A Game-Theoretic Framework for Detecting Evasive Malware

Taha Zouggari
Nara Institute of Science and Technology (NAIST)

Graduate School of Science and Technology
Nara, Japan

Youki Kadobayashi, Ph.D.
Yuzo Taenaka, Ph.D

***Abstract*—The growing sophistication of malware poses significant challenges to traditional detection techniques. Modern threats often employ evasive behaviors, including strategic delays and environment-aware execution, making them increasingly resistant to static and dynamic analysis. In response to this evolution, we propose a strategic defense framework grounded in game-theoretic principles. Specifically, we model the interaction between a defender and an intelligent malware adversary as a hierarchical decision-making process, where the defender anticipates possible evasive maneuvers and adapts its monitoring strategy accordingly. Our approach emphasizes the importance of anticipating attacker intentions rather than reacting post-compromise. By integrating this strategic perspective with enhanced behavioral monitoring capabilities, we demonstrate how defenders can strengthen their ability to disrupt malicious operations before critical damage occurs. This work lays the foundation for more adaptive, resilient, and intelligence-driven malware defense mechanisms.**

***Keywords*—Malware Detection, Stackelberg Game, Markov Decision Process, Ransomware**

I. Introduction

The rapid evolution of cyber threats, particularly evasive malware, such as ransomware, has significantly outpaced traditional detection mechanisms. Attackers increasingly rely on sophisticated techniques ranging from time-based execution delays to anti-analysis checks to avoid detection and maximize impact. As a result, conventional static and dynamic analysis methods often fall short, either missing critical behaviors or being actively evaded.

To counteract this growing challenge, there is a pressing need for defense strategies that not only react to known threats, but also anticipate and adapt to adversarial intent. Game theory offers a powerful framework for this objective. In particular, Stackelberg games where one player (the leader) commits to a strategy first while the other (the follower) responds optimally provide a natural model for the interaction between defenders and intelligent, adaptive malware. By committing to a monitoring strategy and anticipating the attacker's best response, the defender can optimize its resource allocation to detect and disrupt malicious activities more effectively.

In this paper, we will go through:

- the formal modeling of evasive malware behavior using Markov Decision Processes.
- the attacker's optimization strategy under defender surveillance using value iteration,
- the design and comparison of various defender monitoring strategies, including focused and full monitoring,
- the evaluation of attacker reward under each strategy to assess effectiveness,
- and the introduction of a defender utility function to balance detection gain against monitoring cost.

A. *Camouflage Evolution in Malware*

Modern malware increasingly employs a range of sophisticated obfuscation techniques—such as encryption, oligomorphism, polymorphism, metamorphism, stealth, and packing to evade [2]. These techniques are designed to complicate the analysis and recognition process, allowing malware to bypass even kernel-level security tools like firewalls and antivirus software. In many cases, a single malware instance may exhibit traits from multiple categories simultaneously, further complicating detection efforts. As a result, relying on a single detection method is often insufficient for identifying all threats [4]. The most commonly used obfuscation techniques are described below

- Encryption: Malware authors aim to ensure that their malicious programs remain undetected by antivirus and malware detection systems [2]. One of the earliest and simplest techniques used to achieve this concealment is encryption. This method involves wrapping the malware in an encryption layer, coupled with a decryption routine that restores the original code during execution. Typically, a new encryption key is used each time the malware spreads, while the decryption logic remains unchanged. This reuse of the decryption module introduces a potential point of detection. The first known example of such an encrypted malware was Cascade [3], which emerged in 1987. The structural layout of an encrypted virus is illustrated in

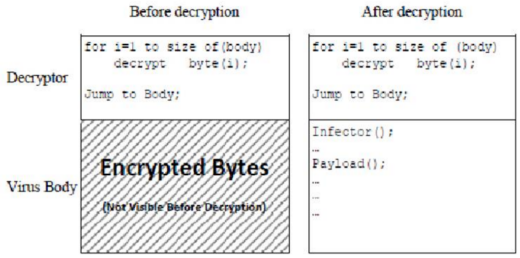

Figure I: Structure of Encrypted Virus

- Oligomorphi: In 1990, First Polymorphic virus 1260 was developed by Mark Washburn [2]. In oligomorphic method, a different key is used when encrypting and decrypting malware payload [3]. Thus, it is more difficult to detect malware which uses oligomorphic method than encryption.

