Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Computer Science
Field of study: Computer Science

Dissertation Thesis

Offensive Applications of Machine Learning in Cybersecurity

Author:
Ing. Maria Rigaki

Supervisor:
Assist. Prof. Sebastian Garcia, PhD

May 2025

Acknowledgements

I would like to sincerely thank my advisor, Sebastian Garcia, for welcoming me into his research group and providing unwavering support and guidance throughout my PhD journey. His patience and mentorship have been invaluable to my growth as a researcher.

I am deeply grateful to Veronica, Ondra, Muris, and the entire Stratosphere family for creating such a collaborative and enjoyable work environment. Your friendship, technical insights, and daily encouragement made even the most challenging aspects of this research a pleasure to pursue. Special thanks to Harpo for our productive and enjoyable collaboration over the past couple of years.

The research presented in this thesis was supported by the Czech TACR project no. TH02010990, the OP RDE-funded project "Research Center for Informatics No.: CZ.02.1.01/0.0./0.0./16_019/0 0 0 0765", Avast Software, and the "Strategic Support for The Development of Security Research in the Czech Republic 2019–2025 (IMPAKT 1) Program, by the Ministry of the Interior of the Czech Republic under No. VJ02010020. Part of the presented research was performed on a Titan V GPU donated by NVIDIA Corporation.

"As you set out for Ithaka, hope your journey is a long one, full of adventure, full of discovery ..."

"Ithaca", by C.P. Cavafy

Abstract

As artificial intelligence and machine learning become deeply embedded in security-critical systems, their dual-use nature raises questions about how these technologies can be used by adversaries. This thesis explores the offensive applications of machine learning and artificial intelligence in cybersecurity, focusing on how adversaries can exploit such technologies to evade detection and execute complex attacks. It addresses two complementary challenges: the evasion of malware detection systems and the use of large language models (LLMs) as autonomous planning agents in network security environments.

The first part of the thesis investigates the evasion of static malware machine learning classifiers and antivirus engines through model extraction attacks. By leveraging active learning and transfer learning, surrogate models are constructed to approximate black-box target detectors under budget-constrained queries. These surrogates are then used to generate evasive malware samples that maintain functionality while bypassing detection. We introduce MEME, a novel algorithm that unifies model extraction and malware evasion into a single, efficient algorithm. We evaluate the ability of the surrogate models and MEME to evade several malware classifiers and AV products and demonstrate that attackers can produce evasive malware using query-efficient strategies.

The second part of the thesis examines the offensive use of LLMs as autonomous agents in simulated network environments. We present an extension of the ReAct [1] agentic architecture (ReAct+), which enables pre-trained LLMs to perform multi-step planning and execute realistic attack sequences in adversarial simulations. To address the limitations of cloud-based models, we develop Hackphyr, a fine-tuned open-weight model capable of running locally with competitive performance. We evaluate these LLM-based agents in two security environments, NetSecGame and Microsoft CyberBattleSim, and analyze their behavior against traditional RL agents and known attack patterns. Our findings demonstrate the viability of LLMs in decision-making tasks in the cybersecurity domain.

By combining practical offensive techniques with a focus on real-world applicability and limitations, this thesis aims to deepen the understanding of emerging threats in AI-driven cybersecurity.

Keywords: adversarial malware, model extraction, reinforcement learning, artificial intelligence agents, large language models

Abstrakt

S rostoucím vyu itím umelé intelligence (AI) a strojového ucení v kritických systémech kybernetické bezpecnosti roste i riziko jejich zneu ití útocníky. Tato disertacní práce se zabývá ofenzivními aplikacemi techto technologií, konkrétne zpusoby, jak mohou útocníci obcházet detekci malwaru a pou ívat velké jazykové modely (LLM) jako autonomní plánovací agenty v síťové bezpecnosti.

První cást zkoumá obcházení statických malware klasi kátoru pomocí technik extrakce modelu. Vyu itím aktivního ucení a transfer learningu jsou konstruovány náhradní (surrogátní) modely, které aproximují chování cílových detektoru fungujících jako cerné skrínky, a to pri omezeném poctu dotazu. Tyto modely pak slou í k tvorbe adversariálního malwaru, který zustává funkcní a zároven se vyhýbá detekci. Predstavujeme algoritmus MEME, který efektivne spojuje extrakci modelu a generování adversariálních vzorku. Evaluace ukazují, e MEME umo nuje útocníkum vytváret malware s minimem dotazu na cílový model.

Druhá cást disertace se zabývá ofenzivním vyu itím velkých jazykových modelu jako autonomních agentu v simulovaných síťových prostredích. Je navr ena architektura ReAct+, která umo nuje predtrénovaným jazykovým modelum provádet vícekrokové plánování a realizovat realistické útoky v adversariálních simulacích. Pro prekonání omezení cloudových rešení byl dále vyvinut model Hackphyr – model s volne prístupnými parametry, optimalizovaný pomocí metody ne-tuning pro lokální vyu ití pri zachování konkurenceschopného výkonu. Tato agentní rešení jsou testována ve dvou simulacních prostredích – NetSecGame a Microsoft CyberBattleSim – a jejich chování je analyzováno ve srovnání s tradicními agenty vyu ívající posilované ucení a známými útocnými strategiemi. Výsledky potvrzují, e velké jazykové modely jsou schopny efektivne plnit rozhodovací úlohy v kontextu kybernetické bezpecnosti.

