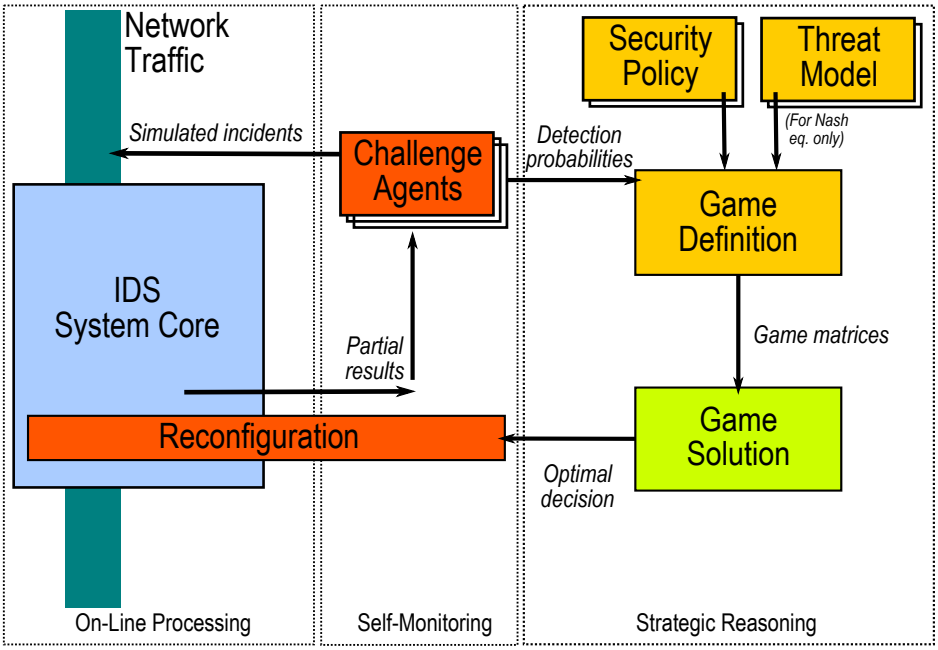


Figure 3.2.1: Indirect online variant of integration of the game with IDS.

3.2.1 Indirect Online Integration

The use of the indirect online integration in practice requires a division of the covered time interval into sub-intervals defining each single game as a sequence. The length of such interval depends on the IDS technology used, line speed, hardware performance and other factors — it can vary between few seconds for pattern-matching packet filters to few minutes/hour for statistical anomaly detectors.

During each interval t, the system measures/estimates the values of parameters (in particular the detection probabilities α_{ij} and the false positive probabilities β_i, V — discussed in details in Section 3.1.1). For the reasons listed above, we suggest the use of challenge-based parameter estimation [112], which relies on insertion of known instances of legitimate or malicious behavior into the background, unclassified traffic. We measure the system response to these challenges, drawn from the realistic attack classes, and use them to estimate the system response to all real-world samples from the same classes. In practice, we will define one class for each broadly defined attack/legitimate traffic type and measure the difference between the system response to legitimate traffic and to various classes of malicious traffic. The adaptation process will then assess the statistical properties of the response and use them to estimate the probability of detection of each strategy combination α_{ij} (where the index i specifies the defender's strategy, i.e. system configuration, and the index j denotes the attack type, i.e. attacker's strategy) and the corresponding expected ratio of false positives β_i for given defender's strategy i. It is worth noting that this method is based on the assumption that the response of the detection method used in the IDS against members of each class is consistent and that the anomaly scores of the class members are distributed according to normal distribution. This assumption has been verified in our past work [5], and can be ensured as the attack class definition is under the full control of game designer — if the response to one of the classes is for example multimodal, it can be easily split into separate classes.

The game definition ordering with respect to each time interval also depends on the type of the underlying IDS. The CAMNEP system [6] is a NetFlow-based [22] collective anomaly detector, and therefore processes the data in well-defined and regularly produced batches rather than in real time. This means that the game is actually defined **after** the data has been recorded. In case of traditional pattern matching IDS that needs to operate on wire speed, the game needs to be defined and solved **beforehand**, so that the strategies can be applied directly to each processed packet, flow or connection. In practice, this means that the systems solving the game after the interval t on which the solution is applied have precise parameter estimations for each particular interval, while the wire-speed systems apply the t-th game results to the interval $t + 1$.[5]

In both cases, once the system obtains the game definition and solves it, it can directly apply the results back into the system configuration and use them on current or next time interval.

3.2.2 Game Strategies for Real World IDS

To test whether the game theoretical concepts can be integrated with a real IDS, we have used the CAMNEP system [6]. As we have noted in Section 3.2.1, the CAMNEP is a NetFlow-based IDS system. In addition, CAMNEP already features self-monitoring and self-optimizing functionality, allowing us to benchmark the performance of game-theoretical self-optimization with other approaches. The existing self-monitoring capabilities are also essential for online empirical estimation of the key utility function coefficients α_{ij} and β_i (see Section 3.1.1), as their values typically evolve throughout the day.

The CAMNEP system is based on a self-organized, multi-level collaboration of detection algorithms, each of them maintaining a different model of traffic normality/anomaly. The algorithms share the anomaly estimates at various stages of processing and once they have reached their partial conclusions (anomaly scores for each network flow/connection), the system needs to aggregate these opinions together. At this stage, it is important to notice that the performance of individual detection algorithms and their combinations varies with background traffic and attack types. For more information about the CAMNEP system, please refer to [29].

The defender strategies in CAMNEP are instantiated as specific aggregation functions used to integrate the opinions of detection algorithms in the system. Defender's strategy selection is thus technically straightforward, as it only picks one particular aggregation operator that aggregates diverse expert opinions with particular weights or methods. In our experimental system, there were 30 operators aggregating the opinions of 6 detection algorithms in total.

3.3 Experiments

In the experimental evaluation we compare the detection performance of the self-adaptation mechanism based on game-theoretical principles described in this chapter to the adaptation mechanism based on trust modeling originally proposed by Rehak et al. [6].

The underlying CAMNEP system manages optimal selection of challenges and their mixing into the background traffic. The selection of challenges is based on a simple threat model [112], which includes estimation of defender's risk and potential losses. The response of the system to the challenges is then used both as an input for the original, trust-based self-adaptation mechanism and for the game-theoretical mechanism, running on the same traffic data and inserted challenges. This ensures that the experimental results, averaged over 40 system runs

[5] The slight delay of application is unlikely to cause a problem, as suggested by our experimental results. The system using the parameters weighted over 5 last intervals performed comparably with the one using only the precise values for the specific interval.

Type of attack	Description
SSH bruteforce	Dictionary based attack with 100 attempts per attack passwords were randomly selected from predefined database.
	Dictionary based attack with 300 attempts per attack, passwords were randomly selected from predefined database
Vertical scan	Vertical TCP scan performed against linux server with enabled OS detection
	Vertical UDP scan for all services performed against linux server
Horizontal scan	Horizontal scan for SSH service performed against network of Department of Cybernetics
	Horizontal UDP scan for DNS service
	Horizontal ICMP ping scan
	Combination horizontal and vertical scan performed against whole network of Department of Cybernetics

Table 3.1: Parameters of performed attack samples (real-world attacks).

on the same inputs of the system, fairly compare the influence of various techniques. The individual runs vary by the actually selected challenge values, as these are selected stochastically from a challenge database.

In our experiments, we want to measure two effects. First, we will compare the ability of the game-theoretical methods to deliver at least comparable performance on the inserted challenges. Then, we will judge the effectiveness of the selected configurations while detecting a classical, real-world attack sequence comprising of exploratory activities: horizontal and vertical scanning, followed by password brute force attack on the SSH service on one of the vulnerable hosts. The results of the second experiment can be then used to measure how does the performance on challenge data translate to the performance on real attack detection.

We compare 6 different solution concepts for the defender: `MaxMin1` and `MaxMin5`, `Nash1` and `Nash5` and `Trust1` and `Trust5`. The name of the method comprises of the strategy selection method, Max-Min rule, Nash equilibrium and original trust-based approach, and the number (either 1 or 5) that determines the number of periods over which the system behavior is observed: the concepts with the "1" suffix react to immediate situation only, while the solution concepts with the suffix "5" consider the values (α_{ij} and β_i) aggregated over the last 5 intervals (25 minutes in total).

3.3.1 Experiments' settings

To correctly evaluate the system's ability to strategically select the best aggregation function we had acquired data recorded on a mid-size university network, with relatively low and stable background activity that comprised of 100 5-minute long intervals. We manually classified recorded data in order to obtain a dataset with a mix of partially classified third-party traffic and our attack with known properties. We have classified most of the legitimate traffic (roughly 75–80% NetFlows of legitimate traffic). The performed attacks summarized in Table 3.1 replicates most common attack scenario, the goal of which is to gain administrator access to the protected system. In these experiments we evaluated brute-force attacks with various speed and different types of scans used to discover vulnerable services (i.e. horizontal scans, vertical scans, various types of fingerprinting, etc.).

Next, we have to specify all coefficients necessary to evaluate the utility function for attacker

Attack strategy	$P_d = P_a$	γ_j
Horizontal scan	300	0.001
SSH brute force request	500	0.001
SSH brute force response	500	0.001
Vertical scan	300	0.001

Table 3.2: Game parameter values for different strategies.

and defender as well (see Equations (3.1.4) and (3.1.6)). Part of these coefficients is listed in Table 3.2. Note that the selection of the coefficients dependent upon the attacker's strategy corresponds with the coefficients obtained from attack trees proposed in [112]. The rest of the coefficients are listed in Equations (3.3.1), (3.3.2), (3.3.3), (3.3.4), (3.3.5) and (3.3.6). Note that the selection represents situation when defender nor attacker gain any asset (or worse in the case of attacker) when the attack is detected. This covers attacks performed from Internet by unknown attacker.

$$C_a(a_j) = 0 \ \forall a_j, \tag{3.3.1}$$
$$C_{FP} = 1, \tag{3.3.2}$$
$$C_M = 0, \tag{3.3.3}$$
$$D_a(a_j) = 0, \tag{3.3.4}$$
$$D_d(a_j) - C_{TP} = 0 \ \forall a_j, \tag{3.3.5}$$
$$V(t) = 1 \ \forall t. \tag{3.3.6}$$

3.3.2 Challenge-based results

In this section we will present results measured on challenges artificially inserted into the background traffic. To fulfill this goal we define the following evaluation function

$$\mathcal{E} = \frac{\bar{\theta}_y - \bar{\theta}_x}{\sigma_x + \sigma_y} \tag{3.3.7}$$

where $\bar{\theta}_y$ and $\bar{\theta}_x$ represent mean and σ_y and σ_x represent the standard deviation of legitimate and malicious challenges. The value of the criterion increases when the detection process correctly separates the legitimate and malicious traffic (i.e. $\bar{\theta}_y$ tends to 1 and $\bar{\theta}_x$ tends to 0) and at the same time minimizes their standard deviation which corresponds with the *trust experience* defined in [112]. Note that the challenges used for the following measurement equal to the challenges used to find optimal aggregation function using trust modeling approach, i.e., the training set. Therefore, in theory, the results of this experiment should show that the solution Trust1 gives the best results.

The values of evaluation criteria $\mathcal{E}$ for all solution concepts are summarized in Table 3.3 extended with values for an average strategy, referred as `AvgOWA`, computed as an average value of evaluation function for all strategies in every time step, and values of best strategy, referred as `Best`, computed as an average value for the best possible strategy in every time step. The values of evaluation criteria confirm the theoretical assumption that `Trust1` provides the best results. However, detailed analysis of the evaluation criteria for every time step (Figures 3.3.2 and 3.3.1) revealed that it is not assured as between the intervals 8–20 `MaxMin1` outperforms

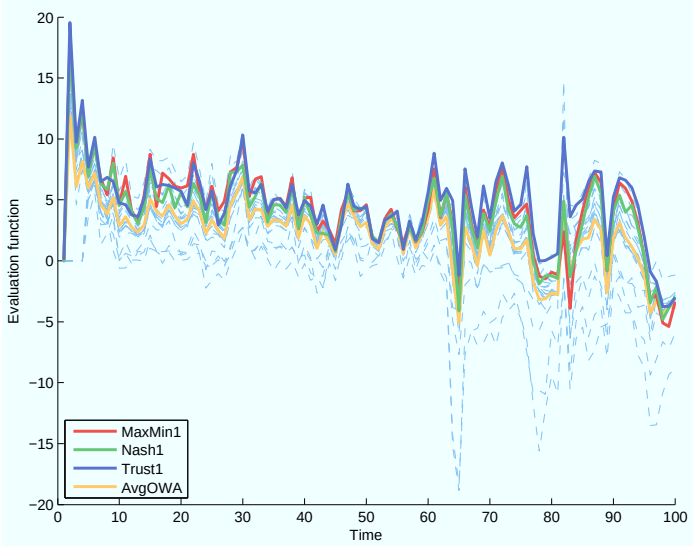


Figure 3.3.1: Performance of solutions on challenge-based attacks over time, using the criteria $\mathcal{E}$ (3.3.7), higher is better, using values α_{ij} and β_i without aggregation.

MaxMin1	Nash1	Trust1	MaxMin5	Nash5	Trust5	AvgOWA	Best
4.09	4.07	4.68	4.16	3.68	4.25	2.24	5.14

Table 3.3: Average value of criteria $\mathcal{E}$ (3.3.7) (higher is better) for challenge-based results

`Trust1`. This is caused by the fact, that the evaluation function does not completely matches the trust experience used in the trust modeling approach.

Furthermore, the values of evaluation score indicate that all solution concepts highly outperform the average strategy `AvgOWA` and provide relatively good results even in comparison to the best possible selection.

Finally, we have to mention that despite using solution concepts with short history provide better results on challenge-based attacks, solutions with longer history ensure more stable results and thereby provide higher robustness against more sophisticated attacks.

3.3.3 Real-world attacks

In the experiment we verify the capability of the system to detect a real-world attack. To measure the quality of detection, we have defined the criterion:

$$\mathcal{E}' = \frac{\bar{\theta}_{all} - \bar{\theta}_x}{\sigma_{all}}, \tag{3.3.8}$$

where the $\bar{\theta}_{all}$ is the arithmetic mean of the anomaly values of the whole observed traffic, $\bar{\theta}_x$ is the arithmetic mean of the flow anomaly values of the measured attacks described above and σ_{all} is the standard deviation of the whole traffic anomaly values. The higher value of the $\mathcal{E}$ is better, as it ensures better separation of the anomalous traffic.To evaluate the equality from

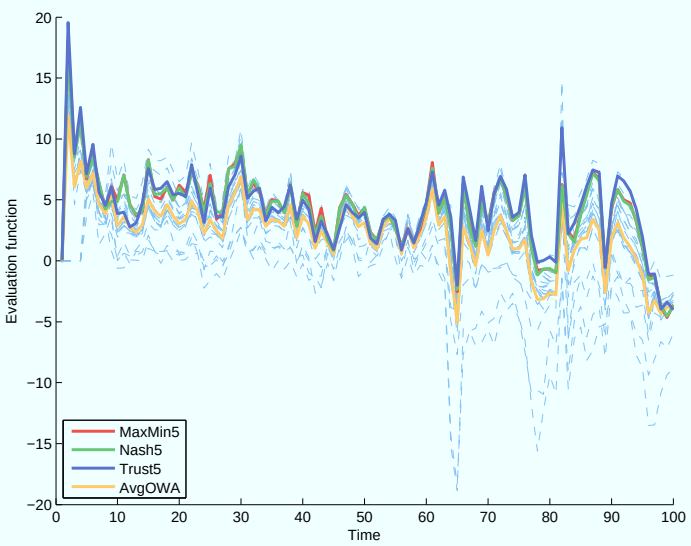


Figure 3.3.2: Performance of solutions on challenge-based attacks over time, using the criteria $\mathcal{E}$ (3.3.7), higher is better, using values $\alpha_{i,j}$ and β_i aggregated over last 5 intervals.

the perspective of the false positives and false negatives, we define another criterion involving these two parameters:

$$\mathcal{E}'' = |\mathrm{FP}| + 3|\mathrm{FN}|, \tag{3.3.9}$$

where the variables |FP| and |FN| represents number of false positive and false negative NetFlows, i.e. the legitimate NetFlows labeled as attack and attacks labeled as legitimate traffic. The definition of this criteria implies that the lower value is better.

Table 3.4 shows how well the individual solution concepts can select the strategies that correctly separate the individual attack types from the legitimate traffic (criteria $\mathcal{E}'$ in Table 3.4a and in Figures 3.3.3 and 3.3.4) and at the same time minimize the false detections (criteria $\mathcal{E}''$ in Table 3.4b and Figures 3.3.5 and 3.3.6). We can clearly see that some of the attacks are more difficult to detect (Horiz. UDP scan), and result in negative values of the criteria, as the known malicious traffic is less anomalous than the average flow. From aggregated results shown in the last line can be seen that the results of all three methods are closer to each other, and two game-theoretical concepts outperform trust-based approach in case of longer time horizon (`MaxMin5`, `Nash5` and `Trust5` columns). This is caused by the fact that the randomized selection of challenges does not match the actual attacks and since the `Trust1` and `Trust5` optimizes the IDS configuration according to these challenges more strictly the selected configurations underperform the configurations selected by the game-theoretical solutions.

It can be assumed that the game theoretical methods may under-perform the trust-based solutions in the situations when the opponent does not behave strategically. The game-theoretical methods will be penalized by their security features, as they optimize for robustness rather than for precise match of the single optimization criteria. This (rather pessimistic) assumption does not hold. While the game theoretical methods score slightly less, the actual lost utility is surprisingly low, lower than the difference due to the use of longer history in directly optimizing trust function. The game theoretical methods outperform the trust-based optimization because they are less sensitive to challenge selection effects. The difference in performance is due to the randomization and inefficiencies related to second-order strategic behavior.

	MaxMin1	Nash1	Trust1	MaxMin5	Nash5	Trust5	AvgOWA	Best
SSH brute force	4.03	4.05	3.76	4.07	3.98	3.83	3.73	6.84
Vertical TCP scan with OS detection	0.86	0.81	0.79	0.94	0.95	0.90	0.79	1.47
Vertical UDP scan	2.44	2.26	2.43	2.40	2.37	2.37	2.20	2.95
Horizontal TCP scan for SSH service	0.65	-0.08	1.26	0.38	0.34	0.10	0.02	3.29
Horizontal UDP scan for DNS service	-1.49	-1.31	-1.48	-1.41	-1.41	-2.09	-1.77	0.23
Horizontal ICMP ping scan	4.30	4.27	4.78	4.79	4.84	4.37	3.89	8.62
Horizontal TCP scan for all services	3.10	3.11	3.01	2.99	3.00	2.98	3.21	4.51
Average	1.98	1.87	2.08	2.02	2.01	1.78	1.72	3.99

(a) Results obtained on real world attacks using criteria $\mathcal{E}'$ 3.3.8 (higher is better).

	MaxMin1	Nash1	Trust1	MaxMin5	Nash5	Trust5	AvgOWA	Best
SSH brute force	23.56	23.31	21.86	20.46	20.05	21.01	23.35	19.24
Vertical TCP scan with OS detection	23.10	22.93	23.47	21.97	21.79	23.32	23.80	21.20
Vertical UDP scan	22.54	22.47	22.96	22.22	22.09	24.91	23.71	19.90
Horizontal TCP scan for SSH service	28.23	28.65	30.45	27.70	27.58	30.83	31.20	27.53
Horizontal UDP scan for DNS service	25.00	24.93	27.45	24.85	24.40	27.25	27.35	23.93
Horizontal ICMP ping scan	18.33	17.93	19.33	17.85	17.48	19.43	19.15	16.35
Horizontal TCP scan for all services	38.70	37.33	36.73	37.71	37.74	37.85	36.19	26.42
Average	25.63	25.36	26.04	24.68	24.44	26.37	26.39	22.08

(b) Results obtained on real world attacks using criteria $\mathcal{E}''$ 3.3.9 (lower is better).

Table 3.4: Results obtained on real world attacks using criteria 3.3.8 and 3.3.9.

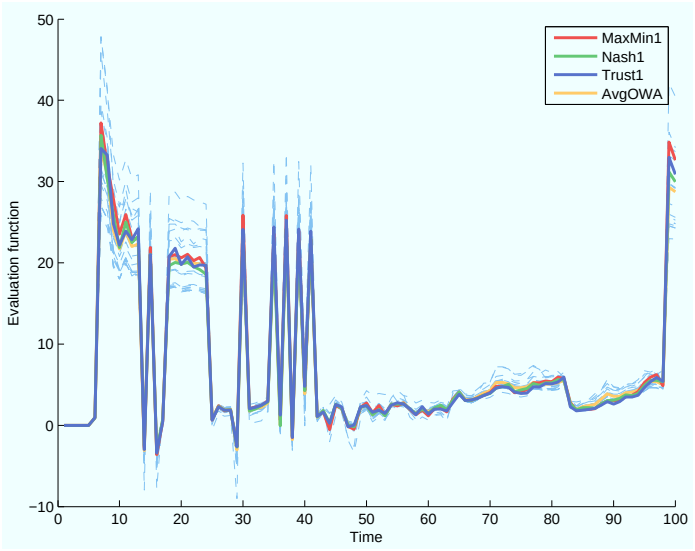


Figure 3.3.3: Performance of solutions on real attacks over time, using the criteria $\mathcal{E}'$ (3.3.8), higher is better, using actual values α_{ij} and β_i without aggregation.

3.3.4 Solution stability

In the last section we will briefly present the stability of each solution concept. We will compare both game-theoretical solution concepts noted in previous sections. For each solution concept we will present two graphs – one when there is used no history and one where values α_{ij} and β_i are aggregated over last 5 intervals. Each graph shows the normalized weights in mixed strategies in time. Therefore, if there is one IDS configuration that is preferred by the solution concept for given period of time, it will appear as large area in the graph as it has large weight in the mixed strategy.

At first, we will discuss the *MaxMin*. As Figures 3.3.7 and 3.3.8 show, the MaxMin solution concept provides relatively stable solution in both versions, with (`MaxMin5`) and without (`MaxMin1`) history. This indicates that there is small number of configurations that outperform the others and as such are frequently selected by this solution concept.

More interesting situation appears in the case of Nash equilibria (Figures 3.3.9 and 3.3.10). In the case that this solution concept does not use history (`Nash1`), the selection of the best aggregation function is more stochastic and, as it was confirmed in Section 3.3.3, this fact affects negatively the final results. This phenomena is caused by the fact that without using history the utility function has a large number of mixed equilibria and the solver has to randomly select the best defender's strategy. In the case when the history is used (`Nash5`), this phenomena does not appear and the solutions are relatively stable.

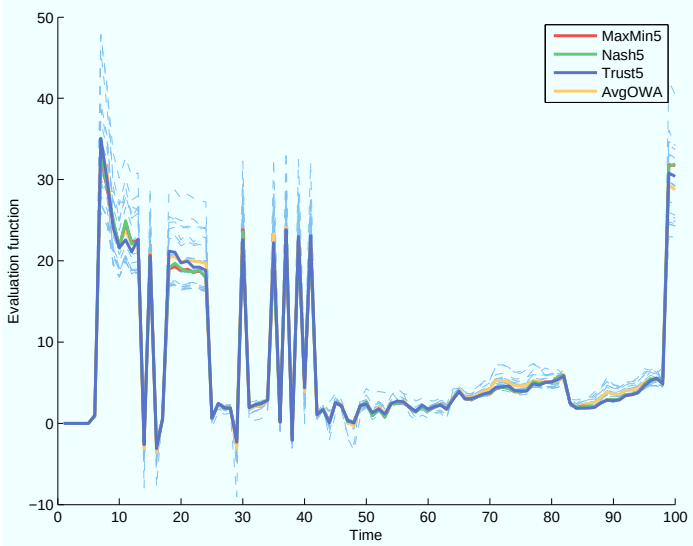


Figure 3.3.4: Performance of solutions on real attacks over time, using the criteria $\mathcal{E}'$ (3.3.8), higher is better using values α_{ij} and β_i aggregated over last 5 intervals.

3.4 Conclusions

This chapter addressed several important research issues related to the use of game-theoretic approaches for strategic adaptation of multi-agent intrusion detection system. In Section 3.1.1, we presented a practical game theoretical model of the IDS problem and discussed its integration with a real-world intrusion detection system. The use of such mechanism shall improve system robustness w.r.t very advanced attacks based on adversarial machine learning approaches. To our knowledge, we have proposed the first implementation of game-theoretical principles suitable for real-time reconfiguration of an IDS system and evaluated the influence the game-theoretical mechanisms to the performance of commercial IDS system on real-world attacks.

The experiments performed with a simplified version of this IDS clearly showed that the cost of game-theoretical formulation use is very low, and does not adversely affect the effectiveness of the adaptation process. In particular, our results suggest that the max-min method provides results that outperform the original trust-based adaptation with very consistent results. Moreover, it does not require an explicit model of opponent's utility function and is computationally trivial, and as such it makes an appropriate first choice for future implementations.

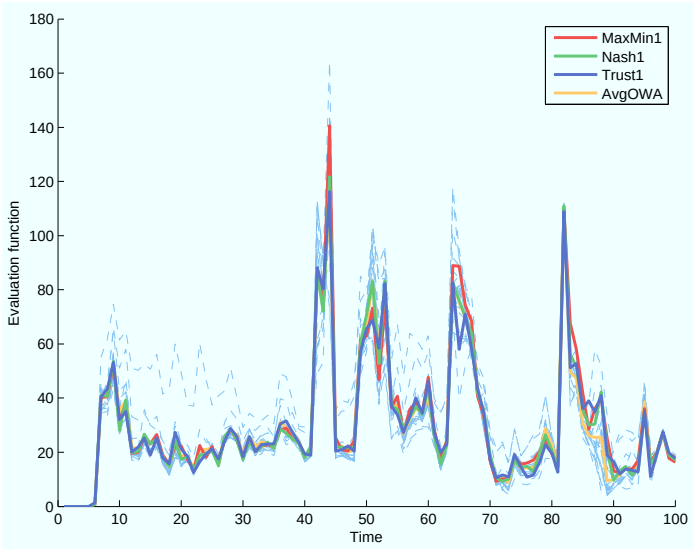


Figure 3.3.5: Performance of solutions on real attacks over time, using the criteria $\mathcal{E}''$ (3.3.9), lower is better, using actual values $\alpha_{i,j}$ and β_i without aggregation.

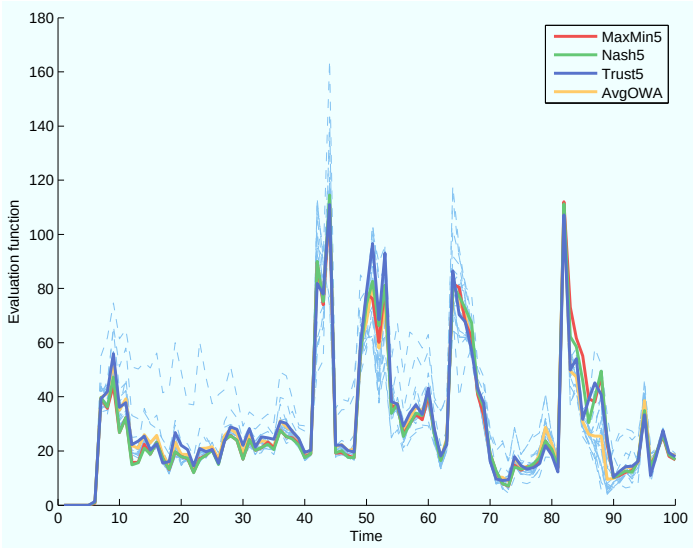


Figure 3.3.6: Performance of solutions on real attacks over time, using the criteria $\mathcal{E}''$ (3.3.9), lower is better, using values $\alpha_{i,j}$ and β_i aggregated over last 5 intervals.