- Polymorphic: In polymorphic method, malware uses a different key to encrypt and decrypt [3] likewise the key used in oligomorphic method. However, the encrypted payload portion contains several copies of the decoder and can be encrypted in layered. Thus, it is more difficult to detect polymorphic malware when compared to oligomorphic malware.t:

- metamorphic: The metamorphic technique does not rely on encryption; instead, it employs dynamic code transformation, where the opcodes are altered with each execution of the malicious process. This results in every new instance of the malware having a completely different binary signature, making it extremely challenging to detect using traditional signature-based methods [3].

- Stealth: Also known as code protection, the stealth technique incorporates various countermeasures to hinder effective analysis. It can modify system components while actively concealing these changes from detection mechanisms [3], making it difficult for security tools to identify its presence.

- Packaging: is an obfuscation technique that compresses or encrypts malware to conceal its true code and evade detection by security systems such as firewalls and antivirus software. This method allows malicious software to bypass defenses more effectively. Before analysis, packed malware must first be unpacked to reveal its actual behavior [5] [6]. Packers are generally classified into four categories: compressors, crypters, protectors, and bundlers, each serving a specific role in disguising the malware's contents.

These techniques are among the most commonly used methods for bypassing malware detection. They significantly hinder traditional static and signature-based detection strategies, necessitating more advanced behavioral and heuristic approaches in modern security solutions.

In this study, we will be focusing on ransomware, one of the most dangerous types of malware, due to its harmful impact on both individuals and organizations. We aim to analyze its behavior and develop effective detection strategies.

II. Game theory

In this section, we introduce the fundamental concepts of game theory, with a particular focus on the Stackelberg game model. We then explore how these theoretical frameworks can be applied to the domain of malware detection.

Game theory is a mathematical framework that analyzes situations of strategic interaction between rational agents, referred to as players. Each player in a game has a set of strategies and makes decisions with the goal of maximizing their own payoff. The outcome of the game depends not only on a player's own actions but also on the actions chosen by others [7]. This interdependence makes game theory a powerful tool for studying competition, cooperation, negotiation, and conflict in various domains, including economics, politics, and cybersecurity.

In this study, we focus on the Stackelberg game, which is a specific type of sequential and asymmetric game.

A. *Stackelberg game*

Stackelberg games are a class of non-cooperative games involving two players: a leader L and a follower F. In this framework, the leader first selects a distribution over possible actions, and the follower, after observing this distribution, responds by choosing their own action distribution. Then, actions for both players are sampled from their respective distributions, and each player receives a utility or payoff based on the resulting action pair [8].

Unlike normal-form games such as Rock–Paper–Scissors or the Prisoner's Dilemma [8], where players choose their actions simultaneously, Stackelberg games are inherently sequential. In these games, a **leader** commits to a strategy first, and the **follower** observes this strategy and responds accordingly.

Stackelberg games also differ from extensive-form games like chess or poker in terms of how actions are sampled. In extensive-form games, each player's move is sampled and observed before the next player makes a decision, meaning the second player conditions their strategy on the first player's realized action. In Stackelberg games, by contrast, the follower can condition their strategy on the *distribution* over the leader's actions, rather than on a single action outcome.

Let each player $i \in \{L, F\}$ have an action set $\mathcal{A}_i$. A joint action profile is denoted by $(a_L, a_F) \in \mathcal{A}_L \times \mathcal{A}_F$. A strategy $s_i \in \mathcal{S}_i = \Delta(\mathcal{A}_i)$ is a probability distribution over player i's action set. That is, for any action $a_i \in \mathcal{A}_i$, $s_i(a_i)$ denotes the probability that action a_i is chosen.

A strategy is called a *pure strategy* if it assigns probability 1 to a single action. If the strategy assigns non-zero probabilities to multiple actions, it is a *mixed strategy*. A strategy profile $(s_L, s_F) \in \mathcal{S}_L \times \mathcal{S}_F$ is a pair of mixed strategies, one for each player.