Spojením praktických ofenzivních technik s durazem na reálnou aplikovatelnost a omezení prispívá tato disertacní práce k hlubšímu porozumení nove se objevujícím hrozbám v oblasti kybernetické bezpecnosti vyu ívající AI.

Klícová slova : adversariální malware, extrakce modelu, posilování ucení, agenti umelé inteligence, velké jazykové modely

Preklad názvu : Ofenzivní aplikace strojového ucení v kybernetické bezpecnosti

Contents

List of Figures

List of Tables

List of Abbreviations

AI	Arti cial Intelligence
API	Application Programming Interface
APT	Advanced Persistence Threat
AV	Anti Virus
AICA	Autonomous Intelligent Cyber Agents
C&C	Command and Control
CoALA	Cognitive Architectures for Language Agents
CoT	Chain of Thought
DLL	Dynamic Loading Library
DQN	Deep Q-Learning Network
ELU	Exponential Linear Unit
FFNN	Feed-Forward Neural Network
FPR	False Positive Rate
GAN	Generative Adeversarial Network
GBM	Gradient Boosting Machine
GPU	Graph Processing Unit
HTTP	Hyper-Text Transfer Protocol
JSON	JavaScript Object Notation
ICL	In-Context Learning
LGB	Light GBM
LLM	Large Language Models
LoRA	Low-Rank Adaptation
MCTS	Monte Carlo Tree Search
MEME	Model Extraction and Malware Evasion
ML	Machine Learning
PE	Portable Executable
PEFT	Parameter-Ef cient Fine-Tuning
PPO	Proximal Policy Optimization
QLoRA	Quantized LoRA
ROC	Receiver Operating Characteristic
ReLU	Recti ed Linear Unit
RL	Reinforcement Learning
RLHF	Reinforcement Learning from Human Feedback
RQ	Research Question
SFT	Supervised Fine-Tuning
SHA256	Secure Hash Algorithm 256-bit
SHAP	SHapley Additive exPlanations
SOTA	State Of The Art
TL	Transfer Learning
TW	Time Window

List of Symbols

D_L	Labeled dataset
D_U	Unlabeled dataset
D_S	Source domain dataset
D_T	Target domain dataset
D_{aux}	Auxiliary dataset used to augment the thief dataset
D_{thief}	Thief dataset used for querying the target
D_{val}	Validation dataset used for performance evaluation during training
D_{test}	Test dataset used for final performance evaluation
f_θ	Model parameterized by θ
$\hat{f}_\phi$	Surrogate model parameterized by ϕ
$\mathcal{L}$	Loss function
a_t	Action taken at time t
s_t	State at time t
o_t	Observation at time t
r_t	Reward at time t
$T(s_t, a_t)$	Transition function that produces s_{t+1} after taking action a_t in state s_t
π_θ	Policy parameterized by θ

Chapter 1

Introduction

The proliferation of machine learning (ML) and the rapid development of Artificial Intelligence (AI) tools are impacting many different domains today. Cybersecurity, in particular, exists as an inherently adversarial environment where attackers and defenders are in constant opposition. Attackers, driven by various motivations such as financial gain, control, or access to information, continually attempt to breach diverse targets ranging from home users and enterprises to governments and civil society organizations. These attackers possess widely differing capabilities, from script kiddies to sophisticated government-backed groups, in terms of their tools and knowledge. Artificial Intelligence may boost attackers' capabilities and increase automation in several tasks.

From the defenders' point of view, studying attackers' capabilities is crucial because awareness of these capabilities is essential for building better defenses. In addition, understanding potential weaknesses allows for better risk mitigation. Furthermore, possessing tools that can automate pentesting and other offensive techniques is essential for defenders if they want to keep up with the evolving threats that are accelerated by the use of AI.

This thesis tackles the problem of how adversaries can leverage machine learning, particularly through model extraction and large language models (LLMs), to bypass security systems in realistic, constrained settings. It examines how advancements in AI can be leveraged by adversaries to enhance their attack capabilities in two distinct sub-problems.

The first problem we examine is "Defense Evasion" (Figure 1.1) in the domain of Windows malware. The overarching question is how to modify Windows malware executables to evade the detection of malware classifiers. The standard approach in this area of work is to apply transformations to the binary file that modify some of its characteristics but do not alter its behavior. The sequence of modifications that need to be applied is usually selected by an optimization algorithm such as reinforcement learning (RL) [2]–[4] or genetic programming [5], [6].