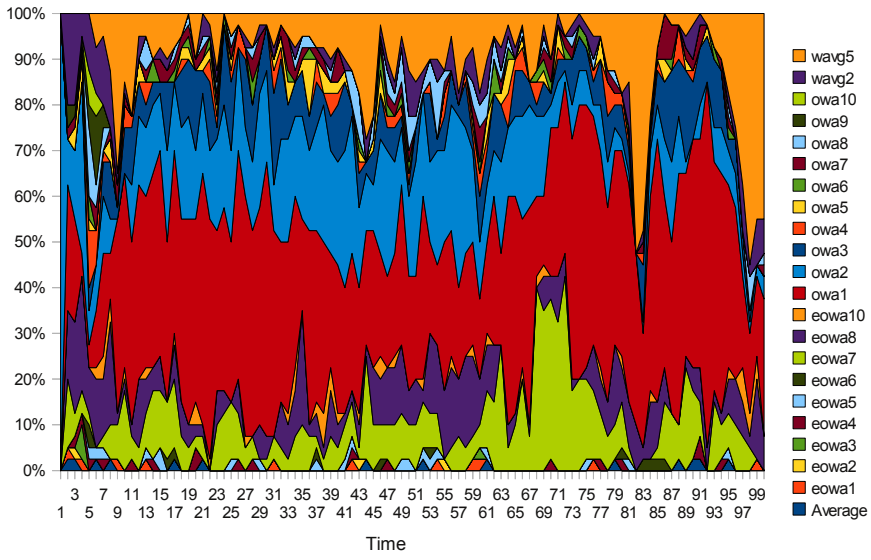


Figure 3.3.7: Max-min rule without history (`MaxMin1`)

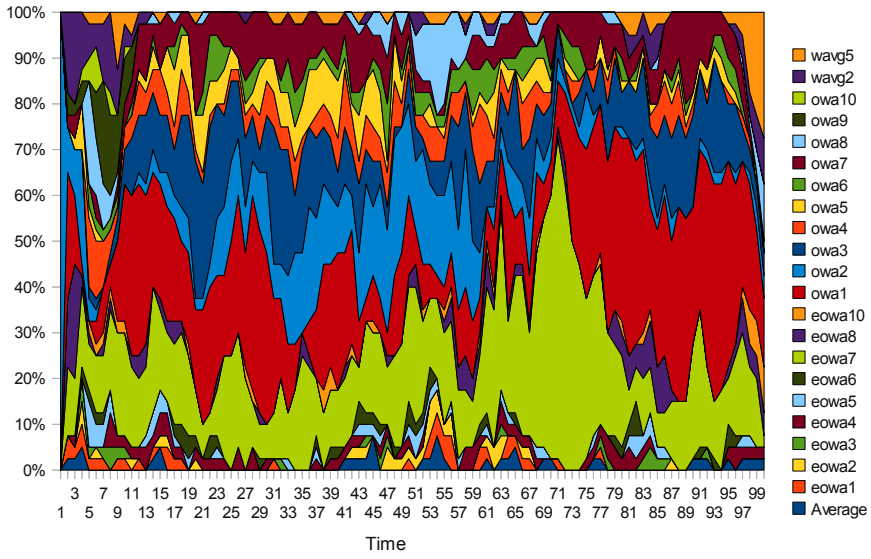


Figure 3.3.8: Max-min rule using history (`MaxMin5`)

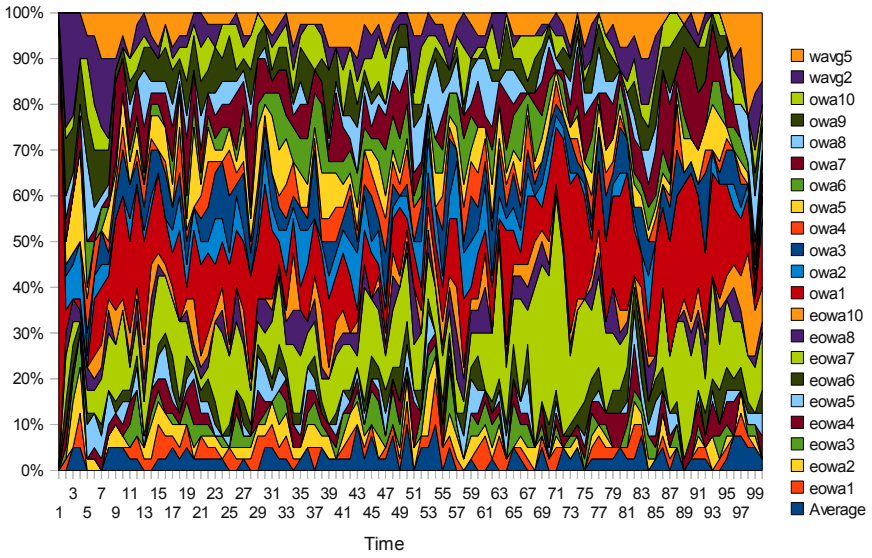


Figure 3.3.9: Nash equilibrium without history (`Nash1`)

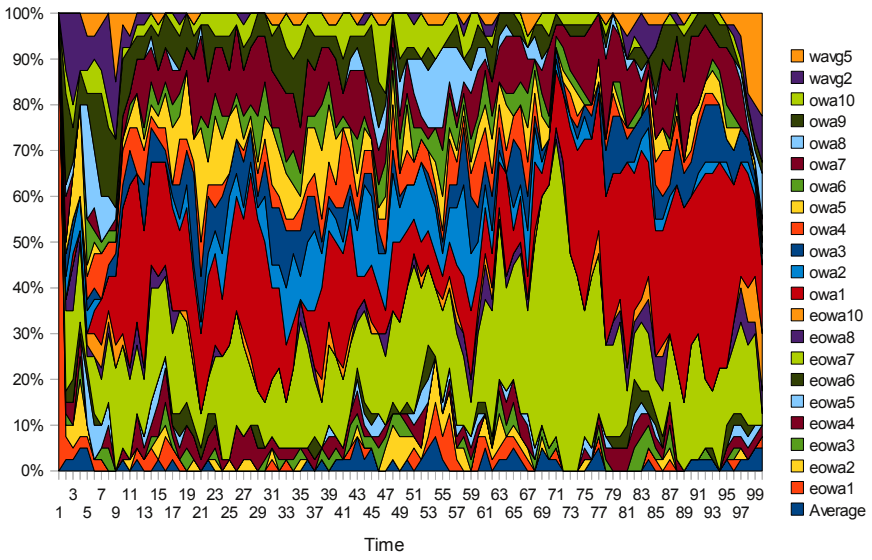


Figure 3.3.10: Nash equilibrium using history (`Nash5`)

Chapter 4

Simulation of legitimate behavior

This chapter discusses an approach to generate realistic network traffic suitable for tuning/evaluation of an intrusion detection system (IDS) designed as an replacement of the fixed database of static challenges. In this chapter we describe a model that simulates behavior of legitimate user and generate data in high-level NetFlow [22] format popular in security community and especially in anomaly-detection-based IDS systems [117, 118, 119]. Simulation of the network traffic in the form of NetFlows allows us to construct the data directly in memory which removes a large amount of computational complexity typically involved in the simulation approaches [17, 19] and make our approach well-suited for online adaptation. Note that similar approach was proposed by Sperotto et al. [120] for online simulation of SSH attacks, however the approach proposed in this chapter is design to simulate much broader spectrum of network traffic.

In order to capture users' behaviors we assume that their high-level behavioral patterns are stable for long period of time (weeks, or even months) as users typically come to work at the same time, visit the same or similar web sites, use the same e-mail server, etc. The main reason is, that in contrast to the malware, legitimate users have no reason to intentionally change their behavior to avoid detection. This aspect allows us to use the same models for relatively long period of time without retraining. At the same time, we assume that the low-level behavioral patterns of legitimate users (individual network connections) are much more random compared to the malware. Instead of following pre-scripted scenarios like malware does, legitimate users behave stochastically, i.e. they visit similar sites but not always the same, they do not follow precisely the same order of actions (first checking e-mail, then opening website, then SSH connection), etc. Both these assumptions motivate to simulate the users' behaviors stochastically which was successfully adopted in previous work [50, 19]. The traffic generated with well-defined stochastic models exhibit stable high-level behavioral patterns but on the low-level, the generated traffic is randomized.

In this chapter we propose three techniques with different level of complexity. The first two use simple statistical models that are easy to implement but not sophisticated enough to capture various aspects of users' behaviors, such as changes in the users' behaviors during the day, dependency between NetFlow features (e.g. packet-bytes ratio), etc. The third one addresses these deficiencies and generates the users' traffic in a way that state-of-the-art detection algorithms are not able to distinguish it from real traffic.[1]

[1] Work presented in this chapter was published in [30].

Field name	Description
Starting time	Time stamp of the first packet of the flow
Duration	Length of the flow
Protocol	TCP, UDP, ICMP, etc.
Source IP	IP address of the source
Source port	
Destination IP	IP address of the target
Destination port	
Bytes	Number of bytes transferred in the flow
Packets	Number of packets transferred in the flow
TCP flags	Not used for non-TCP protocols

Table 4.1: List of NetFlow fields.

4.1 Basic models

The design of a simulation model needs to address the common trade-off between complexity and performance. Before introducing the advanced model in the next section we first discuss two simpler approaches. We show that simplifying assumptions about NetFlow traffic allow for models with low complexity. At the same time we will show where simplified models fail to capture higher-order aspects of users' behaviors such as dependencies between individual NetFlow features. The identified flaws then inspire the definition of the improved model presented in the next section.

4.1.1 Random sampling

The simplest approach to simulation of behavior of a single user is to generate individual NetFlow fields (as listed in Table 4.1) independently[2]. Such approach does not take into account any realistic properties of the NetFlow fields (e.g. distribution of bytes, distribution of source ports, etc.), relation between fields of the NetFlow (e.g. bytes/packet ratio) or relations between individual NetFlows (e.g. request/responses relations). The only condition that has to be satisfied is the validity of NetFlow, i.e. all fields have to be in their allowed ranges and the following condition must be met

$$0 < \text{number of bytes} \leq \text{number of packets} \times 65535. \tag{4.1.1}$$

As such, the only parameter of this algorithm that has to be specified in advance is the IP address of the simulated user. The random sampling generates all but three individual NetFlow features randomly with respect to the condition of validity of the generated NetFlow. The exceptions are the starting time and source and destination IP addresses.

Instead of generating the starting time directly we sample the user's thinking time, the time delay between two consequential NetFlows, uniformly between $0s$ and $10s$. The new starting time is then computed as sum of the last starting time and the current thinking time.

Similarly to the thinking time, the source and destination IP addresses are not generated directly. Instead, we randomly choose whether a given flow is request or response. If the flow is generated as request the source IP field is set the IP address of the simulated user specified

[2]Similar approach was proposed by Sommers et al. [121] and technological solutions such as BreakingPoint [122]

Feature name	Description
Client's thinking time	Time difference between two consequential client's requests
Client port	Source port of the request
Request bytes	Number of bytes in request
Request packets	Number of packets in request
Request protocol	TCP, UDP, ICMP, etc.
Request flags	TCP flags in request
Request length	Duration of request
Server thinking time	Time difference between request and response
Server IP	IP address of server
Server port	Port number of service used by user
Response bytes	Number of bytes in response
Response packets	Number of packets in response
Response length	Duration of response
Response flags	TCP flags in response
Has response	Is there corresponding response to the request?

Table 4.2: List of NetFlow features to be modeled in order to create NetFlow data that correctly reflect requests and responses. Note that TCP flags for request and response are empty for all non-TCP NetFlows.

in advance and the destination IP is chosen randomly. In the case of response it is the other way around.

The main benefit of this algorithm is its independence on any training data or manual tuning, as the only parameter that has to be set in advance is the user's IP address. However at the same time, the complete randomness is the main disadvantage since it can generate completely unrealistic data. Therefore, we use this approach only for syntax testing and as a baseline for the comparison to more sophisticated methods.

4.1.2 Sampling with independent intra-flow relations–marginal model

The marginal model adopts more realistic approach than completely random sampling. It uses training data of a single user in order to train the statistical model of individual NetFlow features (e.g. distribution of bytes or distribution of user's thinking times, etc). Unlike the random sampling, the marginal model considers NetFlows in request/response pairs[3]. Therefore it is able to partially model inter-flow relations (the relation between individual flows), namely the request/response relations, but not the sequential character of the user's behavior. For example, it is able to correctly model the HTTP request/response pairs but not the download of the whole Google home page. Next, the marginal model assumes that the modeled features are independent and thus it does not take into account intra-flow relations (e.g. bytes/packets ratio, etc.). This assumption is a limitation that affects the variance of the internal model and can cause serious sampling artifacts (see Figure 4.1.1). The complete list of modeled features is listed in Table 4.2.

[3]The marginal model focuses on the modeling of user's behavior and thus we consider outgoing flow as *request* and incoming flow as *response*.

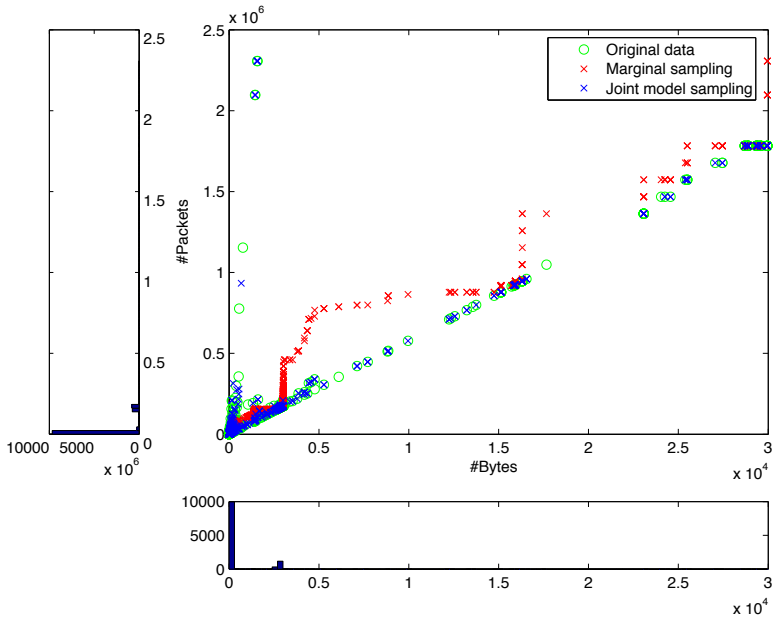


Figure 4.1.1: Artifacts of marginal sampling in real-life example. The figure shows that marginal sampling (red crosses) is not able to mimic the data correctly and thus creates serious sampling artifacts—data that did not appear in the original data—whereas sampling from time variant joint probability model (to be introduced in next section) respects the dependency between features and generates the data correctly.

To train the marginal model, the training data are preprocessed to pair the requests with the corresponding responses. We assume that the user behaves only as a client and thus every NetFlow with user's IP address as source IP is considered as a request. The corresponding response is matched as NetFlow with following properties:

$$\begin{aligned} \text{source IP}_{\text{response}} &= \text{destination IP}_{\text{request}}, \\ \text{source port}_{\text{response}} &= \text{destination port}_{\text{request}}, \\ \text{destination IP}_{\text{response}} &= \text{user's IP}, \\ \text{destination port}_{\text{response}} &= \text{source port}_{\text{response}}, \\ \text{protocol}_{\text{response}} &= \text{protocol}_{\text{request}}. \end{aligned} \tag{4.1.2}$$

Note that there is a maximal delay τ between request and response in order to avoid incorrectly paired flows, current set to 2 seconds. The extracted request/response pairs are used to estimate the distributions of all individual features of the model (Table 4.2) using non-parametric estimates (histogram for continuous features or relative frequencies for categorical features) as no well-known parametric distribution correctly describes these features [50]. Note that in the experimental evaluation individual features were estimated from one-week-long sample of data which prove to be sufficient, as larger datasets did not provide significantly better estimates.

Once model training is finished, request/response pairs are sampled as follows. At first, we randomly choose whether there will be a response or the generated pair will contain only request (the feature *Has response*). Next, we sample values of individual NetFlow fields from distributions of corresponding features estimated from the training data, except for starting

time of the request and response. Instead of sampling starting time of request and response directly, we sample user's and server's thinking times (both from corresponding distributions estimated from training data). The starting time of the request is then computed as a sum of the last starting time and user's thinking time, and the starting time of the response is computed as sum of starting time of the request and thinking time of the server. In the last step is the source address in the request (and destination IP in the response) set to the user's IP address (parameter of the algorithm) and the destination address in the request (source address in the response) is sampled from the distribution estimated from the training data.

4.2 Time variant joint probability model

In Section 4.1.2 we have described a simulation technique that uses simple statistical model but misses more complicated aspects of the user's behavior. This leads to serious sampling artifacts such as

- single connection transfers 60GB in 2 seconds,
- user activity remains constant during the day and night,
- HTTP request precedes corresponding DNS request.

The first issue is caused by the fact that marginal model does not respect *relations between individual features* (see Figure 4.1.1).

Next problem is caused by the fact that *user's behavior changes during the day.* Usually during the night there is no network traffic generated by the user's machine or the volume of the traffic is very low (only the automated behavior of the machine, e.g. periodic updates). However, during the working hours its network activity increases rapidly and fluctuates during the whole day. For example during the lunch break there is a significant drop in the volume of the network traffic followed by a spike when users return to work. Note that the profile of the user's behavior changes through the day as well, since user uses different services at different time of day. The marginal model described in previous section does not reflect such changes and this has negative effects to the quality of the generated data.

The third problem relates to the *sequential character of the network traffic triggered by user's actions.* The typical example is the opening of a web site as the web browser has to resolve the domain name of the HTTP server before it can establish TCP connection with the server. This will appear as request to the DNS server that resolves the domain name of the web site followed by a number of different connections to port 80 over TCP protocol to the HTTP server. This example illustrates that user's network traffic exhibits sequential behavior which is not reflected by the marginal model.

Comparison of all three models and various characteristics captured by these models is summarized in Table 4.3.

4.2.1 Model formalism

In order to address all deficiencies and problems described in previous section, we propose the *time variant joint probability model* (referred as *joint* for short). It assumes following generative process that generates user's traffic: after some time of inactivity (user's thinking time T) user selects specific service s (DNS, HTTP, SSH, etc.) defined by a combination of server port and protocol. For this service, he selects a remote server specified with destination IP dIP, initiate connection, and, if successful, performs communication.

	Random	Marginal	Joint
Properties of fields of NetFlows (e.g. distribution of bytes, source ports, etc.)	-	✓	✓
Intra-flow relations (e.g. Packet/Bytes ration, etc.)	-	-	✓
Inter-flow relations (e.g. request/response ration)	-	✓	✓
Changes in the user's behavior	-	-	✓
Sequential character of the user's behavior	-	-	✓

Table 4.3: Capabilities of presented models. The table summarizes capabilities of random sampling (*Random*), marginal model (*Marginal*) and time variant joint probability model (*Joint*). It shows whether given property of the traffic can be (✓), cannot be (-) captured by given model.

Algorithm 4.1 Sampling of the NetFlow data

procedure sampleFlow(length) ▷ Sampling single flow
  $\mathcal{F} \leftarrow \emptyset$
  $t \leftarrow 0$ ▷ Set current time to 0
  repeat
    $T \leftarrow thinkingTime(t)$ ▷ Sample thinking time
    $s \leftarrow p(s|t)$ ▷ Sample service
    $\text{dIP} \leftarrow p(\text{dIP}|s,t)$ ▷ Sample target
    $cPort \leftarrow p(\text{cPort}|s,t)$ ▷ Sample client port
    $x_s \leftarrow p(x_s|\text{dIP},s,t)$ ▷ Sample remaining features
    $\text{flow} \leftarrow t, s, \text{dIP}, \text{cPort}, x_s$ ▷ Build flow
    $\mathcal{F} \leftarrow \mathcal{F} \cup \{\text{flow}\}$
    $t \leftarrow t + T$ ▷ Increment current time
  until $t \leq$ length
  return $\mathcal{F}$
end procedure

Feature name	Description
Request bytes	Number of bytes in request
Request packets	Number of packets in request
Request protocol	TCP, UDP, ICMP, etc.
Request flags	TCP flags in request
Request length	Duration of request
Server thinking time	Time difference between request and response
Response bytes	Number of bytes in response
Response packets	Number of packets in response
Response length	Duration of response
Response flags	TCP flags in response
Has response	Is there corresponding response to the request?

Table 4.4: Set of features that describe the behavior of a service. Note that TCP flags for request and response are empty for all non-TCP connections.

To capture such interaction we adopt request/response pairing used in marginal model. Additionally, we adopt following assumptions:

1. the whole generative process is parametrized by *day time* (hours of the day), further denoted as t, to address the problem of the variability of user's behavior during the day,
2. the thinking time (T) depends only on the day time t, i.e. no other aspects of user's behavior affect his thinking time,
3. the client port (*cPort*) depends only on the service and day time, as source ports for the outgoing connections are assigned by the operating system without any user' interaction. Most operating systems simply increment the last used port until the range is depleted and thus for different daytime different range of ephemeral ports is used. The dependency on service is caused mainly by long persistent connections that are split by the NetFlow probe into several NetFlow records. All these flows have the same client port and in the statistics, it appears as the service prefers single source port.

The generative process described above is then formalized as probabilistic generative model with following probability distribution function (PDF)

$$p(x|t) = p(s|t) \cdot p(\text{dIP}|s,t) \cdot p(\text{cPort}|s,t) \cdot p(x'|\text{dIP},s,t) \cdot p(T|t), \tag{4.2.1}$$

where t represents the day time, s represents service, dIP represents destination IP, cPort represents client port and x' represents vector of all remaining features (see Table 4.4). Generating new traffic (summarized in Algorithm 4.1) is then implemented as sampling from this distribution.

The first factor of the complete PDF, $p(s|t)$, corresponds to the probability of a user using service s (the server port and protocol tuple) in given day time t. It is modeled as *first order discrete-time Markov chain* [123] which naturally captures the sequential character of the user's behavior (e.g. the example with the HTTP communication described above). In the Markov chain, services are represented as states and transition probabilities are estimated from the data. Generating of a new service is then sampling from the Markov chain for specific day time t given the knowledge of service in previous time step.

The factor $p(\text{dIP}|s,t)$ represents the probability of connection to service s in day time t that runs on destination IP dIP. It is modeled simply as histogram of all destination IP (for specific service in given day time) estimated from the training data.

The factor $p(\text{cPort}|s,t)$ captures the probability of operating system assigning the client port cPort when user opens connection to service s in day time t modeled as histogram of client ports estimated from training data. This model is able to capture the long term connections to a service that appear in the data as different NetFlows with the same client port (connection to e-mail server, long term SSH connection, etc.) as well as different strategies used by operating system to assign the ephemeral source ports to outgoing connections.

The factor $p(x'|\text{dIP},s,t)$ describes the joint distribution of all remaining features for service s running on IP dIP in day time t. It is modeled by multivariate histogram to capture the behavior of the remote service running on given server.

The last factor, $p(T|t)$ describes the user's thinking time in different day times. In contrast to the previous factors, this one does not model the distributions of thinking times directly using e.g. histogram. The main reason is that it results in unrealistic data as the histogram does not capture correctly the time intervals when the activity of the user is very low. In the next section we discuss approach specifically designed to address this issue.

Sampling of user's thinking time

The marginal model, we have proposed in Section 4.1.2, models the starting time of the request as a sum of the starting time of the previous request, and the thinking time of the client. However, such approach cannot be directly adopted in the joint model due to the fact that if we parametrize the thinking time with the daytime, it does not estimate the thinking time for time intervals when the activity of the user is very low. The typical example is the night time, when the last user's activity appears in the evening and we do not have enough data to estimate the thinking times.

To avoid this issue, we do not estimate the thinking time directly. Instead we model the numbers of request n generated by user in given time interval. The thinking time is then computed as follows

$$T = \frac{L}{n} \tag{4.2.2}$$

where L is the duration of usual time interval in seconds. In our setup we set $L = 300s$, the usual length of a batch in anomaly-detection-based IDS systems [124, 125].

To stabilize the estimates of the numbers of requests $[n_1, \ldots, n_i, \ldots]$ in individual time windows, we smooth the data with sliding window as follows

$$n_t = \frac{1}{l} \sum_{i=t-\frac{l}{2}}^{t+\frac{l}{2}} n_i, \tag{4.2.3}$$

where l is the width of the sliding window controlling the smoothness of the estimate. If the window is too long, the value does not follow the trends in the data, and if it is too short the estimated value is too noisy. In the experimental evaluation we use $l = 8$ as it sufficiently smooths the estimates but still follows the dynamic trends in the data.

Next, we divide the list of the number of requests $N = \{n_1, \ldots, n_i, \ldots\}$ into one-day long sets forming matrix N' defined in Equation (4.2.4). For every time interval $t \in \{1, \ldots, 288\}$, that is represented by a row in matrix N', we have k samples where k is the length of the training data in days.

$$N' = \underbrace{\begin{pmatrix} n_1 & n_{289} & \cdots \\ n_2 & n_{290} & \cdots \\ \vdots & \vdots & \\ n_{288} & n_{576} & \cdots \end{pmatrix}}_{k} \tag{4.2.4}$$

We estimate the distribution of number of requests for interval t with histogram $\mathcal{H}_t$ with non-linearly distributed bins $[1, 11), [11, 21), [21, 41), [41, 81), [81, 201), [201, \infty)$ as the number of requests in interval t roughly follows exponential distribution approximated by the bins. Furthermore, for every bin of the histogram $\mathcal{H}_t$ we define distribution of values. If the number of samples that fit into this bin is 1, we assume uniform distribution of values in this particular bin. If there are more samples, we assume normal distribution with parameters estimated from the samples that fit into the bin. This setup helps us to overcome the lack of data as we have only k samples.

The sampling procedure of the thinking time for given time interval is listed in Algorithm 4.2.

Algorithm 4.2 Sampling of thinking time

procedure THINKINGTIME(t) ▷ Sampling thinking time for time interval t
  $b \sim \mathcal{H}_t$ ▷ Select bin with respect to distribution $\mathcal{H}_t$
  if $|b| = 1$ **then** ▷ If there is only one training sample that fits in the bin b
    $n_t \sim \mathcal{U}(b_a, b_b)$ ▷ Sample from uniform dist. defined by boundaries b_a and b_b of the bin b
  else
    $n_t \sim \mathcal{N}(\mu, \sigma)$ ▷ Sample from normal distribution with parameters μ and σ estimated from training samples
  end if
  $T = L/n_t$ ▷ Compute thinking time, L is the length of the time interval (in our case $L = 300s$)
  return T
end procedure

4.3 Experimental evaluation

In this section we evaluate the quality of the data generated by the sampling approaches defined in Sections 4.1 and 4.2. Our assumption is that if the simulated traffic correctly mimics the profile of the real traffic, the IDS will trigger the same response to the simulated and real data. Since the aim of the presented approaches is to evaluate an anomaly-detection-based IDS, we measure the response of the IDS as a distribution of anomaly scores and the difference between responses to the simulated and real data is measured as distance between corresponding distributions of anomaly score. In this text we quantify the distance with *Jensen-Shannon divergence* (D_{JS}) [126], a symmetric version of Kullback–Leibler divergence (D_{KL}). It is formally defined as

$$D_{\mathrm{JS}}\left(\phi_{\mathrm{real}}||\phi_{\mathrm{sim}}\right) = \frac{1}{2} D_{\mathrm{KL}}\left(\phi_{\mathrm{real}}||\phi\right) + \frac{1}{2} D_{\mathrm{KL}}\left(\phi_{\mathrm{sim}}||\phi\right),$$
$$\phi = \frac{1}{2}\left(\phi_{\mathrm{real}} + \phi_{\mathrm{sim}}\right),$$

where ϕ_{real} and ϕ_{sim} represent distributions of anomaly scores for real and simulated data respectively.

4.3.1 Selected anomaly detection algorithms

In order to evaluate the data under different conditions, we have implemented various types of anomaly detection algorithms based on different detection paradigms.

Algorithms proposed by Pevný et al. [127] and Lakhina et al. [128, 129] use the principal component analysis to detect anomalies in the traffic. However, there are several key differences between these methods. First difference is in the features that are used for the definition of the model of the individual detectors. Second difference is the measure used for assigning the anomaly value (Lakhina proposes to use *reconstruction error* and Pevný uses *mahalanobis distance in sub-spaces*). Note that we have implemented four different versions of algorithm proposed by Pevný denoted in the results as *Pevný-f-dIP*, *Pevný-f-sIP*, *Pevný-$f^{\perp}$-dIP* and

Pevný-$f^{\perp}$-sIP (all described in [127]), and two versions of Lakhina's algorithm *Lak.Vol.-sIP* and *Lak.Vol.-dIP* where algorithms with suffix *"-sIP"* models the traffic with respect to a source IP and versions with suffix *"-dIP"* models the traffic with respect to a destination IP.

Second type of algorithm we have used in the evaluation is a modified version of *Minnesota Intrusion Detection System–MINDS* [130]. It uses an internal model of the network traffic but unlike the algorithms proposed by Pevný and Lakhina it does not use PCA but measures the difference between last and current time window. In order to overcome the performance issues we have modified the algorithm from the originally proposed version. The modifications are described in [131].

The last group of algorithms does not use any internal model of the network traffic. The method originally published by Kuai Xu *et. al.* [132] uses a basic assumption that all network traffic can be classified into several categories using a set of static thresholds. In addition to the original algorithm (denoted as *Xu-sIP* in our evaluation) we have implemented modified version (denoted as *Xu-dIP*) that uses complementary features relating to the destination IP.

4.3.2 Training and evaluation data

To evaluate the quality of the simulated traffic we used data recorded on mid-size university campus during one week in April 2013 (further denoted as $\mathcal{D}_{\text{orig}}$). From the recorded data we selected a set of full-time employees with various user profiles (developers, scientists, managers and administrative staff) and sufficient amount of NetFlow data. We separated their traffic based on their IP addresses and use it as training data for two models defined in Sections 4.1.2 and 4.2. The remaining traffic formed the reduced dataset $\mathcal{D}_{\text{red}}$ which was used as background that was mixed with the simulated traffic.

The evaluation of quality of the generated data was separated into two stages. During the first stage we processed the original data $\mathcal{D}_{\text{orig}}$ by all anomaly detection methods and estimated the distribution of anomaly values for selected users for every detection method separately.

In the second stage we simulated the network traffic using three approaches proposed in this chapter: (1) the random sampling (referred as *Random*, see Section 4.1.1), (2) sampling with marginal model (*Marginal*, see Section 4.1.2) and (3) sampling with time variant joint probability model (*Joint*, see Section 4.2). The generated data were mixed with the reduced dataset $\mathcal{D}_{\text{red}}$ and this mix was again processed by all anomaly detection algorithms. Again, we have estimated the distributions of the anomaly values of the simulated traffic for individual anomaly detection algorithms and measure the distance between the distribution of anomaly score for the simulated traffic and the distribution of anomaly score for real traffic estimated in the first step. This process was repeated 20 times and using Kruskal-Wallis statistical test on significance level $\alpha = 0.05$ we verified that the results for the simulation approaches are significantly different. Therefore we can compare individual simulation approaches using only their average values of D_{JS} estimated from the 20 runs.

In order to evaluate how realistic the simulated data are, we setup a baseline (further referred as *Real*) which estimated the self-similarity of the real traffic. We split the original data of selected users in two subsets according to the time and evaluated the distance D_{JS} between distributions of anomaly scores for individual anomaly detection methods estimated on these two subsets. The assumption is that if the distance between two subsets of real traffic is similar to the distance between simulated and real traffic, we consider the simulated traffic realistic enough to be used for tuning/evaluation of an IDS.

Detection alg.	Joint	Marginal	Random	Real
Pevný-f-dIP [127]	0.0321	0.0483	0.5427	0.0769
Pevný-f-sIP [127]	0.0320	0.0464	0.5573	0.0674
Pevný-$f^{\perp}$-dIP [127]	0.0124	0.0214	0.4237	0.0204
Pevný-$f^{\perp}$-sIP [127]	0.0088	0.0216	0.3942	0.0198
Lak.Ent. [129]	0.0472	0.1111	0.1889	0.0549
Lak.Vol.-sIP [128]	0.0353	0.1132	0.1889	0.0118
Lak.Vol.-dIP [128]	0.0433	0.1124	0.1874	0.0152
MINDS [130]	0.0292	0.0976	0.2399	0.0516
Xu-sIP [132]	0.0301	0.0371	0.0286	0.0078
Xu-dIP [132]	0.0421	0.0815	0.1704	0.0354
Average	0.0313	0.0691	0.2922	0.0361

Table 4.5: Jensen-Shannon divergence (D_{JS}) between distribution of anomaly values of real and simulated traffic (lower value is better).