Each player i is associated with a utility function $u_i : \mathcal{A}_L \times \mathcal{A}_F \to \mathbb{R}$, which assigns a real-valued utility to every joint action profile. The expected utility for player i under strategy profile (s_L, s_F) is given by:

$$u_i(s_L, s_F) = \sum_{(a_L, a_F) \in \mathcal{A}_L \times \mathcal{A}_F} s_L(a_L) \cdot s_F(a_F) \cdot u_i(a_L, a_F) \quad (1)$$

Players act to maximize their own expected utility, without regard for the utility of others, except to the extent that it affects their ability to predict opponents' actions.

The Stackelberg game unfolds in the following steps:

1. The leader selects a strategy $s_L \in \mathcal{S}_L$.
2. The follower observes the leader's strategy s_L.
3. The follower selects a best response strategy $s_F \in \mathcal{S}_F$.
4. An action profile (a_L, a_F) is sampled with $a_L \sim s_L$ and $a_F \sim s_F$.
5. Each player i receives a utility $u_i(a_L, a_F)$.

B. *MDP Modeling of Ransomware Behavior*

The verification of MDPs is crucial for the design and evaluation of cyber-physical systems with sensor noise, biological and chemical processes, network protocols, and many other complex systems. MDPs are the standard model for sequential decision making under uncertainty and thus at the heart of reinforcement learning. Many dependability evaluation and safety assurance approaches rely in some form on the verification of MDPs with respect to temporal logic properties. Probabilistic model checking [4,5] provides powerful tools to support this task.

Markov Decision Processes [9]: Let $D_X := \{d : X \to [0,1] \mid \sum_{x \in X} d(x) = 1\}$ be the set of distributions over X. A Markov decision process (MDP) is a tuple $M = (S, A, \delta)$ with finite sets of states S and actions A, and a partially defined transition function $\delta : S \times A \to D_S$ such that $A(s) := \{a \mid (s,a) \in \text{domain}(\delta)\} \neq \emptyset$ for all $s \in S$. $A(s)$ is the set of enabled actions at state s. δ maps enabled state-action pairs to distributions over successor states. A Markov chain (MC) is an MDP with $|A(s)| = 1$ for all s. The semantics of an MDP are defined in the usual way. A (memoryless deterministic) policy—a.k.a. strategy or scheduler—is a function $\pi : S \to A$ that, intuitively, given the current state s prescribes what action $a \in A(s)$ to play. Applying a policy π to an MDP induces an MC M^π. A path in this MC is an infinite sequence $\rho = s_1 s_2 \ldots$ with $\delta(s_i, \pi(s_i))(s_{i+1}) > 0$. Paths denotes the set of all paths and $\mathbb{P}_s^\pi$ denotes the unique probability measure of M^π over infinite paths starting in the state s.

A reachability objective $P^{\text{opt}}(T)$ with set of target states $T \subseteq S$ and $\text{opt} \in \{\max, \min\}$ induces a random variable $X : \text{Paths} \to [0,1]$ over paths by assigning 1 to all paths that eventually reach the target and 0 to all others. $\mathbb{E}^{\text{opt}}(\text{rew})$ denotes an expected reward objective, where $\text{rew} : S \to \mathbb{Q}_{\geq 0}$ assigns a reward to each state. $\text{rew}(\rho) := \sum_{i=1}^{\infty} \text{rew}(s_i)$ is the accumulated reward of a path $\rho = s_1 s_2 \ldots$. This yields a random variable $X : \text{Paths} \to \mathbb{Q} \cup \{\infty\}$ that maps paths to their reward. For a given objective and its random variable X, the value of a state $s \in S$ is the expectation of X under the probability measure $\mathbb{P}_s^\pi$ of the MC induced by an optimal policy π from the set of all policies Π, formally $V(s) := \text{opt}_{\pi \in \Pi} \mathbb{E}_s^\pi[X]$.