Prior work is mainly focused on evasiveness, assuming that access to the target model or antivirus engine is uninterrupted or unlimited. However, we argue that this may be an unrealistic assumption that does not take into account the possibility of the attack being detected due to an unusually high number of queries of slightly

FIGURE 1.1: List of MITRE ATT&CK tactics for the enterprise with their respective descriptions. The last column of the table maps the parts of the thesis that correspond to each tactic.

modified files. A high number of queries and modifications can also be costly in terms of time and, therefore, not very interesting for attackers. Furthermore, several prior attacks [5]–[12] assume that the target model provides information such as a score and not just a binary label. Therefore, they are not strictly black-box attacks. We present an approach that is based on performing a model extraction attack on the target models within a limited query budget and then using the extracted model (surrogate) to produce evasive malware. We also propose a novel algorithm that combines the process of model extraction and evasion using a model-based reinforcement learning approach.

The second problem we explore is the development of autonomous planning agents capable of executing complex attack sequences in network security environments using Large Language Models (LLMs). Motivated by a data exfiltration scenario outlined in the MITRE ATT&CK framework (Figure 1.1), this work assumes an attacker has already established an initial foothold in a network. We propose an agentic design that extends the ReAct [1] framework called ReAct+, which can perform subsequent attack tactics, such as Discovery, Lateral Movement, Collection, Command and Control, and Exfiltration (Figure 1.1) within two distinct network security environments.

Prior work in network security environments is mostly based on reinforcement learning (RL) agents [13]–[16]. However, reinforcement learning requires thousands of episodes for an agent to be able to perform well. Moreover, RL-based agents are harder to adapt to new scenarios and may require retraining that is not necessary for the LLMs. Pre-trained foundational models show a remarkable ability to make planning decisions within the environments, even with the presence of a defender. However, foundational models are often costly and may come with privacy concerns. Therefore, we explore supervised fine-tuning of smaller, open-weight models that can be deployed in local networks.

1.1 Research Goals

Part I: Malware Evasion. This goal was addressed by first conducting an extensive study of model extraction attacks against black-box malware classifiers and AVs under realistic constraints, such as query limitations. Model extraction attacks produce surrogate models that mimic the target model's behavior. The study involved testing various active learning strategies and evaluating different surrogate model architectures. The use of transfer learning with an auxiliary dataset was proposed to enhance the stability of surrogate training, particularly when targeting complex systems like commercial antivirus products.

The surrogate models produced by the model extraction attacks were evaluated to determine whether they could be effectively used to generate adversarial malware binaries. This was achieved by employing existing state-of-the-art black-box attack frameworks, specifically MAB [17] and GAMMA [5], and using the extracted surrogate models to guide the malware modification process instead of querying the original target directly. This evaluation aimed to determine if malware generated to evade the surrogate model could also evade the original target models.

To overcome the limitations of a two-stage approach (extraction and then evasion), we also proposed a unified attack algorithm. The Malware Evasion and Model Extraction (MEME) algorithm is based on model-based RL and combines both evasion and extraction to improve efficiency and effectiveness. MEME integrates the process of training a surrogate model and learning an evasion policy into a single framework, using limited queries to the target to train the surrogate and then using the surrogate for more extensive policy training. The algorithm's performance was evaluated against three different malware classifiers and an Antivirus solution.

Part II: Attacking Agents. This research goal was addressed by exploring the potential of LLMs as autonomous planning agents. This included designing and evaluating an agentic architecture (ReAct+) that is our extension of the ReAct framework [1]. The agents were based on pre-trained foundational LLMs and were evaluated in simulated network security environments, speci cally NetSecGame [18] and Microsoft CyberBattleSim [19]. Performance metrics, including win rates, returns, and detection rates, were compared against baseline RL agents.

To overcome the problems of using cloud-based LLM models, such as a lack of control in terms of performance, as well as privacy concerns, we ne-tuned an open-weight model, Zephyr-7b- b [20] to create a new model called Hackphyr. A speci c dataset was generated for ne-tuning. The dataset addressed weaknesses of the base model, such as syntax errors and action repetition. Hackphyr's performance was compared to the foundational models in both the NetSecGame and CyberBattleSim environments. The model was designed to be runnable locally on minimal hardware.

To evaluate the decision-making process of the LLM-based agents, we also conducted a behavioral analysis, primarily through examining action transition graphs derived from their performance in the simulated environments. This analysis compared the sequences of actions taken by models like Hackphyr, the base Zephyr model, and GPT-4 to typical attack patterns as de ned by the MITRE ATT&CK framework and other similar approaches.

1.2 Ethical Considerations

Research into offensive cybersecurity techniques, particularly those leveraging advanced capabilities like machine learning and arti cial intelligence, carries inherent ethical responsibilities. The techniques explored in this thesis, including the extraction and evasion of malware classi ers and the development of adversarial agents using Large Language Models for network penetration tasks, describe methods that could potentially be misused by malicious actors. It is crucial to emphasize that the overarching motivation for this research is to understand attacker capabilities in order to improve defensive strategies. By investigating how adversaries might exploit advancements in AI, this work aims to provide defenders with critical insights necessary to build better defenses and mitigate potential risks. Understanding these offensive tactics, such as the feasibility of model stealing attacks or the potential for LLM-based agents to perform complex attack sequences, is essential for developing effective countermeasures.