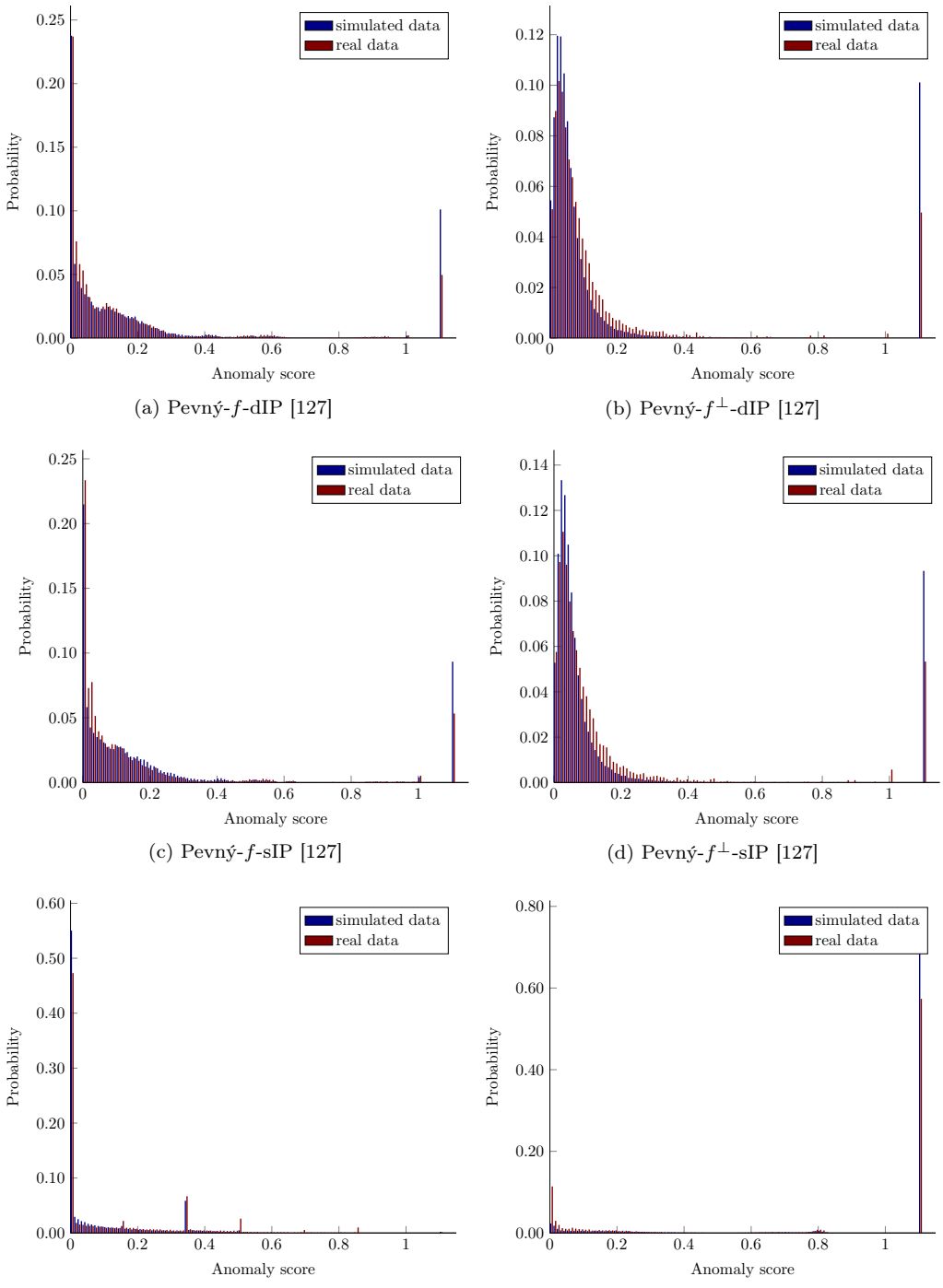
4.3.3 Quality of the generated data

The results summarized in Table 4.5 prove that the *random* sampling generates the least realistic data. The value of D_{JS} is by order of magnitude larger compared to the two remaining models. This confirms the expectations that the random sampling can be used only for syntax testing.

The second approach, the sampling with *marginal model*, provides significantly better results compared to the random sampling. The results show that correct estimation of the marginal distribution of individual features significantly improves the results. However, the most advanced approach, the *joint model*, provides the results more than 2× better than marginal model. This indicates that, assumptions (1) *all inter-flow features are independent* and (2) *user's behavior does not depend on the day time* clearly do not hold.

The last column in Table 4.5 provides the comparison with the baseline that captures the internal variability of the real network data. We can see, that for some detection algorithms (e.g. Pevný-f-dIP, Pevný-f-sIP, Lak.Ent., etc.) the internal variability significantly exceeds the distance between network traffic simulated with joint model and the real traffic. This is caused by the fact that these detection algorithm adapt their internal model to the current state of the network. Therefore even insertion of the same network traffic in two different time windows does not necessarily produce the same response. In the case of the Xu-sIP and Xu-dIP detection methods that do not have any internal model, the internal variance of the real traffic is lower than distance between simulated and real traffic. However, on average, the distance between simulated and real traffic is comparable to the internal variance of the real traffic. This indicates that the joint model realistically simulates the network traffic.

This result is further confirmed in Figures 4.3.1 and 4.3.2 that visualize the distributions of anomaly value of real data and data simulated by the joint model. Note that anomaly score > 1 represents the flows where the particular detector provided no results (due to its limitations). These figures show that our model generates traffic that triggers response of anomaly detection algorithms practically indistinguishable from the response to real traffic.



(a) Pevný-f-dIP [127]

(b) Pevný-$f^{\perp}$-dIP [127]

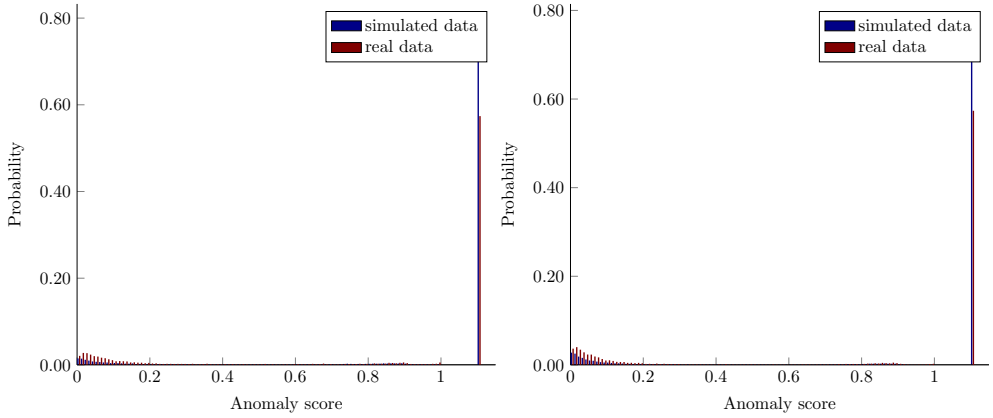
(c) Pevný-f-sIP [127]

(d) Pevný-$f^{\perp}$-sIP [127]

(e) MINDS[130]

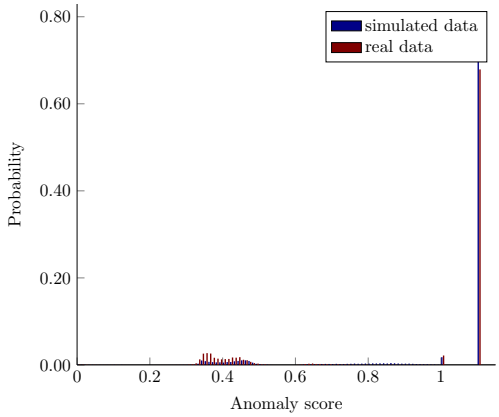
(f) Lak.Ent. [129]

Figure 4.3.1: Distribution of anomaly values for various anomaly detection method estimated on the real traffic and traffic simulated by the time variant joint probability model.

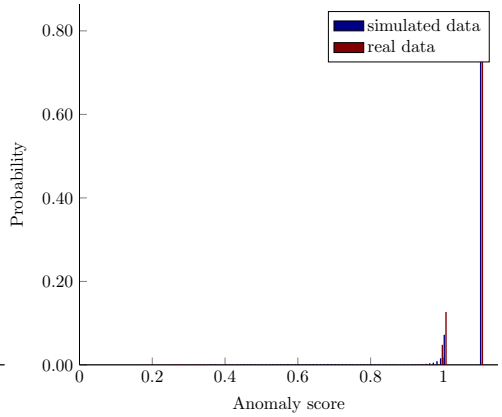


(a) Lak.Vol.-dIP [128]

(b) Lak.Vol.-sIP [128]



(c) Xu-dIP[132]



(d) Xu-sIP[132]

Figure 4.3.2: Distribution of anomaly values for various anomaly detection method estimated on the real traffic and traffic simulated by the time variant joint probability model.

4.4 Conclusion

In this chapter we have proposed a model for generating realistic NetFlow data that can be used for evaluation and configuration of anomaly-detection-based IDS. We introduced the *time variant joint probability model* that captures inter- and intra-flow relations as well as sequential character of user's behavior and compared this solution with two other simpler models (*random sampling* and *sampling with marginal model*). The experimental results have shown that the time variant joint probability model generates data that are more than 2× better in terms of Jensen-Shannon divergence compared to the marginal model. The key difference between our approach and most of the prior art is that we are able to construct the data in memory with much lower overhead compared to approaches that adopt simulation of complete network and as such our approach is well-suited for runtime adaptation of an IDS system.

The main limitation of the proposed model arises from its structure, where every server has its own separate model. In the case that several servers provide the same application, their models can be merged which can improve the estimations of the underlying probabilities. However, such extension will be considered in future work.

Although the proposed model is tailored to the NetFlow format, the general schema of the proposed model can be extended to capture not only NetFlow data but different types of communication as well. Therefore, the proposed model can be extended to other types of data (e.g. HTTP proxy logs) by incorporating necessary features in the proper parts of the model. However, these extensions, though beneficial, are out of the scope of this thesis and will be, again, considered in future work.

Chapter 5

Malware classification using malware behavioral traits

In previous chapter we have discussed the approach to simulation of legitimate network traffic. We have argued that due to characteristics of the human behavior a stochastic model is well suited for simulation of user's behavior.

However, in the case of malware (or malicious behavior in general) the situation is quite different as malicious actors (malware and its authors, or human attackers) constantly alter their network behavior to avoid detection. They introduce new types of attacks, novel methods for command and control communication (C&C) and monetization, at rate that makes the use of trained stochastic model impractical. This leads to a situation when obtaining network traffic for training model of legitimate user is relatively easy, but with malware we are always one step behind.

The main reason is that, we first need to *find* the malware, *record* its behavior, *analyze* it and *build* a model. This process, however, can take days or weeks (or even months in case of very sophisticated malware [24]). In order to overcome the complex process of analysis of malware behavior and extraction of a model, we propose to *emulate* the malware behavior rather than *simulate*. Therefore, we analyze malware samples in controlled environment (*sandbox*) record its network traffic and use it for online adaptation.

However, implementing such approach brings the problem of how to navigate in a vast amount of samples available, as for example VirusTotal [133] receives 2 millions of unknown samples per day. We need to label individual samples since (1) we are interested only in malware at this point and (2) we need to select interesting malware samples as using traffic traces from dominant malware family may introduce significant bias into the adaptation process.

In order to find missing labels, one can use well-known Anti-Virus products (AV products) or techniques based on static analysis. Problem is that such approaches typically relies on static patterns or statically extracted features to classify malware. This leads to situations when completely new malware, malware with encrypted code, or polymorphic malware that is able to modify its own binary, avoids detection as it is too different to be detected. Another problem is that AV engines typically use generic labels such as *generic, trojan, downloader*, etc. instead of names of malware families which does not fully counter the problem of categorization of unknown samples into malware families.

To address these problems we propose first to classify and then to cluster the samples using information extracted from malware behavior recorded in sandbox by means of dynamic analysis [134]. Since the samples are executed, it overcomes the problem with encoded or

polymorphic malware as they are forced to exhibit its true behavior. In the case of completely new malware, using dynamic analysis is beneficial as well, since information about malware behavior significantly simplifies the analysis.

In this chapter we introduce a novel representation of behavioral traces recorded in a sandbox and verify its robustness in classification scenario.[1]

5.1 Classification of sandboxed samples

To capture the malware behavior, we assume that execution of malware's actions involves interactions with resources visible at the operating system level. Examples of such interactions include *operations with files* during encryption of a victim's hard drive, *network communication* during data exfiltration or displaying advertisements, *operation with mutexes* used to ensure a single instance of malware is running, or manipulation with *registry keys* to ensure persistency after reboot. An additional source of information are error messages of the operating system itself. Such information is provided by the sandboxing environment as the following warnings: *dll not found* indicating missing dynamic library, *incorrect executable checksum* indicating corrupted binary, and *sample did not execute* indicating the fact that the binary was not executed at all due to various reasons (corrupted binary, sandbox was not able to copy the binary into VM, etc). All these system resources are then collected by the sandbox and used as an input of further analysis. The particular approach to collect this data can be different for every sandboxing solution, but it typically involves hooking system calls used for handling these resources [135], modification of hardware drivers or external monitoring (e.g. recording of network traffic on the level of host machine).

To model the interactions of a malware binary with resources, this work views each binary executed in a sandbox as a set of pairs of names and types of resources the binary interacted with. This view frames the problem as a *multiple instance learning* (MIL) problem [136] where each sample (binary) is represented as a *bag* that consists of a set of *instances* of different size. In our scenario an instance represents the pair of name and type of a resource the binary interacted with during sandboxing.

Variable sizes of samples and lack of order over their instances pose a challenge to traditional machine learning methods that expect samples to have fixed size. A recent review of MIL algorithms [25] lists various approaches to overcome this variability in sample sizes. One of the most popular (also adopted in this work) is *vocabulary-based method [137]* outlined in Algorithm 5.1. It employs clustering of instances to describe the sample by a fixed-dimensional vector with length equal to the size of vocabulary, i.e. a set of clusters, so that an ordinary machine learning method can be applied.

To convert a sample into a fixed-dimensional vector, all instances I from all training samples S are extracted and clustered by a suitable method per given resource type: files, mutexes, registry keys, network communication, where the resulting clusters represent the vocabulary. Note that due to the low number of possible warnings generated by the sandboxing environment, we consider every warning as a separated cluster. Next, for every instance i the closest cluster prototype c^* (a small random subset of the cluster of instances) of corresponding type is located. Finally, the binary representation is then used such that element of the vector equals to 1 iff there was an instance close to the particular cluster prototype.

[1] This chapter is based on the work published in [31].

Algorithm 5.1 High-level overview of training (function TRAIN) and classification (function PREDICT) of malware samples.

```
1: function TRAIN(S, y)                          ▷ Training samples and labels
2:     I ← extractInstances(S)
3:     C ← cluster(I)                            ▷ Clustering of instances (separately for individual types)
4:     X ← project(S, C)                         ▷ Projection of samples into binary vector (Alg. 5.2)
5:     M ← trainClassifier(X, y)
6:     return M, C                               ▷ Returns cluster centers C and trained classifier M
7: end function
8: function PREDICT(S', C, M)                    ▷ Testing samples S', clusters C and classifier M
9:     X' ← project(S', C)                       ▷ Projection of samples into binary vector (Alg. 5.2)
10:    ŷ ← predict(M, X')                        ▷ Classification of testing samples
11:    return ŷ
12: end function
```

Algorithm 5.2 Projection of samples S into binary vector using cluster centers C.

```
1: function PROJECT(S, C)                        ▷ Samples S and clusters C.
2:     X ← ∅
3:     for all s ∈ S do
4:         I ← extractInstances(s)
5:         x ← 0⃗
6:         for all i ∈ I do
7:             c* ← nnSearch(i, C)               ▷ Finds closest center c* to instance i.
8:             x[c*] ← 1
9:         end for
10:        X ← X ∪ {x}
11:    end for
12:    return X
13: end function
```

Once all samples are encoded as fixed-dimensional vectors, one can use a machine learning algorithm of choice to implement the classifier. This work uses the *random forest classifier* [26] due to its versatility, accuracy, and scalability, which make it a popular choice for many different machine learning tasks including malware classification [100].

Since the clustering is an essential component of the above algorithm, the definition of similarity over instances (resource names) greatly influences the accuracy of the system, and therefore it should reflect properties of the application domain. The rest of this section defines a specific similarity metric for each type of resources the malware interact with, namely on files (Section 5.1.1), network hostnames (Section 5.1.2), mutexes (Section 5.1.3), and registry keys (Section 5.1.4). The above metrics are then used in a modified clustering algorithm justified in Section 5.1.6.

5.1.1 Similarity between file paths

Although viewing file paths as strings would allow to use vast prior art such as Levenshtein distance [138], Hamming distance, Jaro-Winkler distance [139], or string kernels introduced in [140], the file systems were designed as tree structures with names of some folders (fragments of the path) being imposed by the operating system. The distance should reflect that. For example two files with paths `/Documents and Settings/Admin/Start Menu/Programs/Startup/tii9-fwliiv.lnk` and `/Documents and Settings/Admin/Start Menu/Programs/Accessories/Notepad.lnk` share large parts of their paths and common string similarities will return high similarity score, but the paths serve very different purposes, since the first file is a link to an application executed after the start of the operating system (OS), while the second is a regular link in the Start menu in Windows OS. Another aspect that prohibits the use of common string similarities is their computational complexity (typically $O(n^2)$ where n is the length of the string). The complexity combined with the number of resources to be clustered (in order of millions) leads to unfeasible time requirements. This motivates the design of a similarity that is fast and takes into the account the tree structure of the file system, special folders, and differences between folders and filenames.

The proposed similarity $s(x, x')$ of two file paths x and x' is defined as

$$s(x, x') = \exp\left(-w^T f(x, x')\right), \tag{5.1.1}$$

where w is a vector of weights and $f(x, x')$ is a function extracting a feature vector from file paths x and x'. Both the weight vector w and function f play an essential role and are both discussed in detail below.

The function f in (5.1.1) captures differences between the two paths x and x' by a fixed-dimensional vector. It first splits both paths x and x' into fragments x_i and x'_i using OS specific path separator[2], in the cases of MacOS and Windows changes all characters to lowercase, and assigns all fragments into one of the following four categories:

1. *known folder* – fragment x_i is a well-known folder in the list of folders imposed by the operation system (e.g. `Windows`, `Program Files`, `System32`, etc.),

2. *general folder* – fragment x_i is a not-well-known folder (e.g. unknown folders in `Program Files`, randomly generated folders in Internet Explorer cache folder, etc.),

3. *file* – fragment x_i is file,

[2]Unixes and MacOS uses '/' as a path separator, Windows uses '\'.

	Fragment 1	Fragment 2	Fragment 3
x	`Documents and Settings` (K)	`Admin` (G)	`Start Menu` (K)
x'	`Documents and Settings` (K)	`Admin` (G)	`Start Menu` (K)
	Fragment 4	Fragment 5	Fragment 6
	`Programs` (K)	`Startup` (K)	`tii9fwliiv.lnk` (F)
	`Programs` (K)	`Accessories` (G)	`Notepad.lnk` (F)

Table 5.1: Example of two paths x and x' separated into individual fragments with labels (K – known folder, G – general folder and F – file).

4. *empty* – artificial fragment used for padding the paths in cases when paths x and x' have different depths.

When all fragments are assigned to one of the above classes, their dissimilarity is captured by the function f as

$$f(x, x') = (f_{KK}, f_{KG}, f_{KF}, f_{KE}, f_{GG}, f_{GF}, f_{GE}, f_{FF}, f_{FE})$$

where

- f_{KK} is the number of fragments on the same level that were both classified as *known folder* and were not equal,
- f_{GG} is the sum of Levenshtein distances between all fragments on the same level that were classified as *general folder*,
- f_{FF} is the sum of Levenshtein distances of all fragments on the same level that were classified as *file*,
- f_{KG}, f_{KF}, f_{KE}, f_{GF}, f_{GE}, f_{FE} are the sums of all fragments of the same level and were classified as *known* and *general folder, known folder* and *file, known folder* and *empty, general folder* and *file, general folder* and *empty,* and *file* and *empty* respectively.

To illustrate the calculation of $f(x, x')$, let's consider the same two paths used above. At first, function f splits both paths into fragments and assign them into one of four categories (see Table 5.1). Assigning fragment to classes requires a list of known folders[3], which for the purpose of this example we assume to contain `Documents and Settings`, `Start Menu`, `Programs` and `Startup`, which are present in all windows installations. All corresponding folders from those two paths are therefore assigned to known folder class, while `Admin` and `Accessories` are labeled as general folders.[4] Individual elements of the vector $f(x, x')$ are calculated using the above rules as follows: the first rule applies to three fragments 1, 3, and 4 belonging to known folder class, but as they are all equal $f_{KK} = 0$; the second rule returns 0 based on analogous reasoning but for general folders; the third rule returns $f_{FF} = 0.7143$, which is the Levenshtein distance between `tii9fwliiv.lnk` and `Notepad.lnk`; the only mismatch is on fragment 5–known folder

[3]Full list of known folders is available online: https://github.com/SfinxCZ/Malware-analysis-using-multiple-instance-learning

[4]The first three known folders are embedded in the functionality of the Windows OS. The `Startup` folder has a specific meaning altering the behavior of the operation system since all programs listed in this folder are executed after the boot of the OS. On the other hand `Accessories` can be easily changed without major consequences.

and general folder yielding $f_{KG} = 1$; and finally all remaining elements of feature vector are 0. The output of $f(x, x')$ is captured by the feature vector

$$f(x, x') = (0, 0, 0.7143, 1, 0, 0, 0, 0, 0).$$

The weight vector w in (5.1.1) captures the contribution of individual elements of the feature vector $f(x, x')$. Imposing condition $w \geq 0$, in combination with construction of function f, bounds the value of the similarity function (5.1.1) $s(x, x') \in [0, 1]$. The similarity function then returns 1 (or values close to 1) if x and x' belong to the same class (files in `/temp/` directory, cache of the Internet Explorer, files in system directory, etc.) and values approaching 0 if they belong to different classes. Since the similarity function (5.1.1) was inspired by the popular Gaussian kernel, the parameter vector w was optimized using the Centered Kernel Target Alignment [141] (CKTA), which is a method to optimize kernel parameters. CKTA assumes training data $\{(x_i, y_i), \}_{i=1}^{m}$ where in our case x_i is a file path and y_i is the class of the path x_i. CTKA then defines *centered kernel matrix* as

$$[\mathbf{S}_c^w]_{ij} = \mathbf{S}_{ij}^w - \frac{1}{m}\sum_{i=1}^{m}\mathbf{S}_{ij}^w - \frac{1}{m}\sum_{j=1}^{m}\mathbf{S}_{ij}^w + \frac{1}{m^2}\sum_{i,j=1}^{m}\mathbf{S}_{ij}^w, \tag{5.1.2}$$

where $\mathrm{S}_{ij}^w = s_w(x_i, x_j)$ is the kernel matrix corresponding to the similarity function (5.1.1) parametrized by the weight vector w. CKTA maximizes correlation between labels and a similarity matrix by solving the following optimization problem

$$w^* = \arg\max_{w \geq 0} \frac{\langle \mathbf{S}_c^w, \mathbf{Y}_c \rangle_F}{\|\mathbf{S}_c^w\|_F \cdot \|\mathbf{Y}_c\|_F}, \tag{5.1.3}$$

where $\mathbf{Y}$ is target label kernel with $[\mathbf{Y}]_{ij}$ equaling to 1 when i^{th} and j^{th} paths from training data belongs to the same class and -1 otherwise, and $\langle \cdot, \cdot \rangle_F$ and $\|\cdot\|_F$ represent Frobenius product and Frobenius norm respectively defined for matrices $\mathbf{A}$ and $\mathbf{B}$ as

$$\langle \mathbf{A}, \mathbf{B} \rangle_F = \sum_{i=1}^{n}\sum_{j=1}^{m}\mathbf{A}_{ij} \cdot \mathbf{B}_{ij},$$
$$\|\mathbf{A}\|_F = \sqrt{\langle \mathbf{A}, \mathbf{A} \rangle_F}.$$

In below experiments (5.1.3) is solved by *stochastic gradient descent (SGD)* algorithm [27]. Note that although the path similarity $s(x_i, x_j)$ is not a valid kernel because it is not positive definite, the use of centered kernel alignment is still possible as the only limitation is that the global optimum might not be found.

To finish the example, the similarity function (5.1.1) with weight vector

$$w = (2, 10^{-5}, 1, 2.3, 1.6, 1, 0.36, 0.7, 0.9)$$

returns the value $s(x, x') = 0.049$, which correctly indicates that the two paths are different.

http://www.abc.com/av?sv=1&v=3.0.5&e=651af&u=http%3A%2F%2Fxyz.com%2F&if=0				
sv_number	v_version	e_string	u_url	if_number

Table 5.2: An example of URL with query string and extracted keys augmented with type of corresponding types of values.

5.1.2 Similarity of network traffic

To define the similarity between network resources, represented by domain names or server IPs, one has to overcome the randomization often employed by malware authors that renders trivial similarity based on names of network resources ineffective. To evade blacklisting command and control (C&C) channels of malware, malicious actors use various techniques to hide and obscure C&C operation. Popular approaches include randomization of domain names by generating them randomly (DGA), quickly changing hosting servers and/or domain names by fast flux, using large hosting providers like Amazon Web Services to hide among legitimate servers, etc. These techniques are relatively cheap (e.g. registering a new *.com* domain costs ~3USD per 1 year) and they allow for variation in domain names without updating disseminated malware binaries. In contrast, switching from one C&C paradigm to another requires costly update and therefore occurs relatively infrequently. These two properties contribute to the fact that each malware family uses specific patterns of domain names, paths, and parts of URLs. Exploiting these patterns allows to group domain names into clusters. In this work the similarity in network traffic is defined only for HTTP protocol, because it is presently the default choice for malware authors as it is rarely filtered in currently deployed defenses. However, the extension to other network traffic is possible [142].

The similarity of URL patterns used in this work has been adopted from [143], which has proposed to cluster network resources so that each cluster contains resources that exhibit behavior typical for one family of malware or its variant. The calculation of similarity starts by grouping all HTTP requests using the same network resource (domain, server IP) and building separate models for both query strings and paths in the URL. The overall similarity between two network resources is then computed as an average similarity between these resources using individual models. Since the similarity measures are not defined for every pair of resources (details are discussed in following sections), the average similarity is computed only from those models that are valid for given pair of network resources.

Query-based similarity

A query string is a part of the URL that carries information from clients to the specific function or resource on servers in the form of key-value pairs without hierarchical structure or specific order. We assume that servers serving the same application receive the same set of parameters via the query string. Values themselves are not that important, since they can differ from client to client but their types (number, string, e-mail address, etc.) are typically bundled with the parameters. Therefore, we extract all keys from set of all HTTP requests W and enrich every key k by the type of corresponding value (see Table 5.2).

To define similarity between two servers, we adopt the bag-of-words model [144], which is widely used in text mining. The vocabulary V is formed by all keys augmented with corresponding value types extracted from set of all HTTP requests W. The server s is then represented as a vector defined as

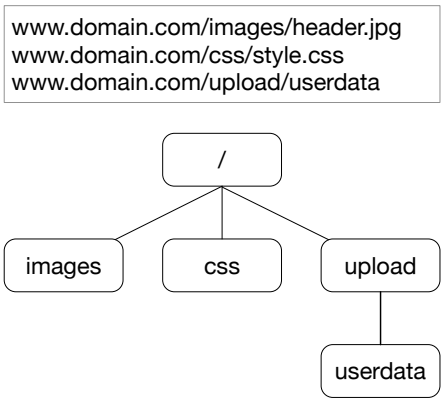


Figure 5.1.1: Three URLs and a path tree constructed from them. Each path is first separated to path fragments and tree is built from top to bottom, representing the directory structure.

$$q^{(s)} = (k_1^{(s)}, \ldots, k_{|V|}^{(s)})$$

where $k_j^{(s)}$ is equal to the number of occurrences of key j in the set of HTTP requests targeting server $s \in S$.

However, some query parameters are very common and don't discriminate between servers. To address this, we apply *term frequency-inverse document frequency* scaling (TF-IDF)[145] to each j^{th} element of vector $q^{(s)}$.

$$\begin{aligned}
\text{tf-idf}(j, s, S) =& \text{tf}(j, s) \cdot \text{idf}(j, S) \text{ where} \\
\text{tf}(j, s) =& 1 + \ln k_j^{(s)} \text{ and} \\
\text{idf}(j, S) =& \ln \frac{|S|}{\left|\left\{s | k_j^{(s)} > 0\right\}\right|}.
\end{aligned}$$

Finally, we employ cosine similarity on rescaled vectors $q^{(s)}$ to determine the similarity of two servers. Only servers that receive at least one query parameter are considered in the similarity calculation.

Path-based similarity

The idea behind using paths in HTTP requests to determine similarity between servers is based on assumption that two servers providing the same service are likely provide it at the same location specified by path. The importance of ordering path elements is also the reason why we do not use the bag-of-words representation in this case.

To reflect this assumption, we propose a representation that captures the directory structure of all paths ever visited on servers. Specifically, each server is represented by a tree, whose root is denoted "/", each node in the tree specifies one directory in the hierarchy and each path from root to a leaf represents one particular path observed in URL.

Figure 5.1.2: An example of data tunneling through URL path.

Construction of path trees is simple. Paths from each URL are split by the directory separator and a tree is built from the top to the bottom, representing the directory structure. Figure 5.1.1 shows an example of several paths and the associated tree.

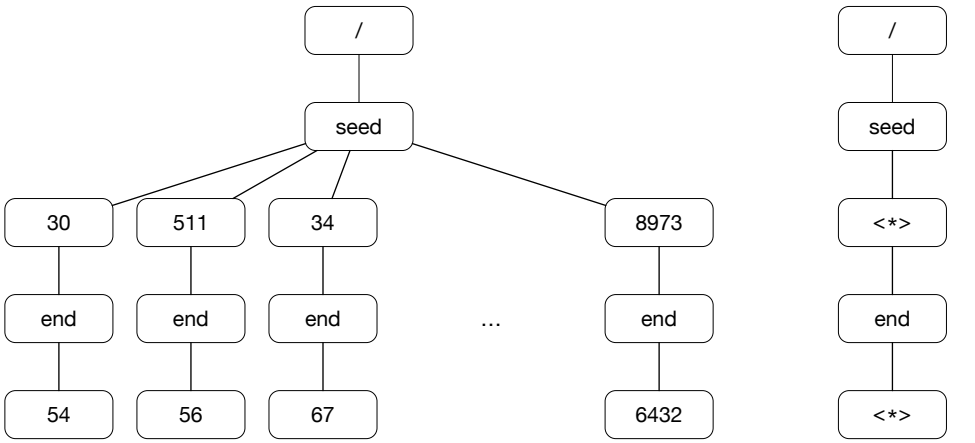
Paths in the URL do not always serve only as a pointers to the resources requested by the clients. Both legitimate and malicious applications may use URLs to tunnel (or exfiltrate) data from client to the server. Figure 5.1.2 shows an example of an URL tunneling of a Base64-encoded JSON. In this case, the corresponding part of the path cannot be considered a pointer to the resource, but rather looked at as data. To capture information about the tunneled data, we define new type of a node in the tree–the *data node*. In the rest of the text we refer to the nodes representing directories as *simple nodes*.