The attacker's perceived reward is modified to reflect monitoring as:

$$R'(s, a, s') = (1 - D(s, a)) \cdot R(s, a, s')$$

The attacker seeks to compute:

$$V^*(s) = \max_a \sum_{s'} P(s'|s, a) \left[R'(s, a, s') + \gamma V^*(s')\right]$$

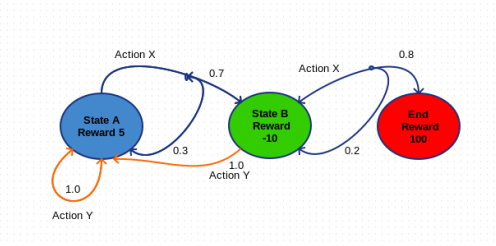

Figure II: Example MDP Transition Diagram with Actions and Rewards.

1. *Ransomware States-Actions*

We was unable to find a publicly available dataset detailing the specific actions performed by ransomware on a target machine. To address this, we executed a ransomware sample within a well-isolated virtual machine environment. Using Procmon, we captured a detailed log file containing the major system-level actions performed by the ransomware during its execution.

Figure III: System-level activity log captured during WannaCry ransomware execution using Procmon.

After collecting detailed logs of the WannaCry ransomware's behavior using Procmon, we performed a manual analysis to extract the key system-level actions it performs during execution. These actions were then abstracted into a set of meaningful states and actions, which collectively represent the ransomware's internal logic and external interactions with the operating system. This abstraction is crucial for modeling the attacker's decision-making process in a game-theoretic or Markovian framework. The following table summarizes the identified states that the ransomware transitions through, along with the corresponding actions that trigger these transitions.

Table I: Actions

Actions	Description
RegQuery	Queries registry to detect sandbox or analysis artifacts
ElevatePrivs	Executes privilege escalation techniques
Sleep	Uses system sleep or delays to avoid triggering sandbox time- outs.
ScanNetwork	Maps network layout and finds vulnerable systems.
OpenSocket	Opens a socket to communicate with the attacker's C&C server.
Encrypt	Encrypts files using cryptographic routines.
Data Exfiltration	Transfers filtered data to a remote attacker location.

Table II: States

States	Description
Init	Malware starts execution. No action has been performed yet
CheckEnv	Checks for sandbox, VM, or analysis tools via registry queries or environment inspection
PrivilegeEscalation	Attempts to gain administrative rights for deeper system access.
NetworkScan	Scans the network to find other machines or resources to target.
C&C	Establishes contact with a remote Command & Control server
Encryption	Begins encrypting files as part of ransomware activity
FilteredData	Exfiltrates selected sensitive files to the attacke.

.

Table III: Attacker's Complete Strategy: Transition Probabilities and Rewards

Start State	End State	Action	Probability	Reward	Notes
Init	CheckEnv	RegQuery	0.10	-1	Sandbox detection
Init	PrivilegeEscalation	ElevatePrivs	0.10	0	Try admin rights
Init	Delay	Sleep	0.20	+2	Stall to evade
Init	NetworkScan	ScanNetwork	0.25	+3	Discover network
Init	C&C	OpenSocket	0.35	+5	Remote contact
CheckEnv	Delay	Sleep	0.50	+2	Delay after check
CheckEnv	PrivilegeEscalation	ElevatePrivs	0.30	0	Escalate post-check
CheckEnv	C&C	OpenSocket	0.20	+4	Contact attacker early
PrivilegeEscalation	Delay	Sleep	0.25	+1	Back to hiding
PrivilegeEscalation	NetworkScan	ScanNetwork	0.25	+4	Scan with higher access
PrivilegeEscalation	C&C	OpenSocket	0.25	+5	Stealth contact
PrivilegeEscalation	CheckEnv	RegQuery	0.25	-1	Recheck for analysis tools
Delay	CheckEnv	Sleep	0.10	-1	Loop evasion
Delay	PrivilegeEscalation	ElevatePrivs	0.10	0	Retry privilege
Delay	C&C	OpenSocket	0.20	+4	Beacon to attacker
Delay	NetworkScan	ScanNetwork	0.10	+3	Retry lateral scan
Delay	Delay	Sleep	0.20	0	Remain inactive
Delay	Encryption	Encrypt	0.30	+8	Begin encryption phase
NetworkScan	C&C	OpenSocket	1.00	+3	Follow up after scan
C&C	Delay	Sleep	0.10	+1	Delay after instruction
C&C	Encryption	OpenSocket	0.35	+6	Start ransomware
C&C	FilteredData	Data Exfiltration	0.40	+9	Steal files
C&C	NetworkScan	ScanNetwork	0.15	+2	Recon as command