The research presented in this thesis adheres to principles of responsible disclosure and conduct. All experiments involving offensive techniques against network environments were conducted exclusively within controlled simulation environments, such as NetSecGame and CyberBattleSim, designed speci cally for research purposes. No real-world systems or networks were targeted. With regards to the attacks against malware classi ers and AVs, no surrogate models or evasive malware were released to avoid misuse.

1.3 Reproducibility

Each of the papers that were part of this thesis was accompanied by the most relevant artifacts, such as code and datasets for reproducibility purposes.

1.4 Thesis Outline

Part I of the thesis is focused on the topic of malware evasion. Chapter 2 provides the necessary background knowledge and related work on the malware evasion literature. Chapter 3 focuses on using model extraction attacks to obtain surrogate models of malware classi ers that can be used subsequently in generating evasive malware. Chapter 4 describes a new algorithm that uses model-based reinforcement learning to generate adversarial malware. Chapter 5 closes Part I with the discussion about model extraction and evasion in the domain of malware classi cation, including the limitations of the current work.

Part II comprises Chapters 6 to 9 and describes attacking agents that use LLMs to plan attacks in a network security setting. Speci cally, Chapter 6 contains the technical background knowledge and the related work on network security environments and agents based on LLMs. Chapter 7 describes the design and experiments of a ReAct-type agent that is based on pre-trained frontier LLMs. Chapter 8 documents the design and experiments of performing supervised ne-tuning (SFT) on smaller open-source models that can compete with the larger frontier models, and Chapter 9 summarizes the ndings for Part II, including the discussion of the ndings and possible limitations.

The thesis concludes with Chapter 10, which provides a summary of the thesis contributions and potential future work. All publications related to the thesis are listed in the appendices.

Part I

Malware Evasion

Chapter 2

Background and Related Work

The increasing number of malware samples drives the need for increased automation and, subsequently, the use of machine learning and artificial intelligence as part of the detection pipeline. Part I of the thesis concerns attacks against machine learning classifiers trained to perform static malware detection. This chapter provides a high-level overview of the concepts and notations relevant to model extraction attacks and the evasion of malware classifiers. The chapter also presents the related prior work in these areas.

2.1 General Machine Learning Concepts

2.1.1 Supervised Learning

The machine learning models used for malware detection within the scope of this work are classifiers trained using supervised learning techniques. Supervised learning is a type of machine learning where the model learns a function that maps inputs to outputs based on labeled training data.

Given a training dataset $D = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$ where each x_i represents an input feature vector and y_i its corresponding target label, supervised learning aims to find a function $f : X \rightarrow Y$ that maps inputs to outputs, minimizing a loss function that measures the difference between predicted and actual outputs:

$$L(f) = \sum_{i=0}^{n} \ell(f(x_i), y_i) \tag{2.1}$$

where ℓ is a predefined loss function that measures the difference between the predicted output $f(x_i)$ and the true label y_i.

In the case of static malware detection, the input vectors are usually features extracted from benign and malicious files, such as the Ember v2 features [2]. Another approach is to use classifiers that can consume the binary file as-is, without feature extraction [21].

2.1.2 Reinforcement Learning

Reinforcement learning (RL) is a subfield of machine learning where an agent learns to make decisions by interacting with an environment and receiving rewards. An RL problem is typically modeled as a Markov Decision Process, defined by the tuple (S, A, P, R, γ), where:

- S is the set of states in the environment. The agent may have full or partial observability (O) of the state.
- A is the set of possible actions.
- $P(s'|s,a)$ is the state transition probability function of reaching state s' from state s by taking action a.
- $R(s,a,s')$ is the reward function, defining the reward received after taking action a in state s and reaching state s'.
- $\gamma \in [0,1]$ is the discount factor for future rewards.

The objective is to learn a policy $\pi : S \to A$ that maximizes the expected cumulative discounted reward:

$$E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1})\right] \tag{2.2}$$

Some of the most successful RL algorithms are *model-free* approaches, i.e., they do not try to build a model of the environment and instead interact directly with it. A downside of model-free algorithms is that they tend to require a lot of data and work under the assumption that the interactions with the environment are not costly. In contrast, *model-based* reinforcement learning algorithms learn a model of the environment and use it to improve policy learning. A fairly standard approach for model-based RL is to alternate between policy optimization and model learning. During the model learning phase, data is gathered from interacting with the environment, and a supervised learning technique is employed to learn the model of the environment. Subsequently, in the policy optimization phase, the trained model explores methods for enhancing the policy [22].

In terms of security applications, reinforcement learning has been proposed for use in several applications such as honeypots [23], [24], IoT and cyber-physical systems [25], [26], network security [27], malware detection [28], [29], malware evasion [4], [7], [9]–[11], [17], [30], [31], autonomous network defense [15], and offense [13], [14], [16].