Data node keeps description of the data being tunneled. The exfiltrated information is not constant and changes in time. Therefore, the encoded strings are not exactly the same. On the other hand, because the structure of the data is always the same for the same malware, we can see some regularities in the encoded strings. The data node keeps track of parts of the data strings that are constant across all requests. In the case of URLs in Figure 5.1.2, the constant positions are emphasized in bold. Specifically, each data node is represented by a set of such positions. In the case depicted in Figure 5.1.2, the respective data node contains numbers between 1 and 15.

There is one more typical URL pattern that poses challenges to the proposed representation. Usually, parameters are specified in the query string, e.g. after the `?` symbol. However, in some applications the parameters are transferred using the path, for example it is often used in streaming media. An example of the parameter tunneling is depicted in Figure 5.1.3a. As you can see, the trees grow to large sizes and children nodes of the parameter values are redundant because most of them have the same value. Therefore, when we detect that parameters are being sent via the path, we join nodes representing parameter values into a single node with special value `<*>`. Figure 5.1.3b shows the same tree after compression.

Similarity metric While there are general similarity metrics for trees, such as edit distance, we propose a similarity tailored to our specific domain where ordering of the path elements is crucial but at the same time, ordering of children of any node is not. We take advantage of the kernel formalism [146] that allows us to define dot product of two trees in a transformed space. We define kernel

$$K(n,m) = L(n,m) \cdot (1 + C \cdot \sum_{\substack{u \in \mathrm{sub}(n) \\ v \in \mathrm{sub}(m)}} K(u,v)), \tag{5.1.4}$$



(a) original (b) compressed

Figure 5.1.3: Compressing of fragments of URL path that tunnels data.

where n and m are two nodes, sub$(\cdot)$ returns all children of the node, C determines how quickly the importance of deeper tree levels increases or decreases, and $L(n, m)$ is defined as

$$L(n,m) = \begin{cases} \mathbb{I}_{\{n=m\}} & \text{if n,m are both ordinary nodes,} \\ \mathrm{JI}(n,m) & \text{if n,m are both data nodes,} \\ 0 & \text{otherwise,} \end{cases} \tag{5.1.5}$$

where $\mathbb{I}_{\{n=m\}}$ is equal to 1 if m and n represent the same directory (name), otherwise it is equal to 0. Data nodes are represented as sets of integers, and $\mathrm{JI}(n, m)$ of two data nodes is Jaccard index of these sets.

It can be shown that K is a valid kernel function. The similarity itself is then determined by the formula

$$s(m,n) = \frac{2K(n,m)}{K(n,n) + K(m,m)}, \tag{5.1.6}$$

where n and m are root nodes of trees being compared.

Not all HTTP requests specify path, part of them can target the root directory itself. Such domains are not considered in the model, and their similarity to other domains from the point of view of the path-based model is 0.

5.1.3 Similarity between mutex names

Mutex (*Mutual exclusive object*) is a service provided by most modern operating systems to synchronize multi-threaded and multi-processes applications. This mechanism is popular among malware authors to prevent multiple infections of the same machine, because running two instances of the same malware can cause conflicts limiting the potential revenue. Mutexes are identified by their name, which can be an arbitrary string. The naming scheme is challenging for malware authors, because the names cannot be static, which would make them good

indicators of compromise of a particular malware, but they cannot be completely random either, because two independent binaries of the same family would not be able to check the presence of each other. Therefore malware authors often resort to pseudo-deterministic algorithms or patterns for generating mutex names. For some malware families these patterns are already well known, for example Sality [147] uses mutex names of the form `"<process name>.exeM_<process ID>_"` - `explorer.exeM_1423_`.

Since operating systems do not impose any restrictions on the names of mutexes, they can be arbitrary strings. Therefore standard string similarities such as Levenshtein distance, Hamming distance, Jaro-Winkler distance, etc. can be used. In experiments presented in Sections 5.2 Damerau-Levenshtein [148] was used, as it gives overall good results.

5.1.4 Similarity between registry names

In Microsoft Windows operating system, the primary target of the majority of malware, the system registry serves as a place where programs can store various configuration data. It is a replacement of configuration files with several improvements such as strongly typed values, faster parsing, ability to store binary data, etc. The registry is a key-value store, where key names have the structure of a file system. The root keys are `HKEY_LOCAL_MACHINE`, `HKEY_CURRENT_USER`, `HKEY_CURRENT_CONFIG`, `HKEY_CLASSES_ROOT`, `HKEY_USERS` and `HKEY_PERFORMANCE_DATA`; some root keys also always have sub-keys with specific names (`Software`, `Microsoft`, `Windows`, etc.). Due to similarity with a file system, the similarity distance is the same as the one defined in Subsection 5.1.1, but with a different set of names of known folders and a weight vector optimized on registry data rather than on files.

5.1.5 Similarity between error messages

As we have discussed in Section 5.1, error messages of operating system considered in this work are limited to: *dll not found* indicating missing dynamic library, *incorrect executable checksum* indicating corrupted binary, and *sample did not execute* indicating the fact that the binary was not executed at all due to various reasons (corrupted binary, sandbox was not able to copy the binary into VM, etc). The low number of possible error messages allows us to use the individual messages directly as cluster centers c^* and the similarity measure is therefore reduced to simple identity.

5.1.6 Clustering of system resources

The above similarities are not true distances, which limits the choice of applicable clustering methods to those that do not require proper distance metric between points. The Louvain method [149] is a popular choice and it is used in experiments below, because it also automatically determines the number of clusters and thus removes the need to set it manually. The use of the Louvain method is the authors' preference, but other clustering methods can be used as well; the reader is referred to [150] for an overview of methods applicable when true distance is not available.

The use of the Louvain method is not straightforward in our scenario because it requires a full adjacency matrix in advance. This results in a lower bound to computational complexity being $O(n^2)$ in the number of resources, which is clearly prohibitive as the number of unique resource names to cluster can easily reach the order of millions. To decrease the number of calculated similarities, an approach inspired by [151, 152, 153] is adopted where the Louvain clustering is used iteratively as summarized in Algorithm 5.3. Given a set of instances I of a

Algorithm 5.3 Approximative clustering algorithm for instances I (resource names).

```
1: function APPROXCLUSTER(I; k, m, ε)
2:     C = ∅
3:     while I ≠ ∅ do
4:         I' ← Random subset of size k from I
5:         C' ← cluster(I', m)              ▷ Cluster instances I' and create
                                              cluster prot. of size m.
6:         for all i ∈ I \ I' do
7:             c* ← nnSearch(i, C')         ▷ Find cluster prot. c* closest to
                                              instance i.
8:             if s(i, c*) > ε then
9:                 c* ← c* ∪ {i}
10:            end if
11:        end for
12:        C ← C ∪ C'
13:    end while
14:    return C
15: end function
```

particular type, in every iteration the algorithm selects a random subset $I' \subset I$ of the data of size k small enough for the Louvain method to be computationally feasible. The results of the Louvaine clustering are then transformed to cluster prototypes—random subsets of clusters with size limited to m. Remaining data $I \backslash I'$ are then traversed and all samples with similarity larger than ϵ to some cluster prototype $c^* \in C'$ are added to c^* and removed from I. Finally, C' is merged with the clustering C obtained in the previous iteration, and if I is not empty, the process is repeated.

Clearly the algorithm is an approximation of a clustering with complete data and its performance depends on the choice of parameters k and ϵ. Experiments indicate that if parameter k is large enough ($k = 10^4$) and parameter ϵ is set reasonably (in the experimental evaluation we use $\epsilon = 0.4$, see Section 5.2.2 for details), the results are comparable with clustering methods applied to the complete data. The computational complexity of this sequential approximation is $O\left(l \cdot \left(k \cdot (k-1)/2 + c_l \cdot m \cdot (n_l - k)\right)\right)$ where l is the number of iterations of algorithm (typically $l \leq 10$), n_l is the number of non-clustered samples in l-th iteration, k is the number of randomly selected samples, c_l is the number of cluster prototypes produced by the clustering algorithm in l-th iteration and m is the maximal size of a cluster prototype (typically $m = 10$). Since the parameter k is fixed and $k \ll n$, we can see that the number of evaluations of the similarity function is linear in the number of samples, which clearly outperforms the quadratic complexity required by the vanilla Louvain method. More details about performance of the proposed clustering and comparison of proposed similarity metrics with state-of-the-art approaches can be found in Appendix (A).

Malware family	#samples	Malware family	#samples
nemucod	13 781	amonetize	1172
cerber	12 829	nanocore	1032
bladabindi	10 945	loadmoney	964
locky	9894	yakes	892
gamarue	7694	bifrose	804
darkkomet	4664	autoit	781
hupigon	3555	kolabc	707
upatre	3269	waldek	686
tinba	3104	pdfka	649
scar	2961	shipup	625
swrort	2868	rebhip	613
zbot	2426	razy	599
virlock	1797	agentb	579
fareit	1763	poison	551
farfli	1749	xtrat	511
zegost	1719	onlinegames	502
virut	1556	ramnit	493
adwind	1537	magania	463
zusy	1505	atraps	461
ircbot	1447	softpulse	460
zerber	1329	banload	387
palevo	1270	ruskill	374
vobfus	1244	downloadassistant	373
delf	1228	binder	350
donoff	1211	*remaining MW families*	31 856
Total malicious			144 229
Total legitimate			87 026

Table 5.3: Number of samples of malware families in the data set. The malware families for individual samples were determined using AVClass tool [154].

5.2 Evaluation

In this section the proposed approach (further referred to as *MIL model*) is compared to the approach proposed by Rieck, et al. [94] (further referred to as *Rieck*) and the approach proposed by Mohaisen, et al. [92] (further referred to as *AMAL*). Rieck has been selected as a representative of the prior art that encodes malware behavior into a high-dimensional feature space using bag-of-words model built directly from data; it uses kernelized SVM to classify binaries. The second approach, AMAL, encodes malware behavior using a relatively low number of hand-made features; to classify unknown binaries AMAL trains multiple classifiers (SVM, decision trees, k-nearest neighbor, etc.) and selects the optimal classifier for given data using cross-validation.

AhnLab, *V3 Internet Security*	G Data, *InternetSecurity*
Avira, *Antivirus Pro*	Kaspersky Lab, *Internet Security*
Bitdefender, *Internet Security*	Microworld, *eScan internet security suite*
ESET, *Internet Security*	Symantec, *Norton Security*
F-Secure, *Safe*	Trend Micro, *Internet Security*

Table 5.4: Selected AV engines that received full 6 points for performance in AV-Test report from December 2016 [155].

5.2.1 Data set description

The dataset used for experiments contained 250 527 files collected from October 24, 2016 to December 12, 2016 using *AMP ThreatGrid* [156]. All files were also analyzed by *VirusTotal.com* service [133] and labeled using its verdicts as follows: a file was labeled as malicious if at least 4 out of 10 selected AV engines (see Table 5.4 for details) detected the file as malicious, and it was labeled as legitimate if none of the AV engines detected the file. Remaining files were discarded as unknown and removed from both training and testing sets in order to limit the effect of misclassifications by individual AV engines. The final numbers of files were: 144 229 malicious, 87 026 legitimate, and 19 272 discarded as unknown. The numbers of samples of individual malware families are summarized in Table 5.3.

All files were executed in sandbox by AMP ThreatGrid [156] service, using Windows 7 64bit (71% samples) environment, as it is the most popular OS at the time of writing[5], and Windows XP (29% samples) environment, since it is still widely deployed on embedded machines such as ATMs. Virtual machines were connected to the Internet without any filtering or restrictions that could by any mean prevent connections to command & control servers or other servers. The work here is not tailored to AMP ThreatGrid, as the same or similar information about binaries can be obtained by a number of different sandboxing solutions such as Cuckoo [157], Ether [158], or CWSandbox [159].

In contrast to the majority of prior art, binaries were divided into training and testing sets according to the dates they were collected rather than randomly. This approach is more realistic since it does not overestimate the detection performance as some malware families may not be known at the time of training, as they might have appeared later. Thus, all training samples collected prior to November 12, 2016 (72 963 malicious binaries and 48 152 legitimate binaries) were used for training, and remaining samples (71 266 malicious binaries and 38 874 legitimate binaries) were used for testing.

5.2.2 Hyper-parameter optimization

All compared methods have several parameters that have to be tuned to achieve good detection accuracy. While in Rieck and the proposed method the parameters have to be optimized using grid search (detailed below), AMAL is designed to perform such optimization during training in order to select both the optimal classifier (SVM, linear SVM, decision trees, logistic regression, k-nearest neighbor and perceptron) and its parameters and thus it does not need to optimize its parameters in advance.

Since Rieck uses SVM with L2 regularization and polynomial kernel there are two parameters that need to be tuned: misclassification cost $C \in \left\{10^{-2}, \ldots, 10^{8}\right\}$ and degree of the kernel

[5]According to http://www.w3schools.com/browsers/browsers_os.asp Windows 7 has 34.6% market share against 1.0% covered by Windows XP, 11.1% covered by Windows 8 and 30.9% covered by Windows 10.

$d \in \{1, \ldots, 5\}$. The optimal configuration achieving highest accuracy estimated by five-fold cross-validation on the training data was $C = 10^4, d = 4$.

The random forest classifier described in Section 5.1 contains several parameters such as the number of trees $K \in \{10, 20, 50, 100, 200\}$, maximal depth $d_m \in \{5, 10, 30, 50, \infty\}$, minimal number of samples in node to perform split $s_n \in \{2, 4, 6, 10, 20\}$, and criterion $c \in \{\text{gini}, \text{entropy}\}$. All remaining parameters (maximal number of features, minimal number of samples in leaf, maximal number of leafs, class weights, minimum weighted fraction of the total sum of weights in leaf, minimal impurity for split) were set to their default values as defined in the Scikit-learn library [160] since according to our experiments they have little influence on detection performance. The optimal configuration of parameters with respect to accuracy estimated by five-fold cross-validation on training data was $K = 100$, $d_m = \infty$, $s_n = 2$ and $c = \text{gini}$.

Additional two parameters (minimal similarity $\epsilon \in \{0.1, \ldots, 0.9\}$ and size of randomly selected subsets $k \in \{10^4, 2 \cdot 10^4, 5 \cdot 10^4, 10^5, 2 \cdot 10^5, 5 \cdot 10^5, \infty\}$) affect the clustering of the resource names described in Section 5.1.6. The minimal similarity was optimized on a manually labeled set of file paths and registry keys that were clustered with different values of ϵ. The resulting clusters were evaluated with respect to the *adjusted rand index [161],* a well-known score for evaluation of clustering algorithms, and the optimal value of $\epsilon = 0.4$ was selected. To find the optimal size of randomly selected subsets k the accuracy of the whole proposed method with different settings of parameter k was estimated using fivefold cross validation on randomly selected subset of training data[6]. Since the differences between various settings were negligible, the value of the parameter $k = 10^5$ was selected as a reasonable balance. Low value of parameter k increases the number of iterations l performed by the clustering algorithm, since too many samples are rejected to be too dissimilar to available cluster prototypes, and high value increases the quadratic cost for computation of adjacency matrix required by Louvain method.

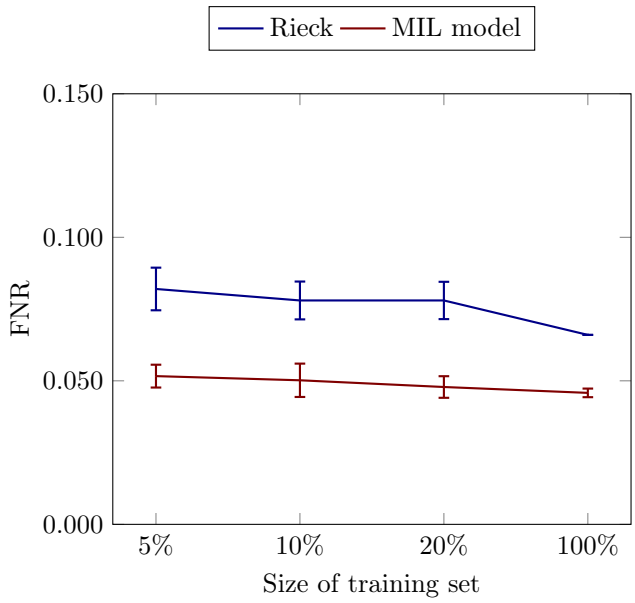
Classification performance was measured with standard evaluation metrics [162]: *true positive rate (TPR), false negative rate (FNR), true negative rate (TNR), false positive rate (FPR)* and *accuracy*. Since the experimental scenario is binary (positive malware vs. negative benign), the TPR (FNR) is the proportion of correctly (incorrectly) classified malware samples, TNR (FPR) is the proportion of correctly (incorrectly) classified legitimate samples and accuracy is the rate of correctly classified samples regardless their class.

5.2.3 Experimental results

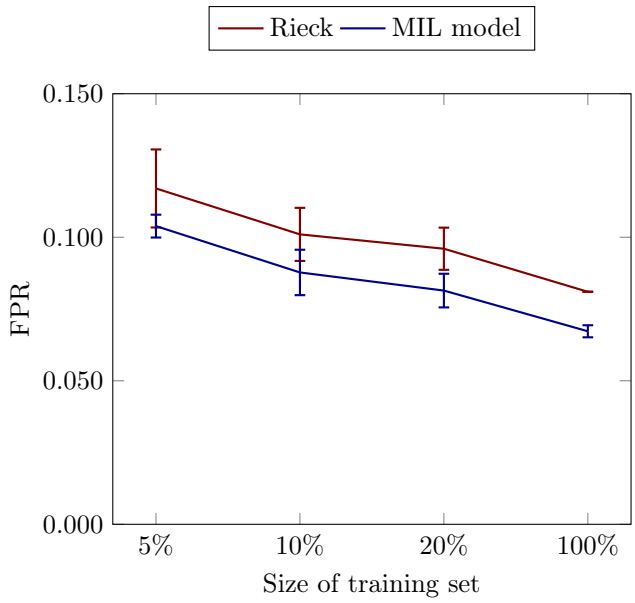
The comparison and evaluation is divided into two parts. The first experiment evaluates the detection performance of the MIL model, Rieck and AMAL trained on the full training set (121 115 samples), while the second experiment measures degradation of detection performance when only a limited number of data are available for training (5%, 10%, 20% and 100% of training samples). Note that to evaluate AMAL on the complete training set, the meta learner was not allowed to use SVM classifier with RBF kernel due to excessive computational requirements. Note that AMAL's meta-learner has never selected this variant of the SVM classifier in smaller experiments performed in this work, hence removing it most probably does not have any impact on the results.

The detection rates and accuracy of classifiers trained on all 121 115 training samples as estimated on testing samples are shown in Table 5.5. The differences between evaluation metrics indicate that the MIL model outperforms both Rieck and AMAL having the lowest false positive

[6]The subset was limited to $\sim$ 30 000 samples in order to limit the number of resources so that complete clustering could be performed.



(a) False negative rates



(b) False positive rate

Figure 5.2.1: Comparison of FNR and FPR for Rieck and proposed method trained on training sets of different sizes (5%, 10%, 20% and 100% of training samples).

	estimated on testing set			estimated on training set		
	TPR	FPR	ACC	TPR	FPR	ACC
MIL model	**0.954**	**0.067**	**0.943**	0.973	0.061	0.956
Rieck	0.934	0.081	0.926	**0.974**	**0.014**	**0.980**
AMAL	0.795	0.108	0.845	0.845	0.047	0.899

Table 5.5: True (TPR) and false (FPR) positive rates of evaluated methods estimated on the training and testing set.

rate and false negative rate. A deeper analysis of the misclassifications produced by the MIL model revealed that most of the false positives (legitimate binaries classified as malware) were software utilities such as TeamViewer that install themselves into system directories without any user interaction. Since their incidence in the training set was relatively low, the random forest was not able to precisely learn this type of behavior. A second source of errors are false negatives (malware samples classified as benign) where almost 70% are caused by insufficient numbers of training samples (less than 100 samples) from corresponding malware families. Another 11% of false negatives was caused by concept drift as a portion of testing samples exhibited different behaviors than training samples, i.e. created files or registry keys followed different pattern, network communication significantly different URLs, etc.

Large gaps between training and testing accuracies for AMAL and Rieck suggest that manually created features and BoW features do not generalize over longer periods of time as well as features created through clustering do. This suggests that clustering removes some randomization of resource names while retaining a large part of information content.

Figures 5.2.1a and 5.2.1b show graphs of FNR and FPR rates for larger sizes of the training set expressed as fraction of the data available for training. For fair comparison the testing set was kept static containing all 110 140 samples collected after November 12, 2016. Both graphs show that the MIL model is able to achieve lower FNR and FPR using fewer samples. In fact, the MIL model achieved FNR of 0.052 using just 5% of samples, while Rieck achieved 0.066 using the full training set. Similarly, the MIL model needed just 20% of samples to achieve the same FPR 0.081 as Rieck on all samples.

Figures 5.2.1a and 5.2.1b also shows that while false negative rates almost do not change with respect to the size of the training set (especially for the MIL model), the false positive rates decrease dramatically. This suggests that learning behavior of legitimate applications is more difficult than that of malware, which can be caused by the fact that the behavior of malware is more uniform than that of legitimate applications. This corroborates the motivation of this work, that even though malware authors try to randomize, they tend to randomize with same sort of regularity, which leads to uniformity.

5.2.4 Detection limits

The experimental results hint at where are the limits of classifying binaries executed in sandbox. When a binary (or all binaries of some malware family) does not perform any actions changing the data used by the discussed methods (dependent on modeling files, mutexes, network communication or registry keys) it clearly evades detection. An example of such malware is bitcoin miner that resides only in memory without any additional footprint (no operations with files, no operations with registry keys, no mutexes, very limited network communication). Such malware has to be carefully crafted to avoid any interaction with system resources (statically

compiled to carry all libraries in the executable, limited network communication, no mutexes ensuring that only single instance is running on the same machine, no persistency after reboot, etc.). Fortunately, at the time of writing this work, this is not an easy task and accordingly the majority of malware authors choose to interact with system resources rather than sacrifice functionality.

Another limitation is the fact that a growing number of malware families are equipped with advanced anti-VM and anti-sandbox features and/or are targeted to specific environments (Stuxnet [24]). Such malware families do not reveal their true purpose during sandboxing or mimic less severe types of malware (adware, PUA[7], etc.). This fact is recognized by the community as the main factor hindering the performance of dynamic analysis as the whole. Addressing this issue is out of the scope of this work.

The last aspect we need to discuss is the false positive rate. The analysis of the results from Section 5.2 revealed that a large number of false alarms is caused by applications that install themselves into system directories without user's interaction and since their number is limited, the classifier was unable to fit this behavior. A solution is of course to improve the training data by including a larger number of such samples and thus achieve lower false positive rate.

5.2.5 Scalability and computational complexity

The last aspect we will discuss is the scalability of the proposed solution and prior art. Since the proposed solution employs clustering to project the input data into a feature space with a lower dimension, a large portion of the training time is spent in the clustering phase. However, the preprocessing of the dataset used in above experiments was much faster ($\sim$ 2h50min) than the highly optimized pre-computation of the full kernel matrix required by Rieck ($\sim$ 7h). This is caused by the fact that the time required by Rieck for preprocessing grows quadratically with the number of training samples in contrast to the proposed solution with linear complexity (up to an additive constant, see Section 5.1.6). Moreover, the proposed solution can be easily distributed since in every iteration the nearest neighbor search depends only on a limited set of current cluster prototypes C'.

Another benefit of the proposed solution is tied to the representation itself. Since the clustering is performed only on training samples, in order to classify unknown samples we need to store only the cluster prototypes determined during training. For the whole training dataset used in this work, which contains over 7 million unique resource names projected into $\sim 40\,000$ features, only 400 000 instances need to be stored. In contrast, the kernelized SVM classifier used by Rieck et al. requires to store all training samples (over 120 000 samples in the data discussed in Section 5.2) with all actions (on average 2000 actions per sample) in order to make prediction on unknown samples.

In contrast to both the proposed solution and Rieck, AMAL does not need any preprocessing since the features can be extracted per sample. However, the complexity arises from the design of the training process. Authors in [92] argue that the dynamic selection of both optimal algorithm and its parameters provides optimal results, but this design makes the training process computationally expensive since every training of the meta-learner requires to evaluate all possible combinations of parameters for all its classifiers. Another aspect is the selection of classifiers itself. Authors propose to use an array of classifiers such as kernelized SVM, linear regression, decision trees, perceptron, etc. However, the complexity of some classifiers (e.g. kernelized SVM) prevents any large-scale training. Moreover, according to the evaluation the AMAL's detection capabilities are not sufficient for real-world deployment since both FPR and FNR are nearly 20%, which is clearly insufficient.

[7] Potentially unwanted application.

5.2.6 Conclusion

In this chapter we have proposed an approach for classification of unknown binaries as malicious or legitimate using a model of malware behavior observed through their interactions with the operating system and network resources (operations with files, mutexes, registry keys, operations with network servers or error messages provided by the operating system). The proposed representation employs an efficient clustering of resource names to reduce the impact of randomization commonly employed by malware authors to avoid detection and projects malware samples into a low-dimensional space suitable for classifiers such as random forest.

The proposed solution was extensively tested and compared to related state-of-the-art on a large corpus of binaries where it demonstrated significant increase in precision of malware detection on independent test data.

Chapter 6

Malware clustering

In Chapter 5 we motivated the use of emulation rather than simulation of malware traffic traces for online adaptation. We introduce all problems connected to the lack of labels and propose approach for malware modeling and classification using information extracted from dynamic analysis of malware samples.

In this chapter we describe the second part of the process focused on the clustering of malware samples to recover malware families. We use the same representation described in previous chapter and build probabilistic model that is able to cluster malware samples according to their behavior. However, we still need human analyst to assess the threat level of malware samples required in the game-theoretical adaptation process described in Chapter (3). To increase the efficiency of the manual analysis we use the probabilistic model to define human-friendly prioritization of identified clusters and extraction of readable behavioral indicators that maximize interpretability.[1]

6.1 Model definition

To describe behavior of a malware we use the approach described in Chapter 5. Here we only briefly summarize its basic principle.

The proposed model assumes that malware actions are visible through their interaction with system resources, which in this work includes *(1) interactions with files* (e.g. during encryption of a victim's hard drive), *(2) network communication* (e.g. during data exfiltration or displaying advertisements), *(3) operation with mutexes* (e.g. used to ensure a single instance of malware is running), *(4)* manipulation with *registry keys* (eg. to ensure persistency after reboot), and *(5) error messages* of the operating system itself.[2]

During analysis of the malware sample in the sandboxing environment we thus record *(1)* all paths of files that were created, deleted or modified, *(2)* all network communication, *(3)* all names of mutexes that were opened or created, *(4)* all registry keys that were created, deleted or modified and *(5)* all error messages emitted by the operating system during sandboxing. All these system resources are then collected by the sandbox and used as an input of further

[1] This chapter is based on the work presented in [32].

[2] Error messages are provided by the sandboxing environment as one of the following warnings: *dll not found* indicating missing dynamic library, *incorrect executable checksum* indicating corrupted binary, and *sample did not execute* indicating the fact that the binary was not executed at all due to various reasons (corrupted binary, sandbox was not able to copy the binary into VM, etc.). Note that in this text error messages have been included into system resources to keep the notation clean even though technically they are not system resources.

analysis. The particular approach to collect this data can be different for every sandboxing solution, but it typically involves hooking system calls used for handling these resources [135], modification of hardware drivers or external monitoring (e.g. recording of network traffic on the level of host machine).

6.1.1 Model description

Let us now assume we have a set of all system resources observed during sandboxing of a set of samples S, i.e. union of all file paths, mutexes, network communications, registry keys and warnings touched by any binary from the set S. We further assume that the set has fixed length D. The interactions of a sample $s \in S$ with these resources are then encoded into a binary vector x, where x_i is 1 iff the sample interacted with i^{th} resource and zero otherwise. Because the length of such vector would be enormous due to the fact that malware's authors frequently employ randomization of the system resources, Section 5.1 discusses in detail how the data is pre-processed using *multiple instance learning*. However, for now it is sufficient to assume that a sample is encoded into a binary feature vector x with fixed dimension D, i.e. $x \in \{0,1\}^D$.

The binary feature vector x encoding the interaction of the malware binary with system resources can be viewed as a realization of a random variable where interaction with a particular resource i follows a *Bernoulli distribution*, $x_i \sim Ber(\mu_i)$, $p(x_i|\mu_i) = \mu_i^{x_i}(1-\mu_i)^{(1-x_i)}$ with $\mu_i \in (0,1)$ controlling the probability that the resource is used. For simplicity it is assumed that the use of resources is independent of each other and therefore for one execution of a binary in sandbox can be written

$$p(x|\mu) = \prod_{i=1}^{D} p(x_i|\mu_i). \tag{6.1.1}$$

Although the independence is rather crude simplification, since in reality the creation of a file can be for example related to a particular network connection, the simplification has been accepted to make the computation tractable. The multivariate Bernoulli distribution [163] that captures relations between individual resources has $2^D - 1$ parameters which would make the model impractical for large number of dimensions (in the experimental evaluation the number of dimensions reaches $\sim 50\,000$).

Parametrizing the use of resources by a single vector μ for all samples is limiting since each malware family can have different pattern of resource usage. Therefore every malware family should have its own parameter vector μ. This phenomenon motivates the use of mixture of Bernoulli models, where vector μ is replaced by a matrix $M = (\mu_{ki})_{k,i=1,1}^{K,D}$ with K rows and D columns where each row corresponds to one malware family and each column corresponds to one resource. The affiliation of particular samples to j^{th} malware family is then indicated by a variable z. The variable z is encoded with one-hot representation vector with 1 on k^{th} position indicating that sample belongs to k^{th} family and zeros everywhere else. The extended model is then formalized as

$$p(x|z,M) = \prod_{k=1}^{K} p(x|M_{k\cdot})^{z_k}, \tag{6.1.2}$$

where $M_{k\cdot} = \mu$ introduced in Equation (6.1.1) for k^{th} malware family.