To model the ransomware's behavior as a Markov Decision Process, we assign transition probabilities to each state-action-state triple based on empirical evidence extracted from system-level execution logs. These logs generated by tools such as Procmon record sequences of observed attacker actions during the malware's runtime. Each log entry represents a transition from a current state s to a successor state s', triggered by a specific action a.

Let:

- S be the set of behavioral states,
- A be the set of attacker actions,
- $n_{s,a,s'}$ be the number of times the malware, when in state s, performed action a and transitioned to state s' (as recorded in the log file),

- $\sum_{s'' \in S} n_{s,a,s''}$ be the total number of observed transitions from state s under action a.

The empirical transition probability $\delta(s,a)(s')$ is estimated as:

$$\delta(s,a)(s') = \frac{n_{s,a,s'}}{\sum\limits_{s'' \in S} n_{s,a,s''}} \tag{1}$$

This definition ensures that the sum of probabilities over all possible successor states s' for a given (s,a) pair is normalized:

$$\sum_{s' \in S} \delta(s,a)(s') = 1 \tag{2}$$

The reward values in the transition model quantify the strategic benefit or cost of each action from the attacker's perspective. They are designed as follows:

- **Negative rewards (-1):** Assigned to actions that risk detection or are redundant (e.g., repeated sandbox checks or evasion loops).
- **Zero rewards (0):** Neutral actions that neither progress the attack significantly nor incur a cost (e.g., privilege escalation attempts or inert delays).
- **Positive rewards (+1 to +5):** Reflect intermediate attack steps like network scanning or establishing a C&C connection.
- **High rewards (+6 to +9):** Indicate key objectives such as file encryption or data exfiltration, which represent the main goals of ransomware behavior.

C. *Solution Concept: Stackelberg Simulation with Monitoring Strategies*

In this simulation-based approximation of Stackelberg Security Games, we consider a leader-follower game where the defender (leader) commits to a *pure monitoring strategy*, and the attacker (follower) responds optimally by computing a best-response policy through value iteration on a Markov Decision Process. This models an attacker who learns the defender's monitoring scheme and then optimizes its malware behavior accordingly.

To evaluate the effectiveness of various defender strategies, we simulate four different configurations of the monitoring function $D(s,a)$, and compute the resulting attacker reward $V^*(s)$ using the value iteration algorithm. The goal is to measure how defender surveillance impacts the attacker's ability to maximize its cumulative reward in the malware execution process.

- No Monitoring : $D(s,a) = 0$ for all state-action pairs.
- Focused Monitoring: Only two critical transitions are monitored:
  - Delay → Encrypt
  - C&C → FilteredData
- Random Monitoring: Six state-action pairs are randomly selected and monitored.
- Full Monitoring: All 22 state-action pairs are monitored.

For each strategy:

1. The reward matrix $R(s,a,s')$ is adjusted using:
$$R'(s,a,s') = (1 - D(s,a)) \cdot R(s,a,s')$$
2. Value iteration is applied with discount factor $\gamma = 0.9$ until convergence (threshold $\epsilon = 10^{-3}$).
3. The total attacker utility is computed as:
$$\sum_{s \in S} V^*(s)$$

1. *Comparison of Attacker Reward Across Defender Strategies*

To quantify the effectiveness of different monitoring strategies, we simulate the attacker's behavior under four distinct defender configurations. Each configuration corresponds to a different assignment of the monitoring function $D(s,a)$, which determines whether a given action in a given state is being observed by the defender.