2.1.3 Active Learning

Active learning is a machine learning paradigm in which the trained model or learner improves with experience acquired from external sources such as an oracle or a human labeler. In contrast, *passive learning* uses all the available training data to train the model. Part of the active learning approach involves the learner evaluating and selecting the data used to query the oracle. Active learning is especially useful when there is an abundance of unlabeled data, but the labeling process is expensive or time-consuming.

There are three main categories of active learning approaches ([32], [33]):

- Membership Query Synthesis : The model requests all unlabeled data to be labeled, including data that are synthesized [34].
- Stream-based selective sampling : Data points come in a stream, and the learner decides whether to obtain a label for each instance. This approach is particularly useful if the learner is deployed in devices with limited processing power or memory, such as mobile devices [35].
- Pool-based sampling : The learner selects the most informative sample from a pool of unlabeled samples [36].

Pool-based sampling is the most popular method. More formally:

Let X be the input space and Y be the output space. Given an initial labeled dataset $D_L = \{(x_i, y_i)\}_{i=1}^{n}$, where $x_i \in X$ and $y_i \in Y$, and a large pool of unlabeled instances $D_U = \{x_j\}_{j=n+1}^{m}$, a pool-based sampling algorithm iteratively performs the following steps:

1. Query Selection: Selects an instance $x \in D_U$ that maximizes a given informativeness criterion.
2. Oracle Query: Requests the oracle to provide the true label y for x.
3. Dataset Update: Updates the labeled dataset: $D_L \leftarrow D_L \cup \{(x, y)\}$ and removes x from D_U.
4. Model Training: Retrains the learning model using the updated D_L.
5. Iteration: Repeats the process until a stopping criterion is met (e.g., budget exhaustion or desired accuracy level reached).

The selection of the most informative samples from the unlabeled pool, or alternatively, the sampling strategy is one of the key research questions in any active learning approach. Different strategies involve uncertainty sampling (selecting samples for which the learner is least certain), diversity-based (selecting samples that are different from each other), error or variance reduction methods (selecting samples that reduce the model errors), and so on [33].

2.1.4 Model Extraction Attacks

Model extraction or model stealing is a class of attacks where the adversary tries to extract information and potentially fully reconstruct a target model f by creating a surrogate model $\hat{f}$ that behaves very similarly to the model under attack [37]. Most often, model extraction attacks involve querying the target model with speci cally selected or synthesized data to receive the target model's responses (labels, probabilities, logits, etc). Using this speci cally crafted dataset D_{thief}, the attacker can then proceed to train the surrogate model.

There are two main practical goals for the surrogate models. The rst goal is to create models that match the target model f's accuracy in a test set drawn from the input data distribution and related to the learning task. This is called task accuracy extraction [38]. This type of attack aims to create a substitute model that learns the same task as the target model equally well or better. The attacker's motivation is to create a substitute that performs reasonably well with lower training costs. Models deployed in the cloud and accessed through APIs (Machine Learning as a Service - MLaaS) can be typical targets in this category.

The second goal is to create a substitute model $\hat{f}$ that matches f at a set of input points that are not necessarily related to the learning task. Jagielski et al. [38] referred to this goal as delity extraction . In delity extraction, the adversary aims to create a substitute that replicates the decision boundary of f as faithfully as possible for some data distribution of interest. This type of attack can later be used as a stepping stone for launching other attacks, such as adversarial or membership inference attacks. In the case of adversarial attacks the thief dataset D_{thief} used to query the target may contain synthetic adversarial examples.

A third, more general but harder adversarial goal is functional equivalent extraction . This can be considered a generalization of delity extraction over all possible data distributions [38]. Functionally equivalent extraction attacks, usually do not employ learning-based algorithms, but use other techniques that allow the direct recovery of neural network weights and model parameters [38]–[43]. While fully recovering a model is desirable, these attacks are usually constrained to speci c models or architectures.

Formally, learning-based model extraction can be de ned as follows:

Let $f : X \rightarrow Y$ be the target model, where X is the input space and X is the output space, and $D_{thief} = \{x_i\}_{i=1}^{n}$ the dataset used by the adversary to query the target model. When queried, the target model provides either a hard-label output $y = f(x)$ or a soft-label output, i.e., a probability distribution over the output classes $p(y|x)$. The thief dataset and the outputs gathered by the queries to the target model are used to train a surrogate model $\hat{f}$.

In both task accuracy and delity extraction attacks, it is assumed that the adversary wants to be as ef cient as possible, i.e., to use as few queries as possible. There are multiple reasons for minimizing the number of queries such as lowering the cost of attack (API costs), lowering the probability of detection, and minimizing the time it takes to perform the attack.

One of the rst model extraction attacks was proposed by Tramer et al. [44], where the authors showed that it is possible to fully reconstruct a linear binary classi er by using enough queries to allow the model parameters to be computed by solving a system of equations. Using randomly generated thief datasets, they also proposed approximate attacks against decision trees, shallow neural networks, and SVMs.