Since the affiliation of sample x to particular malware family is unknown, the variable z is itself a random variable with *categorical distribution* with K possible outcomes parametrized by a vector $\pi \in (0,1)^K$, $\sum_{k=1}^{K} \pi_k = 1$. The probability $p(z|\pi)$ then describes the probability of malware family described by vector z and is formalized as

$$p(z|\pi) = \prod_{k=1}^{K} \pi_k^{z_k} \tag{6.1.3}$$

The complete model is then parametrized by the vector π determining the probability of observing particular malware family and by the matrix $M = (\mu_{ki})_{k,i=1,1}^{K,D}$ characterizing the use of resources by individual malware families. Formally it is defined as

$$p(x) = p(z|\pi)p(x|z,M) \tag{6.1.4}$$

$$= \prod_{k=1}^{K} \pi_k^{z_k} \cdot \prod_{i=1}^{D} \left(M_{ki}^{x_i} (1 - M_{ki})^{(1-x_i)} \right)^{z_k}. \tag{6.1.5}$$

These equations describe a generative model which has the following interpretation. A binary belongs to the malware family j with probability π_j and during its execution it interacts with i^{th} resource with probability M_{ji}. The advantage of the probabilistic formulation is that it captures variability in malware families and their use of resources. It also allows individual binaries from the same malware family to use slightly different resources since the actual usage of resources is a realization of the probability distribution defined in Equation (6.1.1).

The parameters π and M are learned (inferred) from observed data $X = \{x_1, \ldots, x_N\}$ (executions of N malware binaries in the sandbox). The membership of samples to the malware family is captured by variables $\{z_1, \ldots, z_N\}$ arranged in binary matrix Z where $z_{nk} = 1$ if n^{th} malware is a member of k^{th} family. Since this membership is unknown, Z is treated as a hidden unobserved variable and is inferred during the learning process as well. After the inference, Z contains assignments of individual samples to different malware families (clusters).

Using the above notation, the likelihood function of observed data with parameters π and M is given by

$$p(X, Z|\pi, M) = p(Z|\pi) \cdot p(X|Z, M)$$

$$= \prod_{n=1}^{N} \prod_{k=1}^{K} \left(\pi_k^{z_{nk}} \prod_{i=1}^{D} \left(M_{ki}^{x_{ni}} (1 - M_{ki})^{(1-x_{ni})} \right)^{z_{nk}} \right), \tag{6.1.6}$$

corresponding to the well-known *Bernoulli mixture model* [27], which has been successfully used in many different areas [164, 165, 166]. Since it is not possible to derive analytical formula for posteriori distribution of the parameters $p(\pi, M|X)$ and sampling methods [167] are computationally infeasible due to the enormous number of dimensions, a well-known *expectation-maximization (EM)* algorithm [168, 27] is adopted. It iteratively optimizes the likelihood function $p(X, Z|\pi, M)$ until convergence is reached. In each iteration it alternates the following two steps:

1. In E-step the posterior probability of component k for a given sample x_n is approximated as

$$\gamma(z_{nk}^{(t+1)}) = \frac{\pi_k^{(t)} \prod_{i=1}^{D} p(x_{ni}|M_{ki}^{(t)})}{\sum_{k=1}^{K} \pi_k^{(t)} \prod_{i=1}^{D} p(x_{ni}|M_{ki}^{(t)})}$$

   where values of parameters π and M are fixed on values from previous iteration (indicated by superscript t).

2. In M-step the complete-data likelihood function (6.1.6) is maximized with respect to parameters π and M using the values $\gamma(z_{nk}^{(t+1)})$ estimated in E-step which leads to following update equations

$$\mu_k^{(t+1)} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}^{(t+1)}) x_n$$
$$\pi_k^{(t+1)} = \frac{N_k}{N}$$

   where $N_k = \sum_{n=1}^{N} \gamma(z_{nk}^{(t+1)})$ represents the effective number of samples in component k.

The algorithm is started with random initialization of $\gamma(z_{nk})$ using Dirichlet distribution with parameter $\alpha = 10^{-1}$ as it provides good level of sparsity and the values of parameters π and M are recomputed during first iteration.

6.2 Model application

Since the main purpose of the proposed model is to assist the human analyst by grouping unknown binaries into coherent clusters, prioritizing their analysis, and extracting their characteristics we now describe the application of the model described in previous section to the problem of clustering sandboxed samples.

The processing pipeline displayed in Figure 6.2.1 starts with preprocessing of malware traces recorded in sandbox and extraction of vector representation $x_n \in X$ for every sample (see Section 5.1). Next, we estimate parameters Z, M and π of the proposed model using this data X. The main idea is that each malware family or its variant uses specific resources and therefore malware samples should form clusters in $\{0,1\}^D$ space, ideally with one cluster containing samples from one malware family. When the model is fitted to data X, this assumption is reflected by samples from the same malware family following similar distribution over the hidden variable Z. The identifier of the most probable cluster for a given sample x_n is then obtained as

$$z^* = \underset{z \in \{1,\ldots,K\}}{\operatorname{argmax}} \; p(z|x_n, \pi, M) = \underset{k \in \{1,\ldots,K\}}{\operatorname{argmax}} \; \gamma(z_{nk})$$

and serves as a cluster label for given sample x_n. It is possible that one malware family spreads over multiple clusters, which is mainly caused by their modular design as different modules operate with different system resources. Nevertheless the behavior of malware binaries inside one cluster should be relatively uniform and it should be easy to create behavioral indicators for them as we discuss in Section 6.2.2.[3]

[3] In the text malware families and their variants are not differentiated because the model does not make a difference between them.

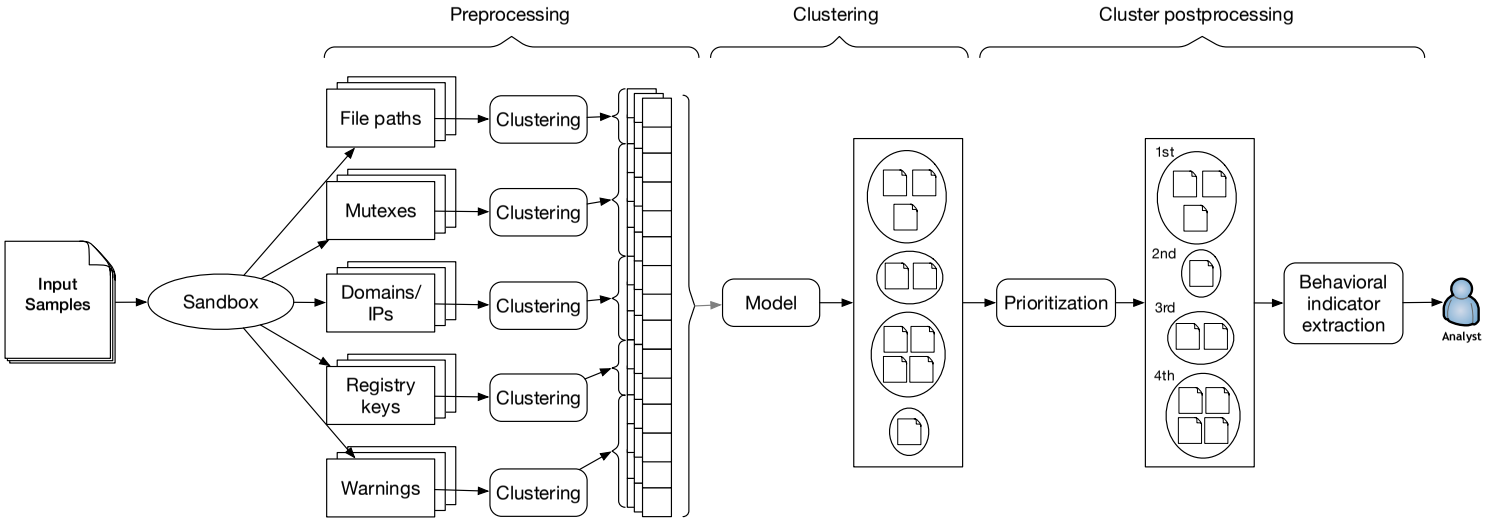


Figure 6.2.1: Schema of the proposed approach that chains the preprocessing (building data representation–see Section 5.1), clustering and preparation of clusters for human analyst (see Section 6.2).

6.2.1 Cluster prioritization

To further simplify the use of outputs, the clusters should be ordered such that a large number of samples can be analyzed quickly. The proposed heuristic (further referred as a *score*) prioritizes clusters of samples with homogenous behavior as their analysis is easier for human analyst and at the same time it promotes the largest clusters since their quick identification has higher impact. The purity of a cluster is approximated by the inverse value of entropy of the distribution defined by Equation (6.1.1) and the size is approximated by its probability π_k. Note that values of both the probability of a cluster π_k and parameters of distribution $p(x|\mu_k)$ are estimated during fitting the model to unknown data X. The prioritization score for cluster k is then formalized as

$$s(k) = \frac{\pi_k}{-\sum_{i=1}^{D} H(\mu_{ki})}, \text{ where} \tag{6.2.1}$$

$$H(\mu_{ki}) = M_{ki} \cdot \log M_{ki} + (1 - M_{ki}) \cdot \log (1 - M_{ki}) \tag{6.2.2}$$

The final ranking of clusters is obtained by computing the score for every cluster and ordering them in the decreasing order.

6.2.2 Behavioral indicator extraction

The last step of the analysis is the extraction of behavior indicators (BIs) that provide an insight into the behavior of analyzed samples. In the proposed model BIs are represented by system resources (files, mutexes, network communications and registry keys) used by the samples during the sandbox run. A behavior indicator that is a good candidate for an *indicator of compromise* (IOC) should be frequently used by samples from a given cluster and rarely used by samples from other clusters. This requirement is formalized for cluster k and system resource i as

$$r(i,k) = \ln M_{ki} + \frac{1}{K-1} \sum_{l=1, l \neq k}^{K} \ln(1 - M_{li}), \tag{6.2.3}$$

where the first term $\ln M_{ki}$ corresponds to the probability that samples from cluster k will interact with the system resource i ($p(x_i = 1, z = k|\pi, M) \to 1$) and the second term represents the average probability that the samples from other clusters will not interact with the system resource i ($\forall l \neq k, p(x_i = 1, z = l|\pi, M) \to 0$). Using average of probability overcomes the instability which can demote system resources common for cluster k but used by the few samples from other clusters as well in favor to system resource rarely used by samples from cluster k and not used by any other samples. This is caused by the fact that without the normalization term $\frac{1}{K-1}$ the term $\sum_{l=1, l \neq k}^{K} \ln(1 - M_{li})$ would have disproportionally higher influence than the term $\ln M_{ki}$. The variable $r(i,k)$ then provides a ranking score for the system resources within the cluster k where higher values indicate good candidates for IOCs.

6.3 Evaluation

This section presents the results of the experimental comparison of the proposed model to AMAL [92] on the problem of clustering of unknown samples. AMAL is the closest prior art, as it clusters malware binaries on basis of their interaction with system resources. Furthermore, clustering using the generative model is compared to the popular K-Means [169] algorithm to observe benefits or disadvantages of the probabilistic formulation. Lastly, we evaluate the cluster prioritization and extraction of behavioral indicators.

6.3.1 Data set description and performance metric definition

The dataset used for experimental evaluations contained random selection of samples from files that were submitted to *AMP ThreatGrid* [156] sandboxing service by its customers between October 24, 2016 and December 12, 2016 and as such can be considered as a representative example of the threat landscape. All samples were analyzed by *VirusTotal.com* service [133] and their labels were determined using *AVClass* tool [154]. Then, malware families with less than hundred samples were removed, which yielded 130 198 samples. The exact numbers of samples of individual malware families are summarized in Table B.2 in Appendix B.2. Since all three compared solutions are applicable to any file type as long as the underlying sandboxing solution is able to instrument them, the evaluation dataset contained samples of various file types such as 32bit PE executables, 64bit PE executables, JavaScripts, VisualBasic scripts, etc. The complete list of file types along with number of samples is summarized in Table B.1 in Appendix B.1.

All samples were analyzed in sandbox by AMP ThreatGrid [156] service, using Windows 7 64bit (71% samples) environment, as it is the most popular OS at the time of writing[4], and Windows XP (29% samples) environment, since it is still widely deployed on embedded machines such as ATMs. The sandboxing environment (OS) for individual samples was selected randomly with exception of 64bit PE executables that were executed solely on Windows 7 64bit. Virtual machines were connected to the Internet without any filtering or restrictions that would intervene connections to command & control infrastructure or other servers. Note that the approach proposed in this chapter is not specific to AMP ThreatGrid. Similar information can be obtained with a number of different sandboxing solutions such as Cuckoo [157], Ether [158], or CWSandbox [159].

Although clustering algorithms are technically unsupervised, they usually contain several hyper-parameters that need to be tuned to achieve good performance. Samples were therefore divided according to the time of their first observations, such that those collected before October 27th, 2016 (17 064 samples further referred to as *training set*) were used to find the optimal configurations of all methods and remaining samples (113 134 samples further referred as *testing set*) were used for the final evaluations. The time-based split makes the optimization phase more realistic since in real-world scenario all parameters have to be set in advance with the risk that some malware families will not be available.

The performance of all methods was evaluated using standard clustering criteria, namely *adjusted rand index*, *homogeneity*, *completeness* and *V-measure*. *Adjusted rand index (ARI)* [170] is a corrected-for-chance version of *Rand index (RI)* that guarantees values close to one for perfect clustering and values close to zero or negative values for random clustering. In order to define ARI between partition $K = \{k_1, \ldots, k_{|K|}\}$ and target partition $C = \{c_1, \ldots, c_{|C|}\}$, we first define a contingency table that summarizes overlap between these two partitions as

[4]According to http://www.w3schools.com/browsers/browsers_os.asp Windows 7 has 33.2% market share against 0.9% covered by Windows XP, 10.2% covered by Windows 8 and 33.1% covered by Windows 10.

Class	c_1	c_2	$\dots$	$c_{\lvert C\rvert}$	$\sum$
k_1	n_{11}	n_{12}		$n_{1\lvert C\rvert}$	$n_{1\cdot}$
k_2	n_{21}	n_{22}		$n_{2\lvert C\rvert}$	$n_{2\cdot}$
$\vdots$			$\ddots$		$\vdots$
$k_{\lvert K\rvert}$	$n_{\lvert K\rvert 1}$	$n_{\lvert K\rvert 2}$	$\dots$	$n_{\lvert K\rvert\lvert C\rvert}$	$n_{\lvert K\rvert\cdot}$
$\sum$	$n_{\cdot 1}$	$n_{\cdot 2}$	$\dots$	$n_{\cdot\lvert C\rvert}$	

where $k_1, \dots, k_{|K|}$ represents clusters of instances generated by the clustering, $c_1, \dots c_{|C|}$ represents groups of instances with the same label, $n_{ij} = |k_i \cap c_j|$ is the number of instances in i-th cluster with j-th label, $n_{i\cdot}$ is the total number of instances in i-th clusters and v_j is the total number of instances with j-th label. Using this contingency table we define the adjusted rand index as follows

$$ARI(K,C) = \frac{\sum_{i,j}\binom{n_{ij}}{2} - \frac{\sum_i \binom{n_{i\cdot}}{2}\sum_j \binom{n_{\cdot j}}{2}}{\binom{n}{2}}}{\frac{1}{2}\left(\sum_i \binom{n_{i\cdot}}{2} + \sum_j \binom{n_{\cdot j}}{2}\right) - \frac{\sum_i \binom{n_{i\cdot}}{2}\sum_j \binom{n_{\cdot j}}{2}}{\binom{n}{2}}}.$$

The main advantage of the adjusted rand index is that it is normalized with respect to random clustering, i.e. it gives values close to zero or negative values for random clustering even for datasets with large number of clusters and values close to one for perfect clusterings. The main drawback of the adjusted rand index is its hard interpretability. Although from its value can be seen the quality of the clustering, it does not provide any insight whether the clusters are pure but fractioned into multiple clusters, or concise but different labels are merged into the same cluster.

To provide additional information about clusters we use entropy-based evaluation metrics known as *homogeneity* h, *completeness* c and *V-measure* v_1 [171]. *Homogeneity* h measures the "purity" of resulting clusters and equals to one if all labels in each cluster are the same, i.e. the distribution of labels for each cluster has zero entropy. It can be seen as a clustering's counterpart to precision used in classification scenarios. Formally it is defined as

$$h(K,C) = \begin{cases} 1 & H(C,K) = 0 \\ 1 - \frac{H(C|K)}{H(C)} & \text{otherwise,} \end{cases}$$

where

$$H(C|K) = -\sum_{i=1}^{|K|}\sum_{j=1}^{|C|}\frac{n_{ij}}{N}\log\frac{n_{ij}}{\sum_{j=1}^{|C|} n_{ij}},$$

$$H(C) = -\sum_{j=1}^{|C|}\frac{\sum_{i=1}^{|K|} n_{ij}}{N}\log\frac{\sum_{i=1}^{|K|} n_{ij}}{N},$$

and N is number of instances in the input data.

Completeness c measures how well individual labels are grouped together. It equals to one if samples with one label are all in one cluster, or more formally, if the distribution of cluster ids for single label has zero entropy, which makes it closely related to recall used in classification. It is defined as

$$c(K,C) = \begin{cases} 1 & H(K,C) = 0 \\ 1 - \frac{H(K|C)}{H(K)} & \text{otherwise,} \end{cases}$$

where

$$H(K|C) = -\sum_{j=1}^{|C|}\sum_{i=1}^{|K|}\frac{n_{ij}}{N}\log\frac{n_{ij}}{\sum_{i=1}^{|K|}n_{ij}},$$

$$H(K) = -\sum_{i=1}^{|K|}\frac{\sum_{j=1}^{|C|}n_{ij}}{N}\log\frac{\sum_{j=1}^{|C|}n_{ij}}{N}.$$

They are combined together with equal importance into a single value by *V-measure* defined as

$$v_1 = \frac{2\cdot h\cdot c}{h+c},$$

that resembles the F-measure from classification scenarios.

Additionally, the quality of individual clusters is measured by *purity* [172, 173] defined as number of samples from most prevalent malware family in given cluster normalized by its size.

6.3.2 Hyper-parameter optimization

AMAL's hyper-parameters consist of the metric that defines similarity between samples (cosine, correlation, hamming or jaccard), the linkage method in hierarchical clustering (average, centroid, complete, median, single, ward), and the value of threshold t for the hierarchical clustering. For every combination of hyper-parameters we have estimated the value of V-measure on the training data and the one with highest value (jaccard metric, average linkage method and threshold $t = 0.25$) was used in final evaluation.

The only parameter in the proposed method is the number of components K, which corresponds to the number of clusters. Setting this number is generally unresolved problem, since one typically does not know the number of clusters in data in advance. The training data were therefore clustered with with different number of components $K = \{50, 100, 200, 500, 1000, 2000\}$ and the K with the highest V-measure was used in the evaluation. Since the EM algorithm depends on starting conditions (the problem may have many local extrema), the accuracy of the clustering for every configuration was estimated using five executions. As the optimal number of clusters was then determined $K = 500$.

Similarly, K-Means clustering algorithm requires specification of the number of cluster in advance as well, which has been determined using the same approach as in the previous paragraph. The optimal number of clusters for K-Means was $K = 200$.

	h	c	v_1	ARI	#clusters
AMAL	0.602	0.499	0.546	0.093	5856
K-Means	0.690	0.561	0.619	0.189	200
Proposed model	**0.767**	**0.609**	**0.679**	**0.321**	500

Table 6.1: Homogeneity (h), completeness (c), V-measure (v_1) and adjusted rand index (ARI) estimated on test dataset for AMAL, K-Means and proposed model.

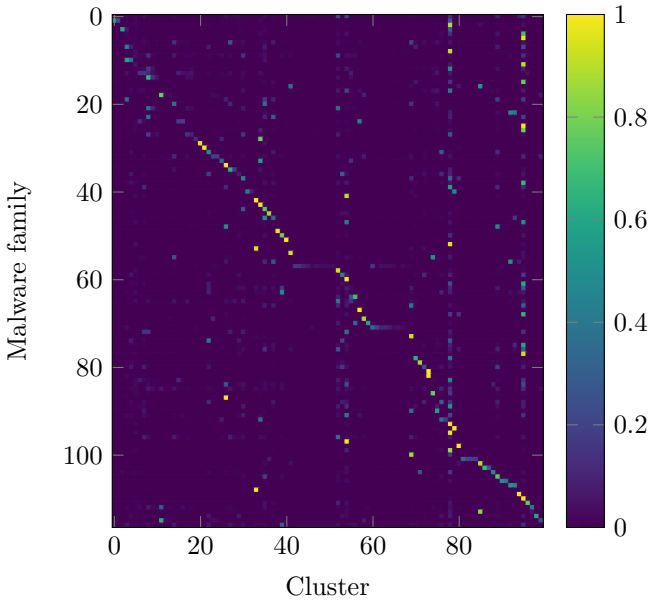


Figure 6.3.1: Confusion matrix of proposed model for 100 largest clusters.

6.3.3 Clustering performance

The first part of experimental evaluation compares the clustering performance of the proposed model, K-Means clustering algorithm with the same features as the proposed model, and AMAL. The performance is measure as homogeneity, completeness and V-measure estimated on testing dataset for all three approaches using settings described above and is summarized in Table 6.1. As can be seen from the values of the V-measure and ARI, the proposed model significantly outperforms both AMAL and K-Means. Clusters recovered by the model are the purest and the individual malware families are the least fractioned into multiple clusters. These results are illustrated in more detail in Figures 6.3.1, 6.3.2, 6.3.3 that show confusion matrices of largest 100 clusters for proposed model, K-Means and AMAL respectively, normalized by the number of samples per family (i.e. every point displays ratio of samples from particular malware family clustered into particular cluster). Although the view is not complete, these selections covered 87%, 90% and 80% of samples for proposed model, K-Means and AMAL respectively which represent significant portion of samples.

Confusion matrices show that the low performance of the AMAL method (Figure 6.3.3) is mainly caused by the fact that it failed to separate large portion of samples ($\sim 37\,000$ samples) and grouped them into single cluster. This indicates that hand-designed features were not able

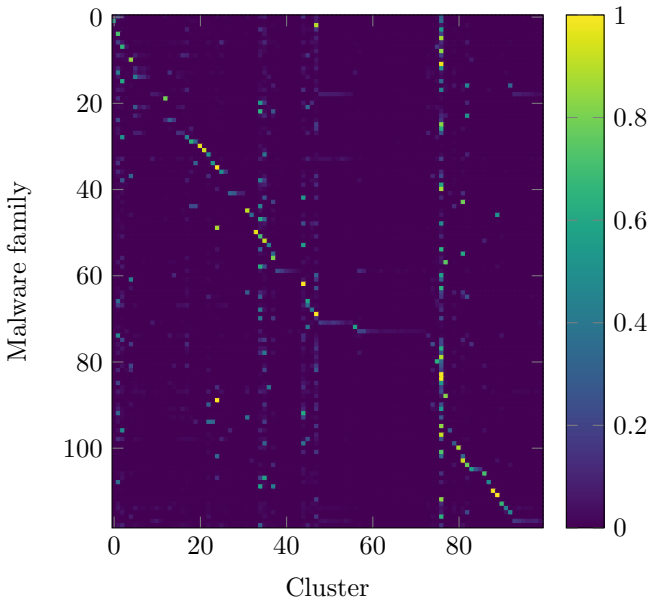


Figure 6.3.2: Confusion matrix of K-Means for 100 largest clusters.

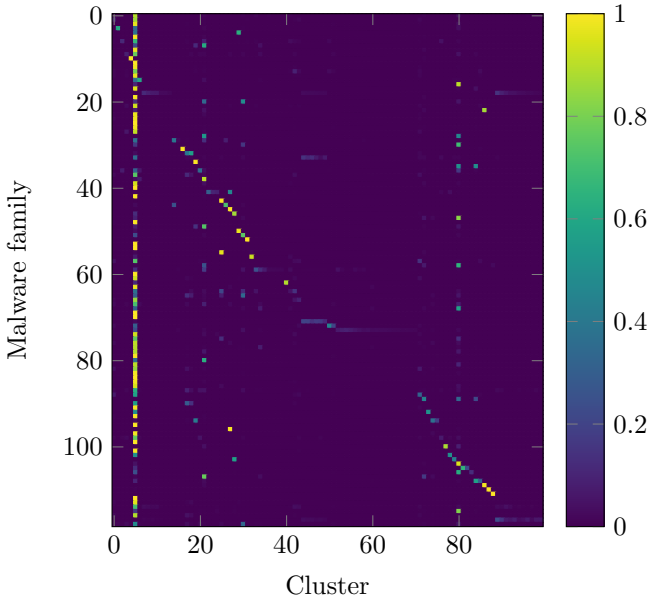


Figure 6.3.3: Confusion matrix of AMAL method for 100 largest clusters.

to precisely capture the behavior of a new, previously unseen malware. Comparison of confusion matrices of proposed method and K-Means revealed that although K-Means provided better results than AMAL, it was still outperformed by the proposed method as it created clusters that failed to separate several malware families. This fact is significant for clusters 2–5, 30–50 and mainly cluster 76 that incorrectly grouped more than 20 different malware families such as *lotoor* (malware family #61), *pdfka* (malware family #84) or *zegost* (malware family #117) that, however, got correctly clustered together by the proposed model. These results indicate that the probabilistic model correctly captures malware behavior. It is caused by the fact that K-Means is not able to distinguish minor differences between individual malware families (e.g. one file specific to particular malware family) as the euclidean distance embedded in the K-Means smooths the distances between individual malware samples. Although it would be possible to find specific combination of distance metric, clustering algorithm and its parameters that provides better or at least comparable results, such combination would introduce high bias as it would be overfitted to our data and would lack the interpretability of the proposed model.

Detailed analysis of clusters created by the proposed model revealed that it correctly discovered several malware families such as *winner, waldek, ircbot, downloadguide*, *downloadassistant*, *kolabc, dlhelper, softpuls,* etc., but some malware families such as *adwind, gamarue, lock*y, *nemucod* or *zbot* were split into multiple pure clusters. This is caused by the fact that a single label may refer to multiple variants of the same malware family (*locky* or *adwind*), by modular design of some malware families (*zbot*), or by the fact that some malware families deliver additional infections (*nemucod*) with different behavior. However, several clusters contained samples from multiple malware families such as *atraps* (also known as *cryptowall*), *delf, flystudio, palevo* or *winsecsrv*. Analysis of these clusters revealed that these samples neither run properly (e.g. crashed due to missing libraries) nor exhibited their true behavior (advanced anti-vm protection, no command from bot master, etc.), and the only interactions with system resources recorded during sandboxing were caused by the operating system handling the crash of the binary.

This exposes the main limitation of the proposed solution, because if malware samples do not exhibit any behavior, the model is not able to cluster them correctly as it does not have enough information to distinguish individual malware families. In case that malware samples are equipped with advanced anti-vm technique we can improve the results by optimizing the sandboxing environment (anti-anti-vm/anti-anti-sandbox techniques) and thus limit its detectability by malware. Although such approach is successful for some malware families (e.g. *gamarue*), malware authors quickly adapt the anti-vm techniques to avoid detection (e.g. *atraps*). As the hardening techniques are usually tailored for specific sandboxing solution, their description is out of the scope of this work and reader is referred to [174] for deeper analysis of this problem.

To conclude the first part of experiment, the proposed solution significantly outperforms the prior art and it is able in most cases correctly recognize binaries from the same family under the condition that the binary has exhibit its true behavior in sandbox.

6.3.4 Prioritization score

As has been discussed in Section 6.2, resulting clusters should be presented to a security analyst sorted to facilitate quick analysis of a large number of samples. Instead of selecting clusters at random, the optimal prioritization should focus the analyst's attention to large and pure clusters that can be easily analyzed (ideally to clusters that contain samples from single family).

To verify this assumption, clusters identified by the proposed model from the testing data were ordered by prioritization score proposed in Section 6.2 (further referred to as *Score*), and by their size (referred to as *Size*). Figure 6.3.4a shows the average purity for first 5, 10, 15, 20,

30, etc. clusters ordered by the Score and those ordered by the Size. Similarly, Figure 6.3.4b shows the cumulative number of samples for first 5, 10, 15, 20, 30, etc. clusters prioritized by the Score and the Size. We observe that the purity of clusters ordered by Score is much higher than of those ordered by Size, while cumulative numbers of samples of top n clusters are very similar. This means that the proposed Score promotes purer clusters without too much sacrificing their size.

In order to further verify these results we have performed deeper analysis of top 25 prioritized by the Score and the Size. Tables 6.2 and 6.3 provide details about malware categories the samples from particular cluster belong to and their ratio. Note that malware category listed as *unknown* covers malware families that we were not able to reliably assign to any category. The statistics indicate that the Score promotes larger number of pure clusters[5] (14 out of 25) compared to the Size (7 out of 25) and these clusters cover larger portion of samples, namely 21 919 samples in the case of Score vs. 18 510 samples in the case of Size. In order to provide complete results, we include results obtained on clusters created by K-Means (Table 6.5) and AMAL (Table 6.4) prioritized by their size as the Score is not applicable. The detailed statistics regarding the clusters created by AMAL show that in top 25 clusters there are 12 considered pure, but they cover only 12 323 samples which is almost half of samples covered by the proposed model. Similarly, top 25 clusters created by K-Means contains 6 pure clusters, covering 9520 samples. These numbers indicates that the proposed model combined with the prioritization score correctly promotes clusters so that the security analyst can analyze large number of samples efficiently.

6.3.5 Behavioral indicators

The last step of the analysis of a cluster is the extraction of behavior indicators that can be considered as indicators of compromise. To verify the performance of the ranking score proposed in Section (6.2), top 20 clusters identified by the model from the testing data and prioritized by Score were manually analyzed and searched for the first system resource that matched some known IOC for malware families in a given cluster. Known IOCs were collected from reports published by various AV companies (e.g. [175]) such as Symantec, ESET, TrendMicro, etc., or publicly available collections of various behaviors frequently exhibited by malware [176, 177].

Table 6.6 summarizes number of resources in cluster, and the rank and the probability μ of usage of given resource for the first BI that matched some known IOC. It shows that for 12 clusters out of 20 extracted BIs matched some known IOC to one of the top system resources promoted by the ranking score.[6] The lower ranks of the matched system resources for clusters 13 and 14 are caused by the nature of the malware families dominant in these clusters. Although *bladabindi* malware uses system resources that are not frequently used by other malware families in our dataset (`/windows/syswow64/netsh.exe`, `/windows/microsoft.net/.../machine.config`, etc), they cannot be considered as IOCs since they are part of Windows operating system or other common software. In this case the only system resources that matched known IOCs are specific registry keys modified by only portion of samples from this cluster which lowers their rank. Similarly, *fujacks* malware interacts with system resources that are unique for this malware family but are part of the operating system and are incorrectly promoted over the actual IOCs.

For 6 clusters we were not able to find any IOC that will match with the proposed BIs. In the case of clusters 3, 6 and 16 it is caused by the fact that these clusters contain samples that

[5]Here, we consider cluster with at least 98% samples from the same category as pure rather than 100%.

[6]The example of matched IOCs is a path to malware's main binary, specific registry key, known C&C domain or mutex with specific name.