The attacker then responds optimally by computing a best-response policy via value iteration over a modified reward model $R'(s,a,s') = (1 - D(s,a)) \cdot R(s,a,s')$, which penalizes monitored actions. The cumulative attacker utility $\sum_{s \in S} V^*(s)$ is computed for each strategy.

The following algorithm summarizes the evaluation process:

Algorithm 1 Compare Attacker Reward Across Defender Strategies

1: Define all possible (s, a) transitions from the MDP model
2: Define four defender strategies:
- No Monitoring: $D(s, a) = 0$ for all pairs
- Focused Monitoring: $D(s, a) = 1$ for:
  - Delay $\rightarrow$ Encrypt
  - C&C $\rightarrow$ DataExfiltration
- Random Monitoring: randomly sample 6 (s, a) pairs to monitor
- Full Monitoring: $D(s, a) = 1$ for all (s, a) pairs

3: **for** each strategy D **do**
4: Adjust rewards: $R'(s, a, s') \leftarrow (1 - D(s, a)) \cdot R(s, a, s')$
5: Run value iteration to compute the optimal value function $V^*(s)$
6: Compute total attacker reward: $\sum_{s \in S} V^*(s)$
7: **end for**
8: Output: Table of total attacker reward under each strategy

Algorithm 2 Attacker Value Iteration (Best-Response Policy)

1: **Input:** S, A, $P(s'|s, a)$, $R(s, a, s')$, $D(s, a)$
2: **Output:** $\pi^*(s)$ and $V^*(s)$
3: Initialize $V(s) \leftarrow 0$ for all $s \in S$
4: **repeat**
5: $\Delta \leftarrow 0$
6: **for all** $s \in S$ **do**
7: $v \leftarrow V(s)$
8: $V(s) \leftarrow \max_{a \in A} \sum_{s'} P(s'|s, a) \cdot [(1 - D(s, a)) \cdot R(s, a, s') + \gamma V(s')]$
9: $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
10: **end for**
11: **until** $\Delta < \epsilon$
12: **for all** $s \in S$ **do**
13: $\pi^*(s) \leftarrow \arg\max_{a \in A} \sum_{s'} P(s'|s, a) \cdot [(1 - D(s, a)) \cdot R(s, a, s') + \gamma V(s')]$
14: **end for**
15: **return** π^*, V

The second algorithm computes the attacker best responses via value iteration:

$$V^*(s) = \max_{a \in A} \sum_{s' \in S} P(s'|s, a) \left[R'(s, a, s') + \gamma V^*(s')\right]$$

The process iterates until convergence (threshold $\epsilon = 10^{-3}$) using a discount factor $\gamma = 0.9$. The optimal policy is computed as:

$$\pi^*(s) = \arg\max_{a \in A} \sum_{s'} P(s'|s, a)[R'(s, a, s') + \gamma V^*(s')]$$

Result Interpretation

Table IV

Defender Strategy	Total Attacker Reward
No Monitoring	19.608
Random Monitoring	14.884
Focused Monitoring	14.210
Full Monitoring	0.000

Table IV shows the cumulative attacker reward across four defender configurations. As expected, the No Monitoring strategy leads to the highest attacker reward. By contrast,Full Monitoring completely suppresses attacker actions, resulting in a reward of zero.

Notably, Focused Monitoring (monitoring only two critical transitions) performs nearly as well as Random Monitoring (monitoring six arbitrary transitions), demonstrating the efficiency of targeted surveillance. This highlights that strategic monitoring can reduce attacker utility without incurring high observational costs.

2. Defender Utility Modelling

While aggressive monitoring can suppress attacker actions, it also comes with an operational cost. Therefore, defenders must balance between the benefit of detection and the overhead of monitoring. We define the defender's utility as the expected detection reward minus the total monitoring cost. The defender's utility is computed as:

$$U_{\text{defender}} = \mathbb{E}[\text{Detection Utility}] - C(D)$$

where:

- $D \subseteq S \times A$ is the set of monitored state–action pairs,
- $\mathbb{E}$[Detection Utility] is the expected reward from catching malicious behavior,
- $C(D)$ is the cost of monitoring, modeled as a linear function: $C(D) = c \cdot |D|$, with $c > 0$.