Other learning attacks test the use of thief datasets that may be related to the original domain or not. The CopyCat attack [45] showed that it is possible to steal a convolutional network that performs image classi cation using both types of thief datasets. The Knock-off attack [46] proposed the use of reinforcement learning to select samples from a thief dataset that would make the attack as query ef cient as possible. The attack was tested against image classi ers and used both types of thief datasets to construct the thief dataset.

A model extraction attack shares many similarities with active learning (Section 2.1.3), where there is an oracle function that provides labels for each sample when queried. Similarly, in model extraction attacks, the target model plays the part of the oracle, and the attacker aims to learn a good approximation of the oracle function. This connection of model extraction to active learning was explored in [47], where the authors proposed using query synthesis techniques to extract different types of ML models such as decision trees, random forests, SVMs, and linear models. The use of synthetic samples had been previously proposed in [48], where a small initial seed of real images was used to create new images using Jacobian-based dataset augmentation, and these synthetic data were used to query the target model.

The ActiveThief attack was proposed in [49], where the authors tested different active learning sampling strategies to create query-ef cient attacks against image and text classi ers. They also proposed to generate adversarial samples for the data in the thief dataset and then use the samples with the smallest distance to their respective adversarial samples as a sampling strategy. The use of adversarial attacks was also proposed in "CloudLeak" [50] with the difference that the synthetic adversarial samples were used to train the surrogate model. The attack targets image-based classi ers that are deployed in the cloud. One of the problems in using synthetic data in the malware domain is that binary les are much harder to construct than images [51].

In the security domain, model extraction has been proposed as the rst step of an attack that aims to generate evasive malware [52], [53]. Rosenberg et al. [52] generated evasive malware by creating a surrogate and then evading the target. Hu et al. [53] use a Generative Adversarial Network (GAN) and a surrogate detector to bypass a malware detector that works with API calls. Neither [53] nor [52] considered the number of queries they are making to the target as an attacck constraint.

2.2 Static Malware Detection and Evasion

Static malware analysis is the process of analyzing a malware le and extracting different pieces of identifying information and statistical features without observing its behavior while it is being executed. This entails inspecting the le's structure and contents using various tools such as disassemblers, decompilers, resource and string extractors, and more. When designing machine learning models for discriminating between malware and benign les, the process entails extracting numerous features related to the le structure and contents [2].

In contrast, dynamic analysis requires malware execution, usually in a controlled environment such as a sandbox or a Virtual Machine (VM). Running the malware allows the analyst to observe behavioral characteristics, such as le system modi - cations, processes, and network traf c. Other dynamic analysis types may involve using an emulator that collects API calls and le paths [54].

Static malware detection, traditionally, uses different techniques: signatures, heuristics, and even machine learning. Machine learning often requires feature extraction, although there is ongoing research in feeding whole executables into deep learning models.

The most popular feature set used in the literature is the Ember v2 [2]. The EMBER v2 dataset offers a rich and structured set of static features extracted from Windows Portable Executable (PE) les, aimed at facilitating machine learning research in malware classi cation. The feature set contains general le attributes, including le size, virtual size, and timestamps. These are accompanied by metadata extracted from the PE headers, such as the number of sections, characteristics of the le, and details from the optional headers like image base and DLL characteristics.

Another component of the feature set focuses on section information, which details the properties of individual sections within a PE le, such as names, sizes, entropy values, and access ags (e.g., executable or writable).

In addition, EMBER v2 includes byte-level statistical features like a 256-bin histogram of byte values and a byte entropy histogram. These histograms capture the distribution of byte values and their local entropy, providing insight into the randomness or structure of different parts of the le. This can be useful in detecting packed or obfuscated binaries. The dataset also extracts and analyzes printable strings, including counts, average lengths, and the presence of features like URLs, le paths, or registry keys—elements often found in malicious payloads.

2.2.1 Windows Portable Executable File Format

The Windows Portable Executable (PE) le format is used by binary les and dynamically loaded libraries (DLLs) in the Windows operating system. It consists of code, data, and metadata necessary for a program to be loaded and executed by the Windows loader.

FIGURE 2.1: Windows Portable Executable File Format

Figure 2.1 shows the different components of the PE format for 32 and 64-bit Windows. There are four main components: the optional MS-DOS headers, the PE/COFF headers, the section headers, and the sections themselves. The MS-DOS headers exist for backward compatibility reasons with the old MS-DOS operating system. They usually start with the string "MZ" and contain the message "This program cannot be run in DOS mode," which is printed if the binary is executed in MS-DOS. The PE header follows the MS-DOS header and contains a 4-byte signature (PE\0\0). After that, the PE le header begins and it contains important information about the le, such as the machine type and the size of the optional header. The Common Object File Format (COFF) le header provides metadata about the PE le, including the number and size of the sections, the time the le was created, and symbols. Finally, the optional header contains information that's necessary for the loader, such as the entry point of the code, the image base, and the sizes of various sections of the PE le.