Cluster	Size	Malware categories with coverage
1	**1302**	**trojan (1.00)**
2	**811**	**banking trojan (1.00)**
3	4337	unknown (0.41), worm (0.26), dropper (0.18)
4	**531**	**banking trojan (1.00)**
5	2135	dropper (0.92), banking trojan (0.08)
6	6320	dropper (0.33), unknown (0.33), trojan (0.14), rat (0.08), information stealer (0.07)
7	**495**	**banking trojan (1.00)**
8	141	dropper (0.94), banking trojan (0.06)
9	4228	dropper (0.62), banking trojan (0.37)
10	**904**	**rat (0.99)**
11	5967	dropper (0.74), ransomware (0.26)
12	**374**	**trojan (1.00)**
13	**91**	**unknown (1.00)**
14	5942	information stealer (0.96)
15	**9366**	**ransomware (0.98)**
16	1197	unknown (0.33), dropper (0.30), trojan (0.09), rat (0.07), ransomware (0.06), information stealer (0.05)
17	191	dropper (0.90), banking trojan (0.10)
18	**1475**	**ransomware (1.00)**
19	2876	banking trojan (0.92), unknown (0.08)
20	**456**	**banking trojan (1.00)**
21	**1556**	**rat (0.98)**
22	**1402**	**ransomware (1.00)**
23	**2273**	**banking trojan (1.00)**
24	1271	trojan (0.78), unknown (0.11), rat (0.05)
25	**883**	**ransomware (0.99)**
Total	56 524	
Total pure	21 919	

Table 6.2: Top 25 clusters created by proposed model and prioritized with *score* with the malware categories the clustered samples belong to along with the ratio of every category in particular cluster.

Cluster	Size	Malware categories with coverage
1	**9366**	**ransomware (0.98)**
2	6320	dropper (0.33), unknown (0.33), trojan (0.14), rat (0.08), information stealer (0.07)
3	5967	dropper (0.74), ransomware (0.26)
4	5942	information stealer (0.96)
5	4337	unknown (0.41), worm (0.26), dropper (0.18)
6	4228	dropper (0.62), banking trojan (0.37)
7	2876	banking trojan (0.92), unknown (0.08)
8	2661	dropper (0.97)
9	**2273**	**banking trojan (1.00)**
10	2135	dropper (0.92), banking trojan (0.08)
11	1726	dropper (0.75), ransomware (0.25)
12	1691	trojan (0.45), dropper (0.39), unknown (0.10)
13	1623	dropper (0.83), ransomware (0.17)
14	**1556**	**rat (0.98)**
15	**1475**	**ransomware (1.00)**
16	**1402**	**ransomware (1.00)**
17	**1302**	**trojan (1.00)**
18	1292	dropper (0.74), ransomware (0.26)
19	1283	information stealer (0.96)
20	1271	trojan (0.78), unknown (0.11), rat (0.05)
21	1266	ransomware (0.82), unknown (0.18)
22	1197	unknown (0.33), dropper (0.30), trojan (0.09), rat (0.07), ransomware (0.06), information stealer (0.05)
23	**1136**	**ransomware (1.00)**
24	1005	unknown (0.53), information stealer (0.31), rat (0.08)
25	964	information stealer (0.73), unknown (0.21)
Total	66 294	
Total pure	18 510	

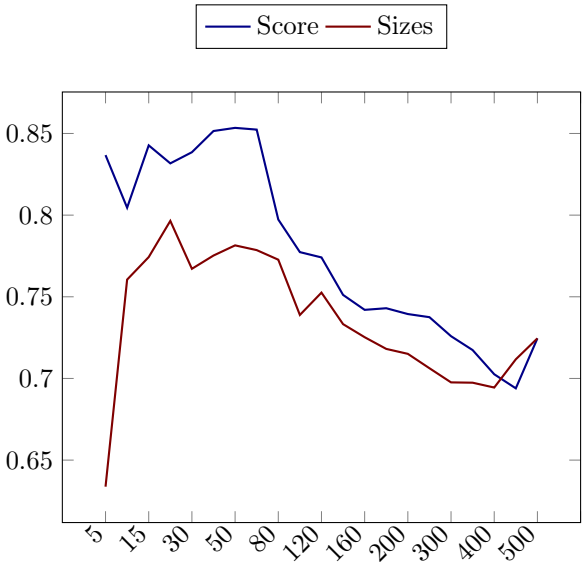
Table 6.3: Top 25 clusters created by the proposed model prioritized by size with the malware categories the clustered samples belong to along with the ratio of every category in particular cluster.

Cluster	Size	Malware categories with coverage
1	37303	information stealer (0.27), unknown (0.21), rat (0.17), dropper (0.16), trojan (0.07)
2	6263	dropper (0.46), banking trojan (0.25), trojan (0.13), pua (0.08)
3	3678	dropper (0.62), banking trojan (0.38)
4	2889	banking trojan (0.92), unknown (0.08)
5	2472	ransomware (0.58), unknown (0.22), pua (0.08), dropper (0.08)
6	**2105**	**ransomware (0.99)**
7	**1710**	**banking trojan (1.00)**
8	1465	trojan (0.92), unknown (0.08)
9	**1438**	**ransomware (0.99)**
10	1153	ransomware (0.91)
11	1099	ransomware (0.33), dropper (0.30), trojan (0.25), unknown (0.12)
12	**1066**	**banking trojan (0.99)**
13	**1013**	**ransomware (0.98)**
14	988	trojan (0.90), unknown (0.08)
15	**889**	**ransomware (0.99)**
16	888	dropper (0.64), ransomware (0.36)
17	861	trojan (0.93), dropper (0.05)
18	819	dropper (0.88), ransomware (0.12)
19	**806**	**dropper (0.99)**
20	**750**	**ransomware (0.98)**
21	**686**	**worm (0.99)**
22	**664**	**dropper (0.98)**
23	658	dropper (0.76), ransomware (0.24)
24	**601**	**ransomware (1.00)**
25	**595**	**ransomware (0.98)**
Total	72 859	
Total pure	12 323	

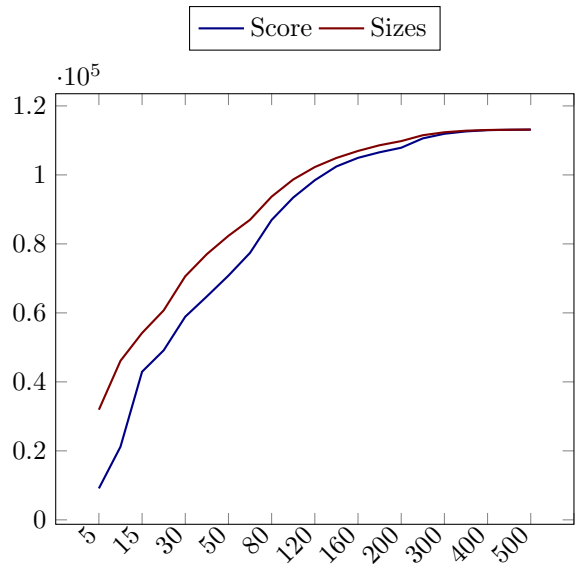
Table 6.4: Top 25 clusters created by AMAL and prioritized by size with the malware categories the clustered samples belong to along with the ratio of every category in particular cluster.

Cluster	Size	Malware categories with coverage
1	12719	unknown (0.30), dropper (0.20), rat (0.15), worm (0.09), trojan (0.09), information stealer (0.05),
2	4211	information stealer (0.96)
3	3478	banking trojan (0.90), unknown (0.06)
4	3450	dropper (0.71), ransomware (0.29)
5	3192	dropper (0.75), banking trojan (0.15), unknown (0.06)
6	3073	dropper (0.79), ransomware (0.21)
7	3044	unknown (0.31), dropper (0.26), pua (0.18), trojan (0.12), ransomware (0.10)
8	3007	unknown (0.31), rat (0.28), information stealer (0.23), worm (0.08), trojan (0.06)
9	2850	unknown (0.42), dropper (0.20), rat (0.14), trojan (0.12)
10	**1943**	**ransomware (0.99)**
11	**1890**	**banking trojan (1.00)**
12	1766	dropper (0.79), unknown (0.15), banking trojan (0.05)
13	**1713**	**banking trojan (1.00)**
14	1574	unknown (0.41), dropper (0.25), trojan (0.09), rat (0.07), information stealer (0.06)
15	1548	unknown (0.40), trojan (0.31), pua (0.16)
16	1511	information stealer (0.97)
17	**1490**	**ransomware (1.00)**
18	1353	rat (0.91), unknown (0.05)
19	**1344**	**trojan (1.00)**
20	1328	trojan (0.75), unknown (0.12), rat (0.05)
21	1266	information stealer (0.95)
22	1143	information stealer (0.33), rat (0.31), unknown (0.24), trojan (0.09)
23	**1140**	**ransomware (0.99)**
24	1140	banking trojan (0.89), dropper (0.08)
25	1130	trojan (0.40), unknown (0.23), information stealer (0.20), rat (0.10), dropper (0.08)
Total	62 303	
Total pure	9520	

Table 6.5: Top 25 clusters created by K-Means and prioritized by size with the malware categories the clustered samples belong to along with the ratio of every category in particular cluster.



(a) Average purity



(b) Cumulative number of samples.

Figure 6.3.4: Average purity and cumulative number of samples for top 10, 20, 30, etc. clusters for proposed prioritization (*Score*) and size-based prioritization (*Size*).

Cluster	Rank	#BIs	Prob	Cluster	Rank	#BIs	Prob
1	1	80	1.000	11	N/A	42	N/A
2	1	19	1.000	12	1	48	1.000
3	N/A	20	N/A	13	13	150	1.000
4	2	15	1.000	14	14	61	0.866
5	1	15	1.000	15	N/A	142	N/A
6	N/A	22	N/A	16	N/A	51	N/A
7	1	15	1.000	17	1	15	1.000
8	1	15	1.000	18	9	37	0.844
9	1	47	0.993	19	1	52	0.994
10	1	22	0.993	20	1	16	1.000

Table 6.6: Rank and probability μ of first behavior indicator that matches known IOCs of most frequent malware family in generated clusters. N/A indicates that no behavior indicator matched known IOC.

did not exhibit their true behavior and the only interactions with system resources recorded during sandboxing relate to operating system handling application crashes. In case of cluster 11 it is caused by the nature of behavior of the malware dominant in this cluster. Similarly to *bladabindi, nemucod* interacts with files specific for this malware but they are all part of the operating system and thus cannot be considered as IOCs. Cluster 15 contains *cerber* ransomware that encrypted various different files during sandboxing which can be easily recognized by malware analyst. However, the recorded behavior indicates that these samples belong to different variants for which we were not able to find any known IOCs.

6.4 Discussion

The experimental results revealed, that when a binary (or all binaries of some malware family) does not perform any action reflected by interaction with files, mutexes, network communication, and registry keys, it clearly evades detection by the proposed model. An example of such malware is bitcoin miner that resides only in memory without any additional footprint (no operations with files, no operations with registry keys, no mutexes, very limited network communication). Such malware has to be carefully crafted to avoid any interaction with system resources (statically compiled to carry all libraries in the executable, limited network communication, no mutexes ensuring that only single instance is running on the same machine, no persistency after reboot, etc.). Fortunately, at the time of writing this work, this is not an easy task and the majority of malware authors choose to interact with system resources rather than sacrifice functionality.

Another limitation is the fact that a growing number of malware families are equipped with advanced anti-VM and anti-sandbox features (Gamarue/Andromeda [178]) and/or are targeted to specific environments (Stuxnet [24]). Such malware families do not reveal their true purpose during sandboxing or mimic less severe types of malware (adware, potentially unwanted application, etc.). This fact is recognized by the community as the main factor hindering the performance of dynamic analysis as a whole. Addressing this issue, however, is out of the scope of this work.

Another important aspect that affects the performance of the proposed method is the quality of the data representation. First problem is when system resources are incorrectly split

into number of different instance clusters due to the randomization of their names. This fragmentation can cause the split of binaries from the same malware family into multiple clusters. Even though this is clearly an error, it does not severely affect the analysis. According to our experience when malware families are split into low number of clear clusters, they can be still quickly analyzed, which is the goal of this work. Second problem arises when unrelated system resources are merged into single instance cluster. This can cause that corresponding malware samples from different malware families are grouped into single cluster. Again, according to our experiments this situation is not so common to drastically affect the analysis. The only clusters containing samples from multiple binaries are the ones that did not reveal their true behavior or did not execute correctly. Such situation can be easily identified, since the interaction with system resources limits to the noise generated by the operating system (operations with system logs, creating generic mutexes, etc.).

The quality of the data representation raises question which system resources are the most important for malware clustering. According to our experiments, model limited only to file paths and mutexes achieves $\sim 70\% - 75\%$ of the clustering performance compared to the case with complete information. However, only the combination of all five types of system resources provides optimal results as there are some malware families (e.g. *scar*) that use legitimate file paths to hide their payload but uses specific command and control infrastructure which is exploited by the model and detected as possible IOC.

The last question we need to discuss is the scalability of the proposed method. The most of the time required for the overall analysis was consumed in the preprocessing phase ($\sim$ 1h15min) and in the fitting of the model ($\sim$ 50min); the time required for cluster prioritization and extraction of behavioral indicators was negligible (less than 1min). The theoretical analysis of the preprocessing phase shows the linear computational complexity in the number of samples (see Chapter 5 for more details). Similarly, the EM algorithm used to find the parameters π and M is linear in the number of samples which enables good scalability to large-scale dataset. In comparison the time required by AMAL was only $\sim$ 28min. However, since the computational complexity of the hierarchical clustering used in AMAL is $\Theta(N^2)$ or $\Theta(N^3)$ depending on the linkage method [179] the scalability to the large-scale dataset is limited.

6.5 Conclusion

This chapter focused on providing complete pipeline for analysis of large number of unknown binaries executed in the sandbox. The pipeline starts with grouping samples according to their behavior in the sandbox, continuing by prioritizing bigger groups with homogeneous behavior, and ending with extraction of human-readable descriptions of groups of binaries. The proposed method utilizes a probabilistic model of malware behavior observed through its interactions with operating system and network resources (operations with files, mutexes, network servers, registry keys or system messages). To the best of our knowledge, such a complete pipeline was not yet published, albeit there is a prior art on the first step, the clustering of unknown binaries.

Individual steps of the pipeline were evaluated on a large corpus of binaries, and the first problem of clustering has been compared to the prior art. This comparison showed that the proposed method outperforms the related state-of-the-art approaches as it produces clusters that are purer, while individual malware families were less fractioned into multiple clusters. Moreover, the evaluation also revealed limitations of the proposed method, as it fails on samples that do not interact with resources monitored by the sandbox.

Chapter 7

Conclusion

The goal of this thesis was to design an approach for dynamic reconfiguration of an anomaly-detection-based intrusion detection system (IDS) under the assumption of *a rational attacker*. The proposed architecture (see Section 1.2) adopts the game-theoretical principle which allows precise modeling of interactions between the IDS and an attacker. The key novelty is the online definition of the security game combined with the *indirect online integration* principle (*challenges*) which allows us to estimate the game parameters, find the optimal strategy and immediately reconfigure the IDS accordingly. The experiments performed with a simplified version of real-world IDS suggest that the computational cost of game-theoretical approach is very low and does not affect the effectiveness of the adaptation process. In particular, our results show that proposed approach outperforms the trust-based baseline solution and thus the proposed solution represents a suitable tool for dynamic reconfiguration of an IDS.

Since parameters of the security game are estimated online, the results of the adaptation process heavily depend on the quality of the injected traffic (challenges). As we have discussed in Chapter 1, using static challenges is far from optimal as they do not follow the dynamic changes in the background traffic (e.g. day vs. night) or the ever-changing trends in the threat landscape. Therefore, we proposed to employ *(1) simulation of legitimate network traffic* and *(2) emulation of malicious behavior with network traffic observed during execution of malware binaries in controlled environment.*

The simulation of network traffic proposed in Chapter 4 adopts a probabilistic model which generates continuous stream of network traffic that simulates behavior of legitimate users. The proposed model incorporates various aspects of user's behavior such as changes in the behavior profile during the day or sequential character of user's behavior and thus it addresses limitations of the static challenges. We experimentally verified that the simulated traffic correctly mimics real network traffic so the anomaly-detection-based IDS cannot distinguish it. Moreover, the data is generated directly in memory with much lower overhead compared to approaches that adopt simulation of the complete network which makes the proposed approach well-suited for the purpose of dynamic reconfiguration.

In order to scale the emulation of malware behavior to cover its frequent modifications employed by malware authors, we proposed a novel approach to classification of unknown binaries as legitimate or malicious (see Chapter 5) and to clustering of the malicious ones into coherent groups (see Chapter 6). The network traffic of these categorized binaries along with their manually estimated threat levels then represent malicious challenges used to estimate the parameters of the game-theoretical model. Such automation allows us to frequently update the database of malicious network behavior so that the IDS adapts to the latest threats.

7.1 Key contributions of this thesis

In this section we summarize the main contribution of this thesis to the state-of-the-art of network security:

1. *Design and implementation of dynamically defined two-player security game between attacker and defender.* In Chapter 3 we have proposed two-player single-stage game with complex utility function that reflects both attacker and defender incentives and allows us to precisely model their goals. Next, we have proposed an indirect online integration approach that enables us to dynamically define and solve the proposed security game using current characteristics of the background traffic and the state of the IDS. We have experimentally evaluated the influence of the game-theoretical approach on real-world IDS using real network dataset and shown that the proposed solution outperforms the trust-based baseline solution and thus successfully addresses the research problem *RP1* of this thesis.

2. *Design and implementation of an approach for simulation of behavior of legitimate user in high-level NetFlow format that can be used for evaluation and configuration of anomaly-detection-based IDS system.* In Chapter 4 we have introduced the *time variant joint probability model* that is designed to capture complex aspects of the user's network traffic such as inter- and intra-flow relations and sequential character of user's behavior. The experimental evaluation of the proposed solution has shown that the simulated data properly mimics the real traffic so that anomaly detection algorithms used in real-world IDS are not able to distinguish it. Another key difference between our approach and most of the prior art is that we are able to construct the data in memory with much lower overhead compared to approaches that adopt simulation of the complete network. This makes the proposed solution well-suited for runtime adaptation of an IDS and thus successfully addresses the research problem *RP2a* of this thesis.

3. *Design and implementation of novel approach for classification and clustering of malware samples using dynamic analysis.* In this thesis we have proposed novel approach for modeling of malware behavior based on malware's interactions with system resources recorded in sandbox. The model is based on assumption that malware's actions involve interactions with system resources, namely *files* (e.g. encryption of files in the case of ransomware), *registry keys* (e.g. to ensure persistence after reboot), *mutexes* (e.g. to ensure that only single instance of the malware is running), *network resources* (e.g. remote servers used for command-and-control or exfiltration of stolen data), and *error messages produced by the operating system* (e.g. error logged when application crashes). Since the number of system resources the malware interacts with can be different for every binary, we have proposed an approach based on *multiple instance learning* that is specifically designed to handle such data. The proposed model was used in two different scenarios: (1) malware classification (see Chapter 5), where the goal was to separate malware and benign applications, and (2) clustering of malware samples into groups that exhibited coherent behavior (see Chapter 6). The experimental evaluation performed on large set of real-world binaries proved that the proposed solution outperforms the current state-of-the-art approaches in both two-class classification and clustering. Moreover, the design of the proposed model allows us to promote clusters important for human analysts and to extract and to prioritize behavioral indicators to further speed up the manual analysis. As such it helps to automate the manual analysis of new samples and helps administrators of an adaptive IDS to keep the database of challenges up-to-date and thus addresses the research problem *RP2b* of this thesis.

7.2 List of author's publications

This section summarizes the author's publications related to this thesis.

Journal articles with impact factor (4)

1. Jan Stiborek, Tomáš Pevný, Martin Rehák. Multiple Instance Learning for Malware Classification. In: *Expert Systems with Applications*, Impact factor: 3.93, revision submitted. (70%)

2. Jan Stiborek, Tomáš Pevný, Martin Rehák. Probabilistic analysis of dynamic malware traces. In: *Computers & Security*, Impact factor: 2.85, revision submitted. (70%)

3. Ján Jusko, Martin Rehák, Jan Stiborek, Jan Kohout, Tomáš Pevný. Using Behavioral Similarity for Botnet Command-and-Control Discovery. In: *IEEE Intelligent Systems*. 2016, 31(5), pages 16–22. Impact factor 2.37 (15%)

4. Sebastian Garcia, Martin Grill, Jan Stiborek, Alejandro Zunino. An empirical comparison of botnet detection methods. Computers & Security, Volume 45, September 2014, pages 100-123. Impact factor 1.03. (5%)

5. Martin Rehák, Michal Pěchouček, Martin Grill, Jan Stiborek, Karel Bartoš, Pavel Čeleda. Adaptive multiagent system for network traffic monitoring. In: *IEEE Intelligent Systems*, vol. 24, no. 3, pages 16–25, May-June 2009. Impact factor 3.14 (16%)

Peer-reviewed journal articles (1)

1. Jan Stiborek, Martin Grill, Martin Rehák, Karel Bartoš and Ján Jusko. Game Theoretical Model for Adaptive Intrusion Detection System. In: *Transactions on Computational Collective Intelligence*, vol. 15, pages 133—163, 2014. (40%)

Patents (1)

1. Jan Stiborek, Martin Rehák: Sample Selection for Data Analysis for Use in Malware Detection. US14934398, submitted 6.11.2015. (*Allowed by US Patent Office, submitted with non-publication request*)

In ISI proceedings (4)

1. Jan Stiborek, Martin Grill, Martin Rehák, Karel Bartoš and Ján Jusko. Game Theoretical Adaptation Model for Intrusion Detection System. In: *Proceedings of the 10th Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS)*, pages 201–210, Springer Berlin, 2012. (20%)

2. Jan Stiborek, Martin Grill, Martin Rehák, Karel Bartoš and Ján Jusko. Game Theoretical Adaptation Model for Intrusion Detection System – Demo Paper. In: *Proceedings of the 10th Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS)*, pages 291—294, Springer Berlin. (20%)

3. Martin Rehák, Eugen Staab, Volker Fusenig, Michal Pěchouček, Martin Grill, Jan Stiborek, Karel Bartoš and Thomas Engel. Threat-model-driven runtime adaptation and

evaluation of intrusion detection system. In: *Proceedings of the 6th International Conference on Autonomic Computing and Communications*, pages 65–66, ACM Press, 2009. (14%)

4. Martin Rehák, Eugen Staab, Volker Fusenig, Michal Pěchouček, Martin Grill, Jan Stiborek, Karel Bartoš and Thomas Engel. Runtime Monitoring and Dynamic Reconfiguration for Intrusion Detection Systems. In: *Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection (RAID)*, pages 61–80, Springer Heidelberg, 2009. (14%)

In other proceedings (8)

1. Jan Stiborek, Martin Rehák, Tomáš Pevný. Towards Scalable Network Host Simulation. In: *Proceedings of the International Conference on Autonomus Agents & Multiagent Systems* (IFAAMAS). 2015. (41%)

2. Viliam Lisý, Radek Píbil, Jan Stiborek, Branislav Bošanský, Michal Pěchouček. Game-theoretic Approach to Adversarial Plan Recognition. In: *Proceedings of the 20th European Conference on Artificial Intelligence*, pages 546—551, IOS Press. 2012. (15%)

3. Martin Rehák, Michal Pěchouček, Martin Grill, Jan Stiborek and Karel Bartoš. Game Theoretical Adaptation Model for Intrusion Detection System, In: *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1123–1124, 2011. (20%)

4. Martin Rehák, Jan Stiborek and Martin Grill. Intelligence, not Integration: Distributed Regret Minimization for IDS Control. In: *IEEE Symposium on Computational Intelligence in Cyber Security*, pages 217–224, IEEE, 2011. (33%)

5. Martin Rehák, Martin Grill and Jan Stiborek. On the Value of Coordination in Distributed Self-Adaptation of Intrusion Detection System. In: *Proceedings Web Intelligence and Intelligent Agent Technology WI-IAT11*, pages 196–203, IEEE, 2011. (33%)

6. Martin Grill, Martin Rehák, and Jan Stiborek. Strategic self-organization methods for intrusion detection systems. In: *Proceedings of the Conference Security and Protection of Information, University of Defense*, Brno, 2011. (33%)

7. Martin Rehák, Karel Bartoš, Martin Grill, Jan Stiborek and Michal Svoboda. Monitoring síťí pomocí NetFlow dat—od paketů ke strategiím. In: *Proceedings of Česká společnost uživatelů otevřených systémů EurOpen.CZ*, pages 75–81, 2009. (20%)

8. Martin Rehák, Eugen Staab, Michal Pěchouček, Jan Stiborek, Martin Grill and Karel Bartoš. Dynamic Information Source Selection for Intrusion Detection Systems. In: *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1009–1016, ACM Press, 2009. (15%)

Bibliography

[1] K. A. Scarfone and P. M. Mell, "Guide to intrusion detection and prevention systems (idps)," *NIST Special Publication*, vol. 800-94, p. 127, 2007.

[2] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, "Network Anomaly Detection: Methods, Systems and Tools," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 303–336, 2014.

[3] P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Fernández, and E. Vázquez, "Anomaly-based network intrusion detection: Techniques, systems and challenges," *Computers & Security*, vol. 28, no. 1-2, pp. 18–28, feb 2009.

[4] M. Grill and T. Pevný, "Learning combination of anomaly detectors for security domain," *Computer Networks*, vol. 107, pp. 55–63, oct 2016.

[5] M. Rehák, E. Staab, M. Pěchouček, J. Stiborek, M. Grill, and K. Bartoš, "Dynamic information source selection for intrusion detection systems," in *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*. International Foundation for Autonomous Agents and Multiagent Systems, 2009, pp. 1009–1016.

[6] M. Rehák, M. Pěchouček, M. Grill, J. Stiborek, K. Bartoš, and P. Čeleda, "Adaptive multiagent system for network traffic monitoring," *IEEE Intelligent Systems*, vol. 24, no. 3, pp. 16–25, 2009.

[7] I. Corona, G. Giacinto, and F. Roli, "Adversarial attacks against intrusion detection systems: Taxonomy, solutions and open issues," *Information Sciences*, vol. 239, pp. 201–225, aug 2013.

[8] N. Nisan, T. Roughgarden, E. Tardos, and V. V. Vazirani, *Algorithmic game theory*. Cambridge University Press, 2007.

[9] Y. Liu, C. Comaniciu, and H. Man, "A bayesian game approach for intrusion detection in wireless ad hoc networks," in *GameNets '06: Proceeding from the 2006 workshop on Game theory for communications and networks*. New York, NY, USA: ACM, 2006, p. 4.

[10] Q. Zhu and T. Başar, "Dynamic policy-based IDS configuration," *Joint 48th IEEE Conference on Decision and Control and 28th Chinese Control Conference*, pp. 8600–8605, 2009.

[11] Q. Zhu, H. Tembine, and T. Başar, "Network Security Configurations: A Nonzero-Sum Stochastic Game Approach," in *Proceedings of the 2010 American Control Conference*. IEEE, jun 2010, pp. 1059–1064.

[12] Q. Zhu and T. Başar, "Indices of Power in Optimal IDS Default Configuration: Theory and Examples," *Proceedings of the 2nd Conference on Decision and Game Theory (GameSec)*, pp. 7–21, oct 2011.

[13] C. V. Zhou, S. Karunasekera, and C. Leckie, "A Peer-to-Peer Collaborative Intrusion Detection System," in *2005 13th IEEE International Conference on Networks Jointly held with the 2005 IEEE 7th Malaysia International Conf on Communic*, vol. 1. IEEE, 2005, pp. 118–123.

[14] H. Moosavi and F. M. Bui, "A discounted stochastic game approach to intrusion detection in wireless ad hoc networks," *2014 IEEE Fifth International Conference on Communications and Electronics (ICCE)*, pp. 536–541, 2014.

[15] ——, "A Game-Theoretic Framework for Robust Optimal Intrusion Detection in Wireless Sensor Networks," *IEEE Transactions on Information Forensics and Security*, vol. 9, no. 9, pp. 1367–1379, 2014.

[16] H. Ringberg, M. Roughan, and J. Rexford, "The need for simulation in evaluating anomaly detectors," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 1, p. 55, 2008.

[17] S. Abt, R. Stampp, and H. Baier, "Towards reproducible cyber-security research through complex node automation," in *2015 7th International Conference on New Technologies, Mobility and Security (NTMS)*. IEEE, jul 2015, pp. 1–5.

[18] J. Sonchack, A. J. Aviv, and J. M. Smith, "Bridging the Data Gap: Data Related Challenges in Evaluating Large Scale Collaborative Security Systems," in *Presented as part of the 6th Workshop on Cyber Security Experimentation and Test*, 2013.

[19] J. Sonchack and A. Aviv, "LESS Is More: Host-Agent Based Simulator for Large-Scale Evaluation of Security Systems," *Computer Security-ESORICS 2014*, pp. 365–382, 2014.

[20] S. Abt and H. Baier, "Are We Missing Labels? A Study of the Availability of Ground-Truth in Network Security Research," in *2014 Third International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS)*. IEEE, sep 2014, pp. 40–55.

[21] K. V. Vishwanath and A. Vahdat, "Realistic and responsive network traffic generation," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 4, p. 111, 2006.

[22] Cisco Systems, "Cisco IOS NetFlow," http://www.cisco.com/go/netflow, 2007.

[23] P. Barford and D. Plonka, "Characteristics of network traffic flow anomalies," *Proceedings of the First ACM SIGCOMM Workshop on Internet Measurement Workshop - IMW '01*, 2001.

[24] N. Falliere, L. O. Murchu, and E. Chien, "W32.Stuxnet Dossier analysis report," Tech. Rep., 2011.

[25] J. Amores, "Multiple instance classification: Review, taxonomy and comparative study," *Artificial Intelligence*, vol. 201, pp. 81–105, aug 2013.

[26] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[27] C. M. Bishop, *Pattern recognition and machine learning*. Springer, 2011, vol. 4.

[28] J. Stiborek, M. Grill, M. Rehák, K. Bartoš, and J. Jusko, "Game Theoretical Adaptation Model for Intrusion Detection System," in *Advances on Practical Applications of Agents and Multi-Agent Systems*, ser. Advances in Intelligent and Soft Computing, 2012, vol. 155, pp. 201–210.

[29] ——, "Game Theoretical Model for Adaptive Intrusion Detection System," in *Transactions on Computational Collective Intelligence XV*. Springer, Berlin, Heidelberg, 2014, vol. 2, pp. 133–163.