A negative U_{defender} indicates that the cost outweighs the detection benefit, which may render a strategy economically suboptimal, even if effective in neutralizing attacks.

Algorithm: Compute Defender Utility:

Algorithm 3 Compute Defender Utility

Require: Transition list T with tuples (s, a, p) ▷ probability p of (s, a) being executed
Require: Monitored set $D \subseteq S \times A$
Require: Detection reward per catch r_{det}
Require: Cost per monitored pair c
$U \leftarrow 0$
for each $(s, a, p) \in T$ **do**
  if $(s, a) \in D$ **then**
    $U \leftarrow U + p \cdot r_{\text{det}}$
  end if
end for
$U \leftarrow U - c \cdot |D|$ ▷ Subtract monitoring cost
return U

Results interpretation

Strategy	$\lvert D \rvert$	U_{defender}
No Monitoring	0	0.0
Focused Monitoring	2	0.5
Random Monitoring	6	3.5
Full Monitoring	11	–3.0

TABLE V: DEFENDER UTILITY UNDER DIFFERENT MONITORING INTENSITIES.

The negative utility in the Full Monitoring case confirms that extensive surveillance can be economically inefficient, especially when marginal detection gain is overshadowed by the cost of monitoring.

D. *Analysis of Defender Strategies on Attacker Reward*

Figure IV presents a comparative analysis of four defender strategies: *No Monitoring*, *Focused Monitoring*, *Random Monitoring*, and *Full Monitoring*. The x-axis corresponds to each strategy (mapped as indices 0 to 3), while the y-axis shows the total expected reward for the attacker under each scenario.

The results clearly indicate a downward trend in attacker utility as the level of monitoring increases. In the absence of any monitoring (index 0), the attacker achieves the maximum reward of approximately 19.6. When the defender employs a focused monitoring strategy (index 1), targeting only a few critical state-action pairs, the attacker's reward drops significantly to around 14.2. This value remains nearly the same under random monitoring (index 2), suggesting that the effectiveness of random selection is similar to targeted selection in this configuration.

However, under full monitoring (index 3), where all possible malicious transitions are observed, the attacker's reward drops to zero. This indicates complete deterrence or disruption of the attack sequence, validating the theoretical effectiveness of exhaustive monitoring. Nonetheless, this comes with a potentially high cost in practice, which is explored further in the next section with the defender utility model.

Visual Comparison: Utility vs Monitoring Cost

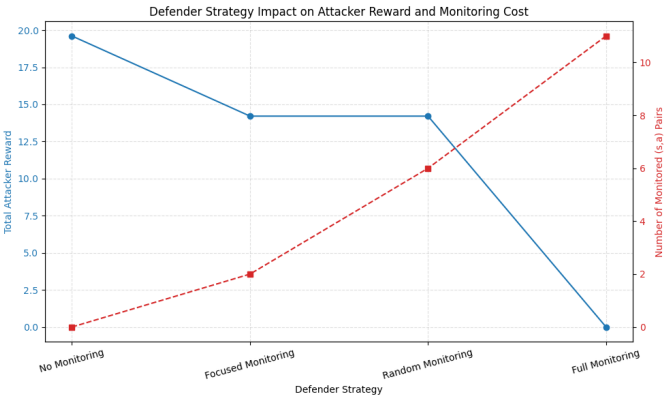

FIGURE IV: IMPACT OF DEFENDER STRATEGY ON ATTACKER REWARD AND MONITORING COST. THE LEFT AXIS SHOWS THE ATTACKER'S TOTAL REWARD; THE RIGHT AXIS SHOWS THE NUMBER OF MONITORED TRANSITIONS. FOCUSED MONITORING ACHIEVES NEAR-OPTIMAL SUPPRESSION WITH MINIMAL MONITORING COST.