After the general le headers, come the section headers that provide information about the sections of the PE le. Each section header describes a section of the le, with information like its size, location, and characteristics (ags). Then, the actual sections follow. Common sections include:

1. .text: Contains the executable code.

2. .data: Holds initialized global and static variables.
3. .rdata: Contains read-only data, like constants and strings.
4. .bss: Holds uninitialized global and static data.
5. .idata: Contains import tables, which are used for importing functions and resources from DLLs.
6. .edata: Contains export tables used for exporting functions and resources.
7. .rsrc: Stores resources like icons, bitmaps, and menus.
8. .reloc: Contains relocation information.
9. .pdata (only for 64-bit): Contains information about exception handling.
10. .debug: Contains compiler-generated debug information.

2.2.2 Functionality-Preserving Modifications

To generate adversarial malware, an attacker needs to modify the original binary file in such a way that it evades detection by machine learning classifiers while preserving its original functionality. A key challenge in this process arises due to the distinction between feature space and problem space modifications in binary executable files. Adversarial examples in objects such as images are often generated by gradient-based attacks that leverage the gradient of the loss function with respect to the input image to calculate perturbations. However, in the case of binary executable files, the feature space is neither invertible nor differentiable [51]. A feature-problem space dilemma arises: A change in the problem space (modification of the binary file) may affect multiple features, and vice versa, a change in the feature space must have a feasible equivalent in the problem space [55].

In addition, all proposed modifications to a binary file must preserve its semantics; for example, the program execution of the file must be identical to the original file. Several different functionality-preserving modifications have been observed or proposed in real-world attacks and academic research:

- modify_machine_type : Change the machine type in the file header to mislead static analysis tools. By altering this field, malware can disguise itself as a binary intended for a different architecture.
- pad_overlay: Append random bytes at the end of the file to modify the file hash, the histograms of bytes, and the histograms of byte entropy without affecting execution.
- append_benign_data_overlay : Append benign section data to the end of the file, which alters multiple features such as the number of sections, byte and entropy histograms, strings, and so on.

- append_benign_binary_overlay : Append an entire benign binary to the end of the malware to confuse classi ers.
- add_bytes_to_section_cave: Inject random bytes into underutilized sections of the binary without altering execution.
- add_section_strings : Insert a new section containing benign strings to mimic legitimate software behavior. Adding strings also affects the le entropy and the byte distributions since strings contain readable characters.
- add_section_benign_data : Create a new benign section in the binary to alter feature representations.
- add_strings_to_overlay : Append benign strings at the end of the le.
- add_imports : Introduce new library imports to import address table to mimic benign les.
- rename_section: Rename an existing section to mimic legitimate executables and evade signature-based detection. Changing section names can confuse heuristics that rely on prede ned section naming conventions.
- remove_debug : Strip debugging symbols from the binary.
- modify_optional_header : Modify elds in the PE optional header such as OS version to alter classi er features.
- modify_timestamp : Change the timestamp to make the le appear to be compiled at a different date.
- break_optional_header_checksum : Corrupt the checksum eld in the PE header, which some security tools rely on. Certain security solutions validate checksums for integrity veri cation, and altering this value can interfere with such checks.
- upx_unpack : Unpack the binary using the Ultimate Packer for eXecutables (UPX) to reveal its original structure. Many malware samples use packing to obfuscate their code, and unpacking may allow further modi cations before repacking.
- upx_pack : Repack the binary using UPX to alter its appearance while preserving execution. Packing can obscure static features.

2.2.3 Generating Evasive Malware

The main goal of malware evasion is to make the malware undetectable by security systems so that it can perform its malicious activities unnoticed. Malware creators continuously develop new techniques to evade detection, which makes it dif cult

Attack	Target Output	Technique
Malware-Gym [7]	prediction score	RL (ACER)
AIMED [6]	prediction score	Genetic
Ceschin et al. [8]	prediction score	Dropper
Fang et al. [9]	prediction score	RL (DDQN)
GAMMA [5]	prediction score	Genetic
AIMED-RL [4]	hard label	RL (Dist. DDQN)
Li et al. [10]	prediction score	Inverse RL
MERLIN [31]	hard label	RL (REINFORCE)
Phan et al. [11]	prediction score	RL + GAN
Rosenberg et al. [12]	prediction score	Explainability
MAB [3]	hard label	RL (Bandits)

TABLE 2.1: Black-box attacks against static malware classi ers of Windows executables

for security experts to keep up with the evolving threat landscape. The concept has evolved to include the evasion of machine learning classi ers and the use of machine learning techniques to evade antivirus (AV) systems. For this work, we are mainly interested in machine learning approaches that attackers may use to generate evasive fully functional Windows PE malware so that static malware classi ers and AVs can not detect them.