[30] J. Stiborek and M. Rehák, "Towards scalable network host simulation," in *ACySE 2015: Second International Workshop on Agents and CyberSecurity Towards*, 2015, pp. 27–35.

[31] J. Stiborek, T. Pevný, and M. Rehák, "Multiple Instance Learning for Malware Classification," *Expert Systems with Applications*, 2017, in review.

[32] ——, "Probabilistic analysis of dynamic malware traces," *Computers & Security*, 2017, in review.

[33] C. Zhang, A. X. Jiang, M. B. Short, P. J. Brantingham, and M. Tambe, "Defending Against Opportunistic Criminals: New Game-Theoretic Frameworks and Algorithms," in *International Conference on Decision and Game Theory for Security*, vol. 8840, 2014, pp. 3–22.

[34] Y. W. Law, T. Alpcan, and M. Palaniswami, "Security Games for Risk Minimization in Automatic Generation Control," *IEEE Transactions on Power Systems*, vol. 30, no. 1, pp. 223–232, 2015.

[35] M. M. H. Manshaei, Q. Zhu, T. Alpcan, T. Başar, and J.-P. Hubaux, "Game theory meets network security and privacy," *ACM Computing Surveys*, vol. 45, no. 3, pp. 1–39, 2013.

[36] X. Liang and Y. Xiao, "Game Theory for Network Security," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 1, pp. 472–486, 2013.

[37] T. Alpcan and T. Başar, "A game theoretic approach to decision and analysis in network intrusion detection," in *42nd IEEE International Conference on Decision and Control*, vol. 3. IEEE, 2003, pp. 2595–2600.

[38] ——, "A game theoretic analysis of intrusion detection in access control systems," in *2004 43rd IEEE Conference on Decision and Control (CDC) (IEEE Cat. No.04CH37601)*, vol. 2. IEEE, 2004, pp. 1568–1573 Vol.2.

[39] ——, "An Intrusion Detection Game with Limited Observations," in *12th International Symposium on Dynamic Games and Applications*, vol. 1, 2006, pp. 1–3.

[40] Y. B. Reddy, "A Game Theory Approach to Detect Malicious Nodes in Wireless Sensor Networks," in *2009 Third International Conference on Sensor Technologies and Applications*, no. June 2009. IEEE, 2009, pp. 462–468.

[41] K. C. Nguyen, T. Alpcan, and T. Başar, "Security Games with Incomplete Information," in *2009 IEEE International Conference on Communications*. IEEE, jun 2009, pp. 1–6.

[42] V. Lisý, R. Píbil, J. Stiborek, B. Bošanský, and M. Pěchouček, "Game-theoretic approach to adversarial plan recognition," *Frontiers in Artificial Intelligence and Applications*, vol. 242, pp. 546–551, 2012.

[43] Q. Zhu and T. Başar, "Game-Theoretic Approach to Feedback-Driven Multi-stage Moving Target Defense," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2013, vol. 8252 LNCS, pp. 246–263.

[44] L. Chen and J. Leneutre, "A Game Theoretical Framework on Intrusion Detection in Heterogeneous Networks," *IEEE Transactions on Information Forensics and Security*, vol. 4, no. 2, pp. 165–178, 2009.

[45] R. Jin, X. He, and H. Dai, "Collaborative IDS configuration: A two-layer game-theoretical approach," *2016 IEEE Global Communications Conference, GLOBECOM 2016 - Proceedings*, 2016.

[46] V. Yegneswaran, P. Barford, and S. Jha, "Global Intrusion Detection in the DOMINO Overlay System," in *In Proceedings of Network and Distributed System Security Symposium (NDSS)*, 2004.

[47] K. Leyton-Brown and Y. Shoham, *Essentials of Game Theory: A Concise Multidisciplinary Introduction*, R. J. Y. R. Brachman and T. O. S. U. Dietterich, Eds. Morgan & Claypool, jan 2008, vol. 2, no. 1.

[48] J. Hu and M. P. Wellman, "Nash Q-Learning for General-Sum Stochastic Games Junling," *Journal of Machine Learning Research*, vol. 4, no. 6, pp. 1039–1069, jan 2003.

[49] S. Floyd and V. Paxson, "Difficulties in simulating the Internet," *IEEE/ACM Transactions on Networking (TON)*, vol. 9, no. 4, pp. 392–403, 2001.

[50] A. Shiravi, H. Shiravi, M. Tavallaee, and A. a. Ghorbani, "Towards developing a systematic approach to generate benchmark datasets for intrusion detection," *Computers & Security*, pp. 1–18, 2012.

[51] D. Brauckhoff and A. Wagner, "FLAME: a flow-level anomaly modeling engine," in *The conference on Cyber security*, 2008.

[52] A. Sperotto, R. Sadre, P.-t. D. Boer, and A. Pras, "Hidden Markov Model modeling of SSH brute-force attacks," *9th IEEE International Workshop on IP Operations and Management (IPOM 09)*, p. 13, 2009.

[53] "The network simulator – NS-2," http://nsnam.isi.edu/nsnam, [Online; accessed June 20th, 2012].

[54] T. Henderson and M. Lacage, "Network simulations with the ns-3 simulator," *SIGCOMM*, p. 2006, 2008.

[55] R. Bye, S. Schmidt, K. Luther, and S. Albayrak, "Application-level simulation for network security," *Proceedings of the First International ICST Conference on Simulation Tools and Techniques for Communications, Networks and Systems*, 2008.

[56] D. Grunewald, R. Bye, K. Bsufka, and S. Albayrak, "Agent-based Network Security Simulation (Demonstration)," *AAMAS*, pp. 1325–1326, 2011.

[57] A. Varga and R. Hornig, "An overview of the OMNet++ simulation environment," *Proceedings of the First International ICST Conference on Simulation Tools and Techniques for Communications Networks and Systems*, 2008.

[58] B. Lantz, B. Heller, and N. McKeown, "A Network in a Laptop: Rapid Prototyping for Software-Defined Networks," in *Proceedings of the Ninth ACM SIGCOMM Workshop on Hot Topics in Networks - Hotnets '10*. New York, New York, USA: ACM Press, 2010, pp. 1–6.

[59] C. Michel and L. Mé, "ADeLe: an attack description language for knowledge-based intrusion detection," *of the 16th International Conference on*, pp. 353–368, 2001.

[60] A. Wagner and B. Plattner, "Entropy Based Worm and Anomaly Detection in Fast IP Networks," in *Enabling Technologies: Infrastructure for Collaborative Enterprise, 2005. 14th IEEE International Workshops on*, 2005.

[61] J. Zhang, P. Porras, and J. Ullrich, "Highly Predictive Blacklisting." in *USENIX Security Symposium*, 2008.

[62] B. Coskun, S. Dietrich, and N. Memon, "Friends of an enemy: identifying local members of peer-to-peer botnets using mutual contacts," in *Proceedings of the 26th Annual Computer Security Applications Conference. ACM*, 2010, pp. 131–140.

[63] N. Boggs, S. Hiremagalore, A. Stavrou, and S. J. Stolfo, "Cross-domain collaborative anomaly detection: So far yet so close," *Lecture Notes in Computer Science*, vol. 6961, pp. 142–160, 2011.

[64] B. Plattner, "DDoSVax," http://www.tik.ee.ethz.ch/~ddosvax/, 2005, [Online; accessed August 3rd, 2012].

[65] P. Levis, N. Lee, M. Welsh, and D. Culler, "TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications," in *Proceedings of the first international conference on Embedded networked sensor systems - SenSys '03*. New York, New York, USA: ACM Press, 2003, p. 126.

[66] J. Banks, J. Carson, B. Nelson, and D. Nicol, *Discrete-event system simulation*. Prentice Hall Upper Saddle River, NJ, 2001.

[67] J. Banks, *Handbook of simulation: Principles, methodology, advances, applications, and practice*. John Wiley & Sons, 1998.

[68] S. Bajaj, L. Breslau, D. Estrin, K. Fall, S. Floyd, P. Haldar, M. Handley, A. Helmy, and Y. Xu, "Improving Simulation for Network Research," Tech. Rep., 1999.

[69] S. Keshav, *REAL: A network simulator*, Berkeley, Calif, USA, 1988.

[70] I. Vari, "INET Framework for OMNeT++," http://inet.omnetpp.org/, 2008, [Online; accessed August 14th, 2012].

[71] I. Baumgart, B. Heep, and S. Krause, "OverSim: A scalable and flexible overlay framework for simulation and real network applications," *2009 IEEE Ninth International Conference on Peer-to-Peer Computing*, pp. 87–88, sep 2009.

[72] B. Hirsch, T. Konnerth, and A. Heßler, "Merging Agents and Services - the JIAC Agent Platform," in *Multi-Agent Programming*, A. El Fallah Seghrouchni, J. Dix, M. Dastani, and R. H. Bordini, Eds. Boston, MA: Springer US, 2009, ch. 5, pp. 159–185.

[73] R. W. Lo, K. N. Levitt, and R. A. Olsson, "MCF: a malicious code filter," *Computers & Security*, vol. 14, no. 6, pp. 541–566, jan 1995.

[74] M. Christodorescu and S. Jha, "Static Analysis of Executables to Detect Malicious Patterns," Computer Sciences Department University of Wisconsin, Madison, Tech. Rep., 2006.

[75] S. Cesare and Y. Xiang, "Classification of malware using structured control flow," in *Conferences in Research and Practice in Information Technology Series*, vol. 107, 2010, pp. 61–70.

[76] J. Kinable and O. Kostakis, "Malware Classification based on Call Graph Clustering," *Journal in computer virology 7.4*, pp. 233–245, 2011.

[77] D. Kong and G. Yan, "Discriminant malware distance learning on structural information for automated malware classification," in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '13*. New York, New York, USA: ACM Press, 2013, p. 1357.

[78] I. Santos, Y. K. Penya, J. Devesa, and P. G. Bringas, "N-grams-based File Signatures for Malware Detection." in *Proceedings of the 11th International Conference on Enterprise Information Systems*, 2009, pp. 317–320.

[79] D. K. S. Reddy and A. K. Pujari, "N-gram analysis for computer virus detection," *Journal in Computer Virology*, vol. 2, no. 3, pp. 231–239, nov 2006.

[80] M. Ahmadi, D. Ulyanov, S. Semenov, M. Trofimov, and G. Giacinto, "Novel Feature Extraction, Selection and Fusion for Effective Malware Family Classification," in *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*, nov 2016, pp. 183–194.

[81] S. Naval, V. Laxmi, M. Rajarajan, M. S. Gaur, and M. Conti, "Employing Program Semantics for Malware Detection," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 12, pp. 2591–2604, dec 2015.

[82] T. Wüchner, M. Ochoa, and A. Pretschner, "Malware detection with quantitative data flow graphs," *Proceedings of the 9th ACM symposium on Information, computer and communications security - ASIA CCS '14*, pp. 271–282, feb 2014.

[83] C. Kolbitsch, P. M. Comparetti, C. Kruegel, E. Kirda, X. Zhou, X. Wang, U. C. S. Barbara, and S. Antipolis, "Effective and Efficient Malware Detection at the End Host," in *USENIX Security Symposium*, vol. 4, no. 1, jan 2009, pp. 351–366.

[84] Y. Park, D. Reeves, V. Mulukutla, and B. Sundaravel, "Fast malware classification by automated behavioral graph matching," in *Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research - CSIIRW '10*, 2010, p. 1.

[85] J. Pfoh, C. Schneider, and C. Eckert, "Leveraging String Kernels for Malware Detection," in *International Conference on Network and System Security*, 2013, vol. 7873 LNCS, pp. 206–219.

[86] A. Lanzi, D. Balzarotti, C. Kruegel, M. Christodorescu, and E. Kirda, "AccessMiner: Using System-Centric Models for Malware Protection," in *ACM Conference on Computer and Communications Security 2010*, 2010, pp. 399–412.

[87] R. Canzanese, M. Kam, and S. Mancoridis, "Toward an Automatic, Online Behavioral Malware Classification System," in *2013 IEEE 7th International Conference on Self-Adaptive and Self-Organizing Systems*. IEEE, sep 2013, pp. 111–120.

[88] R. Canzanese, S. Mancoridis, and M. Kam, "Run-time classification of malicious processes using system call analysis," in *2015 10th International Conference on Malicious and Unwanted Software, MALWARE 2015*, no. May, 2015, pp. 21–28.

[89] K. Rieck, P. Trinius, C. Willems, and T. Holz, "Automatic analysis of malware behavior using machine learning," *Journal of Computer Security*, vol. 19, no. 4, pp. 639–668, jun 2011.

[90] U. Bayer, P. P. M. Comparetti, C. Hlauschek, C. Kruegel, and E. Kirda, "Scalable, Behavior-Based Malware Clustering." in *Proceedings of the 16th Annual Network and Distributed System Security Symposium (NDSS 2009)*, 2009.

[91] R. S. Pirscoveanu, S. S. Hansen, T. M. T. Larsen, M. Stevanovic, J. M. Pedersen, and A. Czech, "Analysis of malware behavior: Type classification using machine learning," in *2015 International Conference on Cyber Situational Awareness, Data Analytics and Assessment (CyberSA)*. IEEE, jun 2015, pp. 1–7.

[92] A. Mohaisen, O. Alrawi, and M. Mohaisen, "AMAL: High-fidelity, behavior-based automated malware analysis and classification," *Computers & Security*, vol. 52, pp. 251–266, jul 2015.

[93] A. Mohaisen, A. G. West, A. Mankin, and O. Alrawi, "Chatter: Exploring Classification of Malware based on the Order of Events," *IEEE CNS*, 2013.

[94] K. Rieck, T. Holz, C. Willems, P. Dussel, and P. Laskov, "Learning and classification of malware behavior," in *DIMVA 2008*, 2008, pp. 108–125.

[95] M. Bailey, J. Oberheide, J. Andersen, Z. M. Mao, F. Jahanian, and J. Nazario, "Automated Classification and Analysis of Internet Malware," *RAID*, 2007.

[96] B. Anderson, C. Storlie, and T. Lane, "Improving malware classification," in *Proceedings of the 5th ACM workshop on Security and artificial intelligence - AISec '12*. New York, New York, USA: ACM Press, 2012, p. 3.

[97] A. Moser, C. Kruegel, and E. Kirda, "Limits of static analysis for malware detection," in *Proceedings - Annual Computer Security Applications Conference, ACSAC*. IEEE, dec 2007, pp. 421–430.

[98] M. Sharif, V. Yegneswaran, H. Saidi, P. Porras, and W. Lee, "Eureka: A framework for enabling static malware analysis," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 5283 LNCS, 2008, pp. 481–500.

[99] T. Chen and C. Guestrin, "XGBoost: a scalable tree boosting system," in *22nd SIGKDD Conference on Knowledge Discovery and Data Mining*, 2016.

[100] S. S. Hansen, T. M. T. Larsen, M. Stevanovic, and J. M. Pedersen, "An approach for detection and family classification of malware based on behavioral analysis," in *2016 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, feb 2016, pp. 1–5.

[101] C. Kolbitsch, T. Holz, C. Kruegel, and E. Kirda, "Inspector Gadget: Automated Extraction of Proprietary Gadgets from Malware Binaries," *2010 IEEE Symposium on Security and Privacy*, pp. 29–44, 2010.

[102] W. Ma, P. Duan, S. Liu, G. Gu, and J. C. Liu, "Shadow attacks: Automatically evading system-call-behavior based malware detection," *Journal in Computer Virology*, vol. 8, no. 1-2, pp. 1–13, 2012.

[103] G. S. Kc, A. D. Keromytis, and V. Prevelakis, "Countering code-injection attacks with instruction-set randomization," in *Proceedings of the 10th ACM conference on Computer and communication security - CCS '03*. New York, New York, USA: ACM Press, 2003, p. 272.

[104] D. R. Garcia, "AntiCuckoo." [Online]. Available: https://github.com/David-Reguera-Garcia-Dreg/anticuckoo

[105] B. Anderson, D. Quist, J. Neil, C. Storlie, and T. Lane, "Graph-based malware detection using dynamic analysis," *Journal in Computer Virology*, vol. 7, no. 4, pp. 247–258, nov 2011.

[106] M. Gonen and E. Alpaydin, "Multiple Kernel Learning Algorithms," *Journal of Machine Learning Research*, vol. 12, pp. 2211–2268, 2011.

[107] H. G. Kayacik and A. N. Zincir-Heywood, "Mimicry attacks demystified: What can attackers do to evade detection?" *Privacy, Security and Trust, Annual Conference on*, vol. 0, pp. 213–223, 2008.

[108] B. I. P. Rubinstein, B. Nelson, L. Huang, A. D. Joseph, S. hon Lau, N. Taft, and J. D. Tygar, "Evading anomaly detection through variance injection attacks on pca," in *Recent Advances in Intrusion Detection, 11th International Symposium, RAID 2008, Cambridge, MA, USA, September 15-17, 2008. Proceedings*, ser. Lecture Notes in Computer Science, R. Lippmann, E. Kirda, and A. Trachtenberg, Eds., vol. 5230. Springer, 2008, pp. 394–395.

[109] M. Barreno, B. Nelson, R. Sears, A. D. Joseph, and J. D. Tygar, "Can machine learning be secure?" in *ASIACCS '06: Proceedings of the 2006 ACM Symposium on Information, computer and communications security*. New York, NY, USA: ACM, 2006, pp. 16–25.

[110] J. Stiborek, "Game Theoretical Approach to Adaptive Intrusion Detection System," Faculty of Nuclear Sciences and Physical Engineering, Czech Technical University in Prague, Tech. Rep., 2010.

[111] D. Fudenberg and J. Tirole, *Game Theory*. MIT Press, 1991.

[112] M. Rehák, E. Staab, V. Fusenig, M. Pěchouček, M. Grill, J. Stiborek, K. Bartoš, and T. Engel, "Runtime monitoring and dynamic reconfiguration for intrusion detection systems," in *Recent Advances in Intrusion Detection, 12th International Symposium, RAID 2009, Saint-Malo, France, September 23-25, 2009. Proceedings*, E. Kirda, S. Jha, and D. Balzarotti, Eds., 2009, pp. 61–80.

[113] G. S. Becker, "Crime and punishment: An economic approach," *The Journal of Political Economy*, vol. 76, no. 2, pp. 169–217, 1968.

[114] T. H. Ptacek and T. N. Newsham, "Insertion, evasion, and denial of service: Eluding network intrusion detection," Secure Networks, Inc., Tech. Rep., 1998.

[115] A. Blum and Y. Mansour, "learning, regret minimization and equilibria," in *Algorithmic Game Theory*, N. Nisan, T. Roughgarden, E. Tardos, and V. Vazirani, Eds. Cambridge University Press, 2007, ch. 4, pp. 79–101.

[116] G. Wagener, R. State, A. Dulaunoy, and T. Engel, "Self adaptive high interaction honeypots driven by game theory," in *SSS*, 2009, pp. 741–755.

[117] A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A. Pras, and B. Stiller, "An Overview of IP Flow-Based Intrusion Detection," *IEEE Communications Surveys & Tutorials*, vol. 12, no. 3, pp. 343–356, 2010.

[118] B. Li, J. Springer, G. Bebis, and M. H. Gunes, "A survey of network flow applications," *Journal of Network and Computer*, vol. 36, no. 2, pp. 567–581, 2013.

[119] R. Hofstede, P. Celeda, B. Trammell, I. Drago, R. Sadre, A. Sperotto, A. Pras, P. Čeleda, B. Trammell, I. Drago, R. Sadre, A. Sperotto, and A. Pras, "Flow monitoring explained: From packet capture to data analysis with NetFlow and IPFIX," *IEEE Communications Surveys and Tutorials*, vol. 16, no. 4, pp. 2037–2064, 2014.

[120] A. Sperotto, M. Mandjes, R. Sadre, P.-t. D. Boer, and A. Pras, "Autonomic Parameter Tuning of Anomaly-Based IDSs: an SSH Case Study," *IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT*, vol. 9, no. 2, pp. 128–141, 2012.

[121] J. Sommers, H. Kim, and P. Barford, "Harpoon," in *Proceedings of the joint international conference on Measurement and modeling of computer systems - SIGMETRICS 2004/PERFORMANCE 2004*. New York, New York, USA: ACM Press, 2004, p. 392.

[122] IXIA, "Breaking point," https://www.ixiacom.com/products/network-security-testing-breakingpoint, online; accessed August 1st, 2017.

[123] K. Murphy, *Machine Learning: a Probabilistic Perspective*, 2012.

[124] Y. Zhang, Z. Ge, A. Greenberg, and M. Roughan, "Network anomography," in *Proceedings of the 5th ACM SIGCOMM conference on Internet measurement - IMC '05*. New York, New York, USA: ACM Press, 2005, p. 1.

[125] F. Silveira, C. Diot, N. Taft, and R. Govindan, "ASTUTE: Detecting a Different Class of Traffic Anomalies," in *Proceedings of the ACM SIGCOMM 2010 conference on SIGCOMM - SIGCOMM '10*. New York, New York, USA: ACM Press, 2010, p. 267.

[126] D. Endres and J. Schindelin, "A new metric for probability distributions," *IEEE Transactions on Information Theory*, vol. 49, no. 7, pp. 1858–1860, jul 2003.

[127] T. Pevny, M. Rehak, and M. Grill, "Detecting anomalous network hosts by means of PCA," in *2012 IEEE International Workshop on Information Forensics and Security (WIFS)*. IEEE, dec 2012, pp. 103–108.

[128] A. Lakhina, M. Crovella, and C. Diot, "Diagnosing network-wide traffic anomalies," in *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications - SIGCOMM '04*. New York, New York, USA: ACM Press, 2004, p. 219.

[129] ——, "Mining anomalies using traffic feature distributions," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 4, p. 217, oct 2005.

[130] L. Ertöz, E. Eilertson, A. Lazarevic, P.-n. Tan, V. Kumar, J. Srivastava, and P. Dokas, "Minds-minnesota intrusion detection system," *Next Generation Data Mining*, pp. 199–218, 2004.

[131] M. Rehák, M. Pěchouček, K. Bartoš, M. Grill, and P. Čeleda, "Network Intrusion Detection by Means of Community of Trusting Agents," in *2007 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT'07)*. IEEE, nov 2007, pp. 498–504.

[132] K. Xu, Z. Zhang, and S. Bhattacharyya, "Reducing unwanted traffic in a backbone network," *Usenix Workshop on Steps to Reduce Un- wanted Traffic in the Internet (SRUTI 05)*, 2005.

[133] G. Inc., "VirusTotal.com." [Online]. Available: https://www.virustotal.com/

[134] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, "A survey on automated dynamic malware-analysis techniques and tools," *ACM Computing Surveys*, vol. 44, no. 2, pp. 1–42, feb 2012.

[135] J. Bremer, "x86 API Hooking Demystified," 2012, accessed July 26th, 2017. [Online]. Available: http://jbremer.org/x86-api-hooking-demystified/

[136] T. G. Dietterich, R. H. Lathrop, and T. Lozano-Pérez, "Solving the multiple instance problem with axis-parallel rectangles," *Artificial Intelligence*, vol. 89, no. 1-2, pp. 31–71, jan 1997.

[137] J. Sivic and A. Zisserman, "Video Google: a text retrieval approach to object matching in videos," in *Proceedings Ninth IEEE International Conference on Computer Vision*. IEEE, apr 2003, pp. 1470–1477 vol.2.

[138] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Soviet Physics Doklady*, vol. 10, no. 8, pp. 707–710, 1966.

[139] G. Navarro, "A guided tour to approximate string matching," *ACM Computing Surveys*, vol. 33, no. 1, pp. 31–88, mar 2001.

[140] H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins, "Text Classification using String Kernels," *Journal of Machine Learning Research*, vol. 2, pp. 419–444, 2002.

[141] C. Cortes, M. Mohri, and A. Rostamizadeh, "Algorithms for Learning Kernels Based on Centered Alignment," *The Journal of Machine Learning*, mar 2012.

[142] J. Kohout and T. Pevny, "Automatic discovery of web servers hosting similar applications," in *Proceedings of the 2015 IFIP/IEEE International Symposium on Integrated Network Management, IM 2015*, 2015, pp. 1310–1315.

[143] J. Jusko, M. Rehák, J. Stiborek, J. Kohout, and T. Pevný, "Using Behavioral Similarity for Botnet Command and Control Discovery," *IEEE Intelligent Systems*, 2016.

[144] C. D. Manning, P. Raghavan, and Schütze, *An Introduction to Information Retrieval*. Cambridge, England: Cambridge University Press, 2009, no. c.

[145] G. Salton, *Introduction to modern information retrieval*, 1983.

[146] J. Shawe-Taylor and N. Cristianini, *Kernel methods for pattern analysis*, 2004.

[147] Symantec Security Response, "Sality: Story of a Peer- to-Peer Viral Network," Symantec, Inc., Tech. Rep., 2011.

[148] G. V. Bard, "Spelling-Error Tolerant, Order-Independent Pass-Phrases via the Damerau-Levenshtein String-Edit Distance Metric," in *Research and Practice in Information Technology , (CRPIT), Vol. 68*. Australian Computer Society, Inc., 2007.

[149] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2008, no. 10, p. P10008, mar 2008.

[150] S. Fortunato, "Community detection in graphs," *Physics Reports*, vol. 486, no. 3-5, pp. 75–174, feb 2010.

[151] T. Zhang, R. Ramakrishnan, and M. Livny, "BIRCH: A New Data Clustering Algorithm and Its Applications," *Data Mining and Knowledge Discovery*, vol. 1, no. 2, pp. 141–182, 1997.

[152] A. Zimek, M. Gaudet, R. J. G. B. Campello, and J. Sander, "Subsampling for efficient and effective unsupervised outlier detection ensembles," in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '13*, 2013, p. 428.

[153] J. Jang, D. Brumley, and S. Venkataraman, "BitShred : Fast , Scalable Malware Triage," Tech. Rep., 2010.

[154] M. Sebastián, R. Rivera, P. Kotzias, and J. Caballero, "AVclass: A Tool for Massive Malware Labeling," in *Research in Attacks, Intrusions, and Defenses: 19th International Symposium, RAID 2016, Paris, France, September 19-21, 2016, Proceedings*, 2016, pp. 230–253.

[155] AV-Test, "Consumer Full Product Testing November/December 2016," AV-TEST GmbH, Tech. Rep., 2016. [Online]. Available: https://www.av-test.org/fileadmin/tests/homeuser/avtest{_}summary{_}homeuser{_}2016-12.xlsx

[156] C. S. Inc., "AMP ThreatGrid," 2017. [Online]. Available: http://www.cisco.com/c/en/us/solutions/enterprise-networks/amp-threat-grid/index.html

[157] D. Oktavianto and I. Muhardianto, *Cuckoo Malware Analysis*. Packt Publishing, 2013.

[158] A. Dinaburg, P. Royal, M. Sharif, and W. Lee, "Ether: Malware Analysis via Hardware Virtualization Extensions," in *Proceedings of the 15th ACM conference on Computer and communications security - CCS '08*. New York, New York, USA: ACM Press, 2008, p. 51.

[159] C. Willems, T. Holz, and F. Freiling, "Toward Automated Dynamic Malware Analysis Using CWSandbox," *IEEE Security and Privacy Magazine*, vol. 5, no. 2, pp. 32–39, mar 2007.

[160] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[161] W. M. Rand, "Objective Criteria for the Evaluation of Clustering Methods Objective Criteria for the Evaluation of Clustering Methods," *Journal of the American Statistical Association*, vol. 66, no. 336, pp. 846–850, 1971.

[162] T. Fawcett, "An introduction to ROC analysis," *Pattern Recognition Letters*, vol. 27, no. 8, pp. 861–874, jun 2006.

[163] B. Dai, S. Ding, and G. Wahba, "Multivariate Bernoulli distribution," *Bernoulli*, vol. 19, no. 4, pp. 1465–1483, sep 2013.

[164] A. Juan and E. Vidal, "On the use of Bernoulli mixture models for text classification," *Pattern Recognition*, vol. 35, no. 12, pp. 2705–2710, dec 2002.

[165] G. Govaert and M. Nadif, "Block clustering with Bernoulli mixture models: Comparison of different approaches," *Computational Statistics & Data Analysis*, vol. 52, no. 6, pp. 3233–3245, feb 2008.

[166] M. H. C. Law, M. A. T. Figueiredo, and A. K. Jain, "Simultaneous feature selection and clustering using mixture models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 9, pp. 1154–1166, 2004.

[167] D. Gamerman and F. L. Hedibert, *Markov Chain Monte Carlo: Stochastic Simulation for Bayesian Inference*, 2006.

[168] A. Dempster, N. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *Journal of the Royal Statistical Society Series B Methodological*, vol. 39, no. 1, pp. 1–38, 1977.

[169] A. K. Jain, "Data clustering: 50 years beyond K-means," *Pattern Recognition Letters*, vol. 31, no. 8, pp. 651–666, 2010.

[170] L. Hubert and P. Arabie, "Comparing partitions," *Journal of Classification*, vol. 2, no. 1, pp. 193–218, dec 1985.

[171] A. Rosenberg and J. Hirschberg, "V-measure: A conditional entropy-based external cluster evaluation measure," in *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, vol. 1, no. June, 2007, pp. 410–420.

[172] Yixin Chen, J. Wang, and R. Krovetz, "CLUE: cluster-based retrieval of images by unsupervised learning," *IEEE Transactions on Image Processing*, vol. 14, no. 8, pp. 1187–1201, aug 2005.

[173] Y. Zhao and K. George, "Criterion functions for document clustering: Experiments and analysis (Technical Report)," Tech. Rep., 2001.

[174] P. Chen, C. Huygens, L. Desmet, and W. Joosen, "Advanced or Not? A Comparative Study of the Use of Anti-debugging and Anti-VM Techniques in Generic and Targeted Malware," in *IFIP International Federation for Information Processing 2016*. Springer, Cham, may 2016, pp. 323–336.

[175] S. S. Response, "W32.Ramnit analysis," Tech. Rep., 2015.

[176] A. Aronow, "SilentRunners," 2017, accessed July 26th, 2017. [Online]. Available: http://www.silentrunners.org/launchpoints.html

[177] "Beyond good ol' Run key," Hexacorn Limited, 2017, accessed July 26th, 2017. [Online]. Available: http://www.hexacorn.com/blog/2017/01/28/beyond-good-ol-run-key-all-parts/

[178] "Andromeda under the microscope," AVAST Threat Intelligence Team, 2016, accessed July 26th, 2017. [Online]. Available: https://blog.avast.com/andromeda-under-the-microscope

[179] D. Müllner, "fastcluster : Fast Hierarchical, Agglomerative Clustering Routines for R and Python," *Journal of Statistical Software*, vol. 53, no. 9, pp. 1–18, 2013.