III. LIMITATIONS

While this simulation provides valuable insights into attacker–defender dynamics within a Stackelberg framework, several limitations should be noted:

- **Static Defender Strategy:** The defender commits to a fixed monitoring policy, which may not reflect adaptive or learning-based real-world defenses that evolve based on observed attacker behavior.
- **Binary Monitoring Model:** The monitoring function $D(s, a) \in \{0, 1\}$ assumes perfect detection or no detection at all. This simplification ignores detection noise, false positives, and varying levels of observability.
- **Simplified Cost Function:** The monitoring cost is modeled as a linear function of the number of monitored pairs. In practice, costs may vary by transition or depend on system load, resource constraints, or data volume.
- **Limited Strategy Space:** The defender strategies evaluated (No, Random, Focused, Full) represent only a small subset of the possible monitoring configurations. An exhaustive or optimized search over the strategy space is not performed.
- **Markov Assumption:** The attacker behavior is modeled as an MDP, assuming that the next state depends only on the current state and action. This excludes more complex temporal dependencies or multi-stage attack planning.

- **Single Attacker Model:** The simulation assumes a single rational attacker. It does not account for diverse attacker profiles, stochastic behavior, or interactions between multiple adversaries.

Addressing these limitations in future work would improve the realism and applicability of the proposed framework in operational cybersecurity environments.

IV. Conclusion

This work presented a game-theoretic framework for detecting evasive ransomware by modeling attacker behavior as a Markov Decision Process and simulating strategic defender–attacker interactions within a Stackelberg Security Game. Through empirical modeling of ransomware execution logs, we identified critical behavioral transitions and demonstrated how selective monitoring of these transitions can effectively degrade attacker performance.

Our experiments showed that focused monitoring, even when applied to only two high-impact transitions, can achieve suppression levels comparable to broader random monitoring while significantly reducing surveillance costs. The introduction of a defender utility function further revealed that excessive monitoring, despite achieving total deterrence, may be economically suboptimal due to high operational costs.

These findings highlight the value of strategic, cost-aware defense planning over exhaustive monitoring approaches. By anticipating attacker adaptation and targeting key points of the attack chain, defenders can maintain strong security postures without overextending resources.

Future research will address current limitations by incorporating adaptive defender strategies, probabilistic detection models, and optimization-based monitoring selection. Expanding the framework to handle multiple attacker profiles and more complex temporal dependencies will also enhance its applicability to real-world cybersecurity environments.

References

[1] Rad, Babak Bashari, Maslin Masrom, and Suhaimi Ibrahim. "Camouflage in malware: from encryption to metamorphism." International Journal of Computer Science and Network Security 12.8 (2012): 74-83

[2] You, Ilsun, and Kangbin Yim. "Malware obfuscation techniques: A brief survey." Broadband, Wireless Computing, Communication and Applications (BWCCA), 2010 International Conference on. IEEE, 2010

[3] Beaucamps, Philippe. "Advanced polymorphic techniques." International Journal of Computer Science 2.3 (2007): 194-205.

[4] Yim, I. (2010). Malware obfuscation techniques: A brief survey. *2010 International Conference on Broadband, Wireless Computing, Communication and Applications (BWCCA)*, 297–300. Retrieved from `https://www.researchgate.net/publication/221420990_Malware_Obfuscation_Techniques_A_Brief_Survey`

[5] M. D. Preda, *Code Obfuscation and Malware Detection by Abstract Interpretation*. [Online]. Available: `https://iris.univr.it/retrieve/handle/11562/337972/3306/main.pdf`. Accessed: Nov. 12, 2019.

[6] W. Yan, Z. Zhang, and N. Ansari, "Revealing packed malware," *IEEE Security & Privacy*, vol. 6, no. 5, pp. 65–69, Sept. 2008.

[7] Wikipedia, *Game theory*, `https://en.wikipedia.org/wiki/Game_theory`, accessed July 2025.

[8] Revan MacQueen, Natalie Bombardieri, James R. Wright, and Karim Ali.

[9] Sriram Sankaranarayanan and Natasha Sharygina (Eds.). *Tools and Algorithms for the Construction and Analysis of Systems*. 29th International Conference, TACAS 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2023, Paris, France, April 22–27, 2023, Proceedings, Part I. Lecture Notes in Computer Science, vol. 13993, Springer, 2023. *Game-Theoretic Malware Detection*.