One of the main assumptions behind the attacks presented in this work is that the target models are black-boxes and that they provide only a hard label (malicious or benign) as an output of the static analysis. Table 2.1 provides a list of the prior work on attacking static malware Windows classi ers and AVs using black-box attacks. All the attacks use one or more functionality-preserving modi cations (Section 2.2.2) to modify the malware until they become evasive. The selection of modi cations is performed using several different approaches such as reinforcement learning [3], [4], [7], [9], genetic algorithms [5], [6], explainability [12], or other empirical approaches such as droppers [8].

The only attacks that assume that the output of the target is a hard-label are MAB [3], AIMED-RL [4], and MERLIN [31]. The rest of the attacks assume that the target output is a probability vector or a score (soft label), which provides a strong signal to the optimization algorithms that are used, and is not a realistic assumption when it comes to AVs.

A subset of the attacks tests the validity of their generated binaries ([3], [6]) or checks the malware against a real antivirus or VirusTotal ([3], [5], [7], [8], [56]).

Finally, different attacks allow for different maximum values for the number of permitted modi cations, varying from as low as 5 to as high as 80. However, none of the prior works constraints the number of queries to the target or discuss whether unlimited query access or unlimited time to generate evasive samples is an important aspect of realistic attacks.

Chapter 3

Stealing and Evading Malware Classi ers

3.1 Introduction

Modern malware classi ers and antivirus systems increasingly rely on machine learning to identify malicious binaries. However, this reliance introduces a security concern: can an adversary replicate the behavior of these classi ers through model extraction and then use the surrogate models to generate evasive malware? This chapter addresses the problem of adversaries exploiting limited black-box access to such systems to both reverse-engineer their decision boundaries and craft undetectable malware variants.

While model extraction has been widely explored in domains like image and text classi cation, its application in cybersecurity is less explored [37]. The security domain also imposes harsher adversarial constraints. In contrast to many academic setups where attackers are granted access to probability vectors ([45], [46], [49]), realistic threat models in cybersecurity must assume black-box access, where the attacker only receives a binary classi cation (malicious or benign). This signi cantly limits the feedback available to guide learning. Moreover, malware classi ers and AV systems are often part of large, complex pipelines, potentially combining static and dynamic analysis, heuristics, and signature-based detection. Attackers typically lack any detailed knowledge of the model's feature space, architecture, or training data.

Compounding these challenges is the cost and risk associated with querying such systems. Prior work on malware evasion assumes that the attackers operate with unlimited query budgets ([2], [5], [6], [9], [12]). However, querying commercial

---

This chapter is based on the following publications:

- M. Rigaki and S. Garcia, “Stealing and evading malware classi ers and antivirus at low false positive conditions,” Computers & Security, vol. 129, Jun. 2023, ISSN: 0167-4048. DOI: 10.1016/j.cose.2023.103192.
- M. Rigaki and S. Garcia, “A survey of privacy attacks in machine learning,” ACM Computing Surveys, vol. 56, no. 4, pp 1-34, Nov. 2023, ISSN: 0360-0300. DOI: 10.1145/3624010.

AV engines or malware classifiers too frequently, especially with only slightly modified samples, risks detection or rate-limiting. From a practical standpoint, attackers are not only constrained by cost and stealth but also by time. In real-world scenarios, adversaries are unlikely to spend hours or days repeatedly modifying and submitting samples. Excessive time spent in the attack loop increases operational exposure and reduces the practicality of evasion at scale. Therefore, achieving query-efficient and time-efficient model extraction is a key goal for realistic offensive strategies.

Furthermore, constructing a realistic attacker dataset is nontrivial. Unlike other domains where synthetic or augmented data can be easily generated (e.g., rotated images), generating valid synthetic malware samples that preserve malicious functionality is significantly more complex [51].

This chapter presents a systematic methodology to address this adversarial problem. It investigates whether surrogate models—trained via active learning and transfer learning under constrained conditions—can both approximate malware classifiers and be leveraged to generate real, functionality-preserving, adversarial malware. Our central aim is to explore the following research questions:

- RQ1: How successful are model extraction attacks against black-box malware classifiers and antivirus systems?
- RQ2: How effective are surrogate models at generating adversarial malware?

3.2 High-Level Methodology

A high-level diagram of the attack methodology is shown in Figure 3.1. In the first part of the methodology, an adversary creates surrogate models that extract, or steal, the functionality of the target models. This part of the attack is detailed in Section 3.3. The second part of this paper uses those surrogate models to create adversarial malware binaries that test their evasion ability against target models, including real antivirus systems (both offline and online). Section 3.6.1 details this methodology's second part.

3.2.1 Threat Model and Assumptions

The threat model for this work is depicted in Figure 3.2 and assumes an attacker, such as a malware author, with only black-box access to a target (classifier or AV). The attacker aims to generate valid, malicious Windows executable files that bypass the malware classifier or the AV. To achieve this goal, the attacker queries the target and uses the responses to generate a dataset that will aid them in creating a surrogate model. The surrogate model is an intermediate step that replicates the functionality of the original target and allows the attacker to generate the adversarial malware without further engaging with the target. The attacker queries the target during inference and can submit binary files for static scanning. The target provides binary labels ('0' if benign, '1' if malicious).