[180] W. E. Winkler, "String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage," in *Proceedings of the Section on Survey Research*, 1990, pp. 354–359.

[181] D. Bakkelund, "An LCS-based string metric," Oslo, Tech. Rep., 2009. [Online]. Available: http://hjem.ifi.uio.no/danielry/StringMetric.pdf

[182] L. R. Dice, "Measures of the Amount of Ecologic Association Between Species," *Ecology*, vol. 26, no. 3, pp. 297–302, jul 1945.

[183] G. Kondrak, "N-Gram Similarity and Distance," *Lecture Notes in Computer Science*, vol. 3772, pp. 115–126, 2005.

[184] N. Perdisci, Roberto and Lee, Wenke and Feamster, "Behavioral clustering of HTTP-based malware and signature generation using malicious network traces," in *Proceedings of the 7th USENIX conference on Networked systems design and implementation*. IEEE, 2010, pp. 26—-26.

[185] F. J. Damerau, "A technique for computer detection and correction of spelling errors," *Communications of the ACM*, vol. 7, no. 3, pp. 171–176, mar 1964.

[186] D. Defays, "An efficient algorithm for a complete link method," *The Computer Journal*, vol. 20, no. 4, pp. 364–366, apr 1977.

[187] R. Sibson, "SLINK: An optimally efficient algorithm for the single-link cluster method," *The Computer Journal*, vol. 16, no. 1, pp. 30–34, jan 1973.

[188] J. Sander, M. Ester, H.-P. Kriegel, and X. Xu, "Density-Based Clustering in Spatial Databases: The Algorithm GDBSCAN and its Applications," *Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 169–194, 1998.

Appendix A

Evaluation of system resources' similarity metrics

The evaluation of the MIL model proposed in Section 5.1 raises two principal questions that need to be discussed:

1. Are the proposed similarities and clustering algorithm the optimal choices?
2. Is the proposed approach scalable?

In order to answer the first question, we would ideally need to find the best possible combination of similarity and clustering algorithm for every type of system resource along with the optimal settings of their parameters using the two-class classification as a testbed. However, such exhaustive search is computationally infeasible as the number of combination grows exponentially with number of parameters we need to vary.

Instead of testing the whole stack, we estimate the clustering performance of various similarity metrics and clustering algorithms on the scenario of clustering of system resources. The idea is that if the resulting clusters correctly represent behavioral indicators they will be well-suited for malware classification since behavioral indicator specific for a malware family is optimal feature for classification. Moreover, this approximation gives us the opportunity to precisely assess the scalability of the proposed approach and thus answer the second question, whether the optimal combination of similarity metric and clustering algorithm scales to large datasets.

A.1 Compared metrics and clustering algorithms

Since system resources can be viewed as strings with specific structure, we compare proposed similarities with well-known string metrics [139] (see Table A.1 for complete list) with the only exception—the network traffic. Since there is a prior art related to computing similarity between two HTTP requests, we compare the proposed similarity to the similarity proposed by Perdisci et al. [184].

The strings metrics we have selected as a baseline fall into three main categories:

- Metrics based on *edit distance*,
- metrics modeling n-grams,
- string kernel.

Name	Metric?	Type	Complexity
Levenshtein dist. [138]	No	edit dist.	$O(\|x\| \cdot \|x'\|)$
Damerau-Levenshtein dist. [148]	Yes	edit dist.	$O(\|x\| \cdot \|x'\|)$
Jaro-Winkler distance [180]	No	edit dist.	$O(\|x\| \cdot \|x'\|)$
Longest Common Subsequence [181]	Yes	edit dist.	$O(\|x\| \cdot \|x'\|)$
Cosine similarity	No	n-gram	$O(\|x\| + \|x'\|)$
Jaccard similarity	Yes	n-gram	$O(\|x\| + \|x'\|)$
Sorensen-Dice coefficient [182]	No	n-gram	$O(\|x\| + \|x'\|)$
N-Gram [183]	No	n-gram	$O(\|x\| \cdot \|x'\|)$
String kernel [140]	Yes	subseq.	$O(\|x\| \cdot \|x'\|)$
Proposed similarity	No		$O(\max(\|\tilde{x}\|, \|\tilde{x'}\|))$

Table A.1: String metrics used for comparison with path similarity. Parameters $|x|$ and $|x'|$ denote lengths of compared paths and parameters $|\tilde{x}|$ and $|\tilde{x'}|$ denote number of fragments. Note that $|x|, |x'| \gg |\tilde{x}|, |\tilde{x'}|$ since the number of fragments is much lower than number of characters in file path.

Metrics based on *edit distance* compute the distance as the minimal number of character operations required to convert one string to another. The particular selection of operations is specific for each string metric (Levenshtein distance, Jaro-Winkler distance, Metric Longest Common Substring) but in general they include

1. *insertion* of new character (*kiten*→*ki**t**ten*),
2. *deletion* of a character (*ki**t**ten*→*kiten*),
3. *substitution* of a character (*kit**e**n*→*kit**a**n*) and
4. *transposition* of two adjacent characters (*k**it**ten*→*k**ti**ten*).

Originally, these string metrics were designed for correction of misspelled words, where individual operations model typical typing mistakes [185]. Another important application area is computational genetics where DNA and protein sequences are considered as long strings over specific alphabet (e.g. $\Sigma = \{A, T, C, G\}$ in case of DNA). The goal is then to determine the similarity of two sequences which can be used to reconstruct the evolution tree. In this scenario the character operations model the mutations of the analyzed sequence.

The second category of string metrics (Cosine similarity, Jaccard similarity, N-Gram) is based on modeling n-grams, continuous sequences of n characters. For examples, bigrams ($n = 2$) for word *kitten* are: *ki, it, tt, te, en.* The extracted n-grams are then typically modeled with bag-of-words model where every n-gram represents single feature with the value equal to the frequency of the n-gram in the string. The similarity between two strings is then defined as a similarity between two feature vectors corresponding to compared strings.

A specific category of string metrics are *string kernels*. In general, kernels are based on *feature mapping* $\phi(x)$ that projects the input data (in our case strings) into a vector space with potentially infinite dimension. The kernel function that represents the distance between strings x and x' is then defined as

$$k(x, x') = \phi(x)^T \phi(x').$$

The concept of kernels was successfully applied on many machine learning tasks since it allows to use various machine learning algorithms for non-numerical data (e.g. strings, graphs, images, etc.).

Since some of the string metrics are defined as distances rather than similarities we use following formula to compute the similarity required by the Louvaine method

$$s(x, x') = 1 - f(d(x, x')), \tag{A.1.1}$$

where $d(x, x')$ is the underlying string distance and $f(\cdot)$ is a function that normalize the value of the string distance $d(x, x')$ into $[0, 1]$ interval, typically defined as $\max(|x|, |x'|)$.

Similarity metric, however, is only one of the components necessary for clustering of system resources. The second component that affects the results is the clustering algorithm itself. Therefore, we need to verify that the proposed approximation of Louvaine clustering algorithm is suitable for clustering of system resources. We compare the results obtained with approximative Louvaine clustering defined in Section 5.1.6 (further referred as *ApproxL*) and four different state-of-the-art clustering algorithms that can be combined with similarities under comparison: hierarchical clustering with complete linkage (further referred as *CLINK*) [186], hierarchical clustering with single linkage (further referred as *SLINK*) [187] and generalized density-based clustering (further referred as *GBSCAN*) [188].

A.2 Dataset description

The performance of the similarities proposed in Section 5.1 was evaluated on four different datasets, one for every type of system resources (file paths, mutexes, registry keys, network traffic). The data were extracted from behavioral traits of malware samples originating from multiple malware families such as Zeus, Sality, Ramnit, Nemucod, CyberGate, Bifrost, DarkComet, Bedep, etc. These samples were analyzed using AMP ThreatGrid service [156] and submitted to manual analysis. Analyst then grouped system resources of individual types according to their purpose in the operating system or their structure such these groups represent behavioral indicators of corresponding malware families. The examples of groups from the file path dataset include

- files that resembles installation of Acrobat Reader (e.g. `/program files/adobe/reader 9.0/reader/acrobroker.exe`),
- files in Windows DLL cache (e.g. `/windows/system32/dllcache/spcplui.dll`),
- files in Start folder (e.g. `/documents and settings/all users/start menu/programs /games.exe`), etc.

The mutex dataset contains

- mutexes specific to a particular malware family (e.g. `DC_MUTEX-R4RAJJ3`),
- mutexes with similar structure (e.g. `D30D239201D1732D000007E82`), etc.

The registry key dataset includes

- registry keys that contains information about binaries that are executed after startup of the operating system (e.g. `HKLM\software\microsoft\windows\currentversion\run\ fbf674c6aed6196c206250415e53e947`),

	Training data		Testing data	
	#instances	#groups	#instances	#groups
Paths	5907	50	8907	52
Mutexes	2056	49	1021	45
Registry keys	5705	59	2782	78
Network	790	124	477	108

Table A.2: Numbers of instances and groups in training and testing datasets for individual system resources.

- keys that contains list of applications that are authorized to make network connections (e.g. `HKLM\system\controlset001\services\sharedaccess\parameters\firewall policy\standardprofile\authorizedapplications\list\c:\temp\1432671mgr.exe`).

The last dataset, the network resources dataset, contains traits of communication with various domains/IPs related to malware's command and control or monetization structures (e.g. `ubnsyhv27fa2j.ru`).

All four datasets were divided into training and testing data such that only part of the groups is present in both training and testing data (detailed numbers are summarized in Table A.2). Such split allows us to partially overcome the bias that is typically introduced when both training and testing data contains the same groups of instances. The training data were used for optimization of following parameters of the clustering algorithms:

- CLINK: $\epsilon = \{0, 0.025, 0.05, 0.075, \dots 1\} \cdot \text{maxValue}$,
- SLINK: $\epsilon = \{0, 0.025, 0.05, 0.075, \dots 1\} \cdot \text{maxValue}$,
- GDBSCAN: $\epsilon = \{0, 0.025, 0.05, 0.075, \dots 1\} \cdot \text{maxValue}, \text{minPts} = \{1, 2, 3, 4, 5\}$,
- approxL: $\epsilon = \{0, 0.025, 0.05, 0.075, \dots 1\} \cdot \text{maxValue}$,

where maxValue represents theoretical maximal value of particular similarity metric. Since the results of the approximative Louvaine clustering depends on the starting conditions we have repeated the clustering 5 times and used average value of ARI.

The performance of individual combinations of similarity metrics and clustering algorithms is evaluated with adjusted rand index (ARI) defined in Section 6.3.1. All metrics and clustering algorithms were implemented in Java 8 as single-threaded applications and the scalability was evaluated on Amazon AWS virtual machine (r4.x4large, 16-core Intel Xeon E5-2686 v4 Broadwell Processors with 122GB RAM). Each experiment was repeated 10times and the running times were averaged.

A.3 Comparison of proposed metrics with related work

In the first part of the experimental evaluation we evaluated the suitability of the proposed similarities and the clustering algorithm for clustering system resources. As we have discussed above, appropriate combination of similarity and clustering algorithm should correctly recover behavioral indicators for individual system resources. The results summarized in Tables A.4, A.5, A.6 and A.7 indicate that the proposed combinations of similarity and clustering algorithm provide best clustering results (in the case of network traffic, see Table A.5) or near best

Number of unique file paths	5 416 757
Number of unique registry keys	823 852
Number of unique mutexes	78 963
Number of unique domains/IPs	20 926

Table A.3: Numbers of unique system resources extracted from dataset used in Section 5.2 for evaluation of the classification performance of MIL-model.

	CLINK	SLINK	GDBSCAN	ApproxL
Cosine ($n = 1$)	0.267	0.171	0.178	0.452
Cosine ($n = 2$)	0.313	0.293	0.304	0.508
Cosine ($n = 3$)	0.377	0.297	0.309	0.542
Damerau-Levenshtein	0.340	0.444	0.444	0.537
Jaccard ($n = 1$)	0.207	0.099	0.099	0.343
Jaccard ($n = 2$)	0.336	0.441	0.420	0.508
Jaccard ($n = 3$)	**0.462**	0.483	0.453	0.534
Jaro-Winkler	0.247	0.469	0.416	0.491
LCS	0.255	0.486	**0.486**	**0.589**
Levenshtein	0.367	0.443	0443	0.524
n-gram ($n = 1$)	0.369	0.532	0.508	**0.571**
n-gram ($n = 2$)	0.398	**0.534**	0.509	0.550
n-gram ($n = 3$)	0.308	**0.544**	**0.518**	0.536
Sorensen-Dice ($n = 1$)	0.220	0.099	0.099	0.421
Sorensen-Dice ($n = 2$)	0.287	0.469	0.427	0.513
Sorensen-Dice ($n = 3$)	**0.458**	0.472	0.448	0.525
String kernel ($n = 1$)	0.256	0.171	0.178	0.459
String kernel ($n = 2$)	0.247	0.330	0.330	0.504
String kernel ($n = 3$)	0.210	0.387	0.387	0.505
Proposed similarity	**0.620**	**0.687**	**0.687**	**0.677**

Table A.4: File paths clustering: Values of ARI (higher is better) estimated using testing data for proposed path similarity metric and selected string metrics combined with selected clustering algorithms.

	CLINK	SLINK	GDBSCAN	ApproxL
Perdisci ($n = 1$)	0.350	**0.562**	**0.562**	0.599
Perdisci ($n = 2$)	0.524	0.497	0.497	0.603
Perdisci ($n = 5$)	0.462	0.424	0.472	0.610
Proposed similarity	**0.668**	0.495	0.417	**0.708**

Table A.5: Clustering of network traffic: Values of ARI (higher is better) estimated using testing data for similarity proposed by Perdisci et al. [184] and proposed similarity combined with selected clustering algorithms.

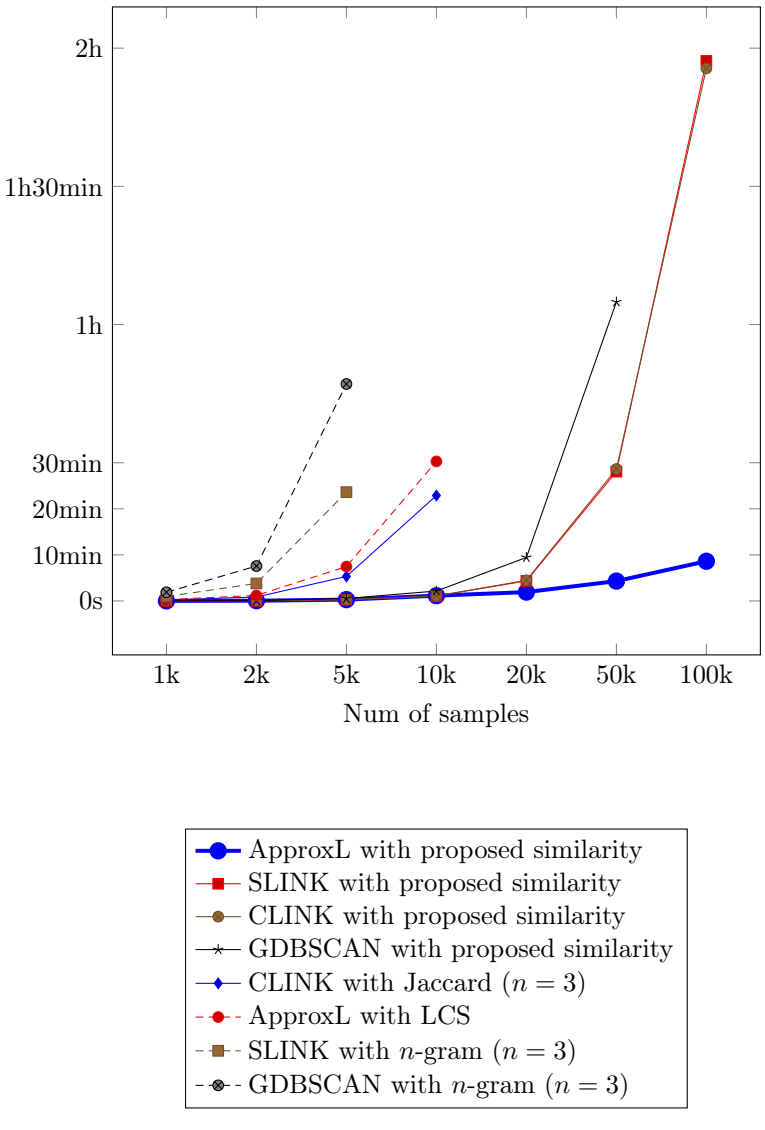


Figure A.3.1: Average processing time (lower is better) required to cluster file paths with various clustering algorithms coupled with two best performing path similarities.

	CLINK	SLINK	GDBSCAN	ApproxL
Cosine ($n = 1$)	0.391	0.532	0.532	0.624
Cosine ($n = 2$)	0.574	0.504	0.504	0.759
Cosine ($n = 3$)	**0.785**	0.660	0.660	0.794
Damerau-Levenshtein	0.385	0.343	0.343	**0.823**
Jaccard ($n = 1$)	0.234	0.530	0.493	0.687
Jaccard ($n = 2$)	0.426	**0.788**	**0.795**	0.762
Jaccard ($n = 3$)	**0.786**	0.579	0.539	0.755
Jaro-Winkler	0.328	0.715	0.715	0.688
LCS	0.453	**0.835**	**0.835**	**0.806**
Levenshtein	0.380	0.343	0.343	**0.812**
n-gram ($n = 1$)	0.445	0.754	0.754	0.793
n-gram ($n = 2$)	0.490	**0.822**	**0.822**	0.794
n-gram ($n = 3$)	0.424	0.774	0.774	0.790
Sorensen-Dice ($n = 1$)	0.227	0.536	0.536	0.662
Sorensen-Dice ($n = 2$)	0.426	**0.788**	0.795	0.799
Sorensen-Dice ($n = 3$)	**0.786**	0.579	0.579	0.800
String kernel ($n = 1$)	0.391	0.532	0.532	0.627
String kernel ($n = 2$)	0.322	0.590	0.590	0.656
String kernel ($n = 3$)	0.222	0.598	0.598	0.650

Table A.6: Mutex clustering: Values of ARI (higher is better) estimated using testing data for selected string metrics combined with selected clustering algorithms.

clustering results compared to all combinations of similarity metrics and clustering algorithms with very small difference between best possible combination and the proposed combination. This indicates that the proposed metrics correctly capture the similarity between corresponding system resources and is able to correctly recover the behavioral indicators captured in the testing data.

However, the optimal clustering results are useless if the proposed solution is not able to process the required amount of data. In the second part of evaluation we study the scalability of clustering algorithms in the scenario of clustering file paths as they constitute the largest amount of data we need to process (see Table A.3). We have selected for every clustering algorithm two similarity metrics that provided best results and compare the time required for clustering 1000, 2000, 5000,...,100 000 file paths. The growth of the processing time summarized in Figure A.3.1 proves that only the proposed similarity combined with the approximative Louvaine clustering algorithm provides the scalability necessary for clustering of required amount of system resources. The experiments therefore prove, that the proposed similarities combined with approximative Louvaine clustering algorithm are overall the best choices for clustering file paths, registry keys, mutexes and network traffic.

A.4 Scalability of the approximative Louvaine clustering algorithm

In the last set of experiments we discuss the influence of parameter k, the size of subset I', to the clustering performance and scalability of *ApproxL* algorithm defined in Section 5.1.6. As we have discussed above, results of the proposed *ApproxL* algorithm depend on the value

	CLINK	SLINK	GDBSCAN	ApproxL
Cosine ($n = 1$)	0.407	0.747	0.747	0.834
Cosine ($n = 2$)	0.461	0.822	0.822	0.916
Cosine ($n = 3$)	**0.695**	**0.860**	**0.860**	**0.928**
Damerau-Levenshtein	0.594	0.754	0.744	0.804
Jaccard ($n = 1$)	0.578	0.547	0.451	0.753
Jaccard ($n = 2$)	0.475	0.761	0.769	0.912
Jaccard ($n = 3$)	0.667	0.831	0.831	0.892
Jaro-Winkler	0.619	0.772	0.772	0.879
LCS	0.600	0.832	0.832	0.813
Levenshtein	0.654	0.754	0.743	**0.928**
n-gram ($n = 1$)	0.416	0.830	0.830	0.904
n-gram ($n = 2$)	**0.720**	0.825	0.825	0.890
n-gram ($n = 3$)	0.556	0.813	0.813	0.922
Sorensen-Dice ($n = 1$)	0.574	0.498	0.451	0.762
Sorensen-Dice ($n = 2$)	0.479	0.772	0.772	0.906
Sorensen-Dice ($n = 3$)	0.666	**0.834**	**0.834**	0.823
String kernel ($n = 1$)	0.407	0.747	0.747	0.840
String kernel ($n = 2$)	0.379	0.780	0.780	0.831
String kernel ($n = 3$)	0.348	0.826	0.826	0.821
Proposed similarity	**0.924**	**0.950**	**0.950**	**0.941**

Table A.7: Registry keys clustering: Values of ARI (higher is better) for proposed registry similarity metric and selected string metrics combined with selected clustering algorithms estimated using testing data.

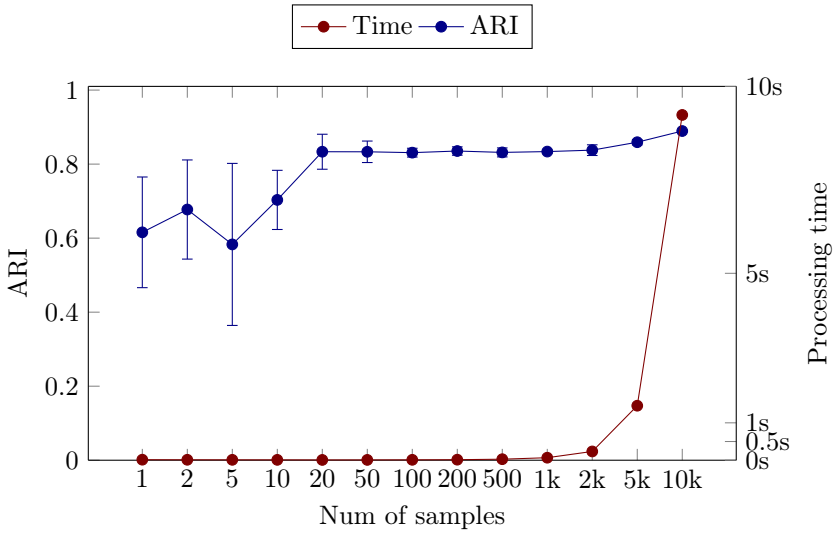


Figure A.4.1: Influence of the parameter k to the quality of the clustering of "mouse" dataset related to processing time.

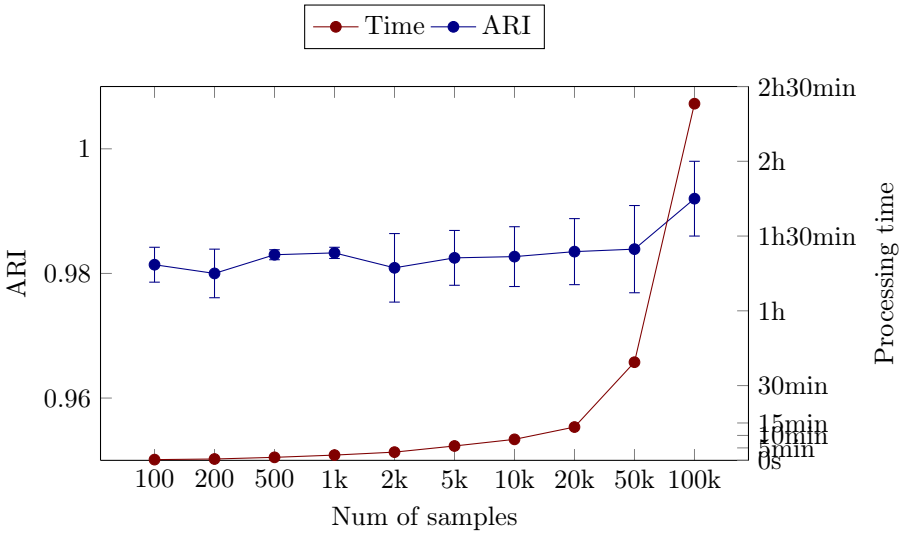


Figure A.4.2: Influence of the parameter k to the quality of the clustering of file paths compared to processing time. The approximative clustering with different values of $k = \{100, 200, 500, \ldots, 100\,000\}$ is compared to modularity clustering with complete data (100 000 samples).

of parameter k as if the value is too low, the clustering algorithm effectively degenerates to nearest neighbor search, and if it is too large, the initial clustering becomes too complex which prohibits the scalability to large datasets.

First we evaluated the influence of the parameter k on artificial dataset constructed from three gaussian distributions with parameters $\mu_1 = (2.5, 4)$, $\sigma_1 = 0.3$, $n_1 = 2500$, $\mu_2 = (-2.5, 4)$, $\sigma_2 = 0.3$, $n_2 = 2500$ and $\mu_3 = (0, -2)$, $\sigma_3 = 4$, $n_3 = 5000$, also known as "mouse dataset". For every value of parameter $k = \{1, 2, 5, 10, 20, 50, \ldots, 10\,000\}$ we have found the optimal value of parameter ϵ and estimated the ARI and processing time of the clustering algorithm. Values of ARI summarized in Figure A.4.1 show that lower values of parameter $k = \{1, 2, 5, 10\}$ degrades the clustering performance as the values of ARI are lower and with higher variance. It is caused by the fact that the *ApproxL* algorithm is not able to correctly estimate the initial clusters. When the initial subset I' is larger ($k = \{20, 50, 100, 200, 500, 1000\}$), the results stabilize (ARI = 0.83) and increasing size of the subset I' does not further improve the performance. With the last setting ($k = 10\,000$) the proposed algorithm becomes equivalent to original Louvain method since all samples are used in the initial step. The value of ARI for this setting proves that the approximation slightly affects the results (ARI = 0.89 for complete clustering). However, the processing time required for complete clustering of the artificial dataset becomes too large which advocates the approximation.

In the second experiment we evaluated the influence of the parameter $k = \{100, 200, 500, 1000, \ldots, 100\,000\}$ on real dataset composed of 100 000 file paths extracted from malware traits used in Chapter 5. Due to the fact that we do not have ground truth labels, we compared the clustering performance of the *ApproxL* algorithm with the results of Louvaine clustering method. The values of ARI and processing time summarized in Figure A.4.2 indicates that even though there is a drop in the performance between original Louvain method ($k = 100\,000$) and the *ApproxL* algorithm, the degradation of the clustering performance that can appeared on artificial dataset, does not appear on the real dataset. This is mainly caused by the fact that the space induced by the

file path similarity metric is very sparse and thus the data are very well separated. However, for the sake of generality of the solution we set the value of parameter $k = 10\,000$ as it provides good balance between the scalability and clustering performance. Note that for the dataset used in Chapter 5 (see Table A.3 for complete numbers) the nearest neighbor search dominates the processing time and therefore the performance gain achieved by setting lower values of parameter k is negligible and is overweighted by the possibility of unstable results.

Appendix B

Details about evaluation dataset

B.1 Number of samples per file type

File type	#samples	File type	#samples
Microsoft Office - DOC	898	Microsoft Office 2007+ - DOCM	546
Microsoft Office - HWP	1	Microsoft Office 2007+ - DOCX	18
Microsoft Office - MSI	17	Microsoft Office 2007+ - DOTM	49
Microsoft Office - PPS	1	Microsoft Office 2007+ - PPSX	2
Microsoft Office - XLS	21	Microsoft Office 2007+ - PPTX	10
HTML document	3464	Microsoft Office 2007+ - XLSM	138
Java archive data	1845	MS-DOS executable	1536
JavaScript	6370	PDF document	859
PE32 executable	97 428	Rich Text Format data	4
PE32+ executable	182	VisualBasic Script	4749
XML document text	5	Windows Script File	5295
Zip archive data	6760		
Total			130 198

Table B.1: Number of samples per file type extracted from evaluation datasets.

B.2 Number of samples of individual malware families

Table B.2: Number of samples of malware families in the data set. The malware families for individual samples were determined using AVClass tool [154]. The categories were determined using various reports published by AV companies and independent researchers.

Family	Category	#samples	Family	Category	#samples
nemucod	dropper	15 644	paneidix		278
cerber	ransomware	12 813	gamehack		277
bladabindi	inf. stealer	10 984	myxah		276
locky	ransomware	10 697	delbar		274
gamarue	banking trojan	7677	winner		264
darkkomet	RAT	4661	advml		261
hupigon	dropper	3568	confuser		260
upatre	dropper	3256	flystudio		252
tinba	banking trojan	3089	dynamer		249
scar	trojan	2958	redirector		242
swrort	dropper	2866	multiplug		241
zbot	banking trojan	2421	nitol		230
adwind	RAT	1875	mamianune		228
virlock	ransomware	1789	browsefox		217
fareit	inf. stealer	1784	fakejquery		211
farfli	trojan	1761	vopak		209
zegost	RAT	1745	cosmu		203
virut	trojan	1591	filefinder		194
zusy	trojan	1502	linkury		192
ircbot	trojan	1442	dlhelper		190
zerber	ransomware	1322	ardamax		188
palevo	worm	1269	msilperseus		184
vobfus	worm	1262	lotoor		183
delf	dropper	1255	pcclient		181
donoff	dropper	1211	vittalia		180
amonetize	adware	1198	lethic		170
loadmoney	PUA	1037	ngrbot		161
nanocore	RAT	1031	buzus		156
autoit		1006	skeeyah		151
yakes	trojan	901	koutodoor		150
poison		836	neshta		148
bifrose	inf. stealer	819	barys		148
kolabc	worm	705	parite		146
waldek		684	mikey		146
downloadassistant	PUA	678	redosdru		142
pdfka	exploitkit	644	hancitor		139
shipup	trojan	624	mintluks		134
rebhip	inf. stealer	612	cheatengine		133
razy		598	miancha		133
agentb		576	midie		133

onlinegames		529	fujacks	127
xtrat		510	pidief	126
ramnit	banking trojan	498	teslacrypt	126
atraps		478	kolovorot	124
magania		471	dupzom	123
softpulse	PUA	467	bublik	122
banload		435	opencandy	120
installmonster		411	socks	118
ruskill		373	icloader	114
convertad		363	fadok	113
installcore	PUA	361	garrun	113
sality	dropper	356	disfa	113
binder		355	chisburg	112
shiz		341	bedep	111
downloadguide	PUA	340	banbra	111
shyape		324	mydoom	108
llac		296	jaik	103
bayrob	inf. stealer	293	hlux	103
mywebsearch		283	refroso	101
winsecsrv		281		
Total malicious				130 198