RansomAI: AI-powered Ransomware for Stealthy Encryption

Jan von der Assen[1], Alberto Huertas Celdrán[1], Janik Luechinger[1]
Pedro Miguel Sánchez Sánchez[2], Gérôme Bovet[3], Gregorio Martínez Pérez[2], Burkhard Stiller[1]

[1]Communication Systems Group CSG, Department of Informatics, University of Zurich UZH, CH–8050 Zürich, Switzerland
[vonderassen, huertas, stiller]@ifi.uzh.ch, janik.luechinger@uzh.ch
[2]Department of Information and Communications Engineering, University of Murcia, 30100–Murcia, Spain gregorio@um.es
[3]Cyber-Defence Campus, armasuisse Science & Technology, CH–3602 Thun, Switzerland gerome.bovet@armasuisse.ch

Abstract—Cybersecurity solutions have shown promising performance when detecting ransomware samples that use fixed algorithms and encryption rates. However, due to the current explosion of Artificial Intelligence (AI), sooner than later, ransomware (and malware in general) will incorporate AI techniques to intelligently and dynamically adapt its encryption behavior to be undetected. It might result in ineffective and obsolete cybersecurity solutions, but the literature lacks AI-powered ransomware to verify it. Thus, this work proposes *RansomAI*, a Reinforcement Learning-based framework that can be integrated into existing ransomware samples to adapt their encryption behavior and stay stealthy while encrypting files. *RansomAI* presents an agent that learns the best encryption algorithm, rate, and duration that minimizes its detection (using a reward mechanism and a fingerprinting intelligent detection system) while maximizing its damage function. The proposed framework was validated in a ransomware, Ransomware-PoC, that infected a Raspberry Pi 4, acting as a crowdsensor. A pool of experiments with Deep Q-Learning and Isolation Forest (deployed on the agent and detection system, respectively) has demonstrated that *RansomAI* evades the detection of Ransomware-PoC affecting the Raspberry Pi 4 in a few minutes with >90% accuracy.

Index Terms—Ransomware, Reinforcement Learning, Artificial Intelligence, Malware, Evasion

I. Introduction

With the growing progress of digitalization, companies have become more dependent on information systems that uphold their business missions. It has influenced a recent increment in malware-based attacks targeting heterogeneous enterprises [1]. Among existing attack vectors, ransomware is one of the most significant threats affecting companies due to its impact on data and economic losses. In 2022, industries experienced a rise of 87% in ransomware attacks [2]. Moreover, although ransom payments are slowly declining [2], ransomware is still a highly impactful threat to most companies. As an example, IBM assessed in 2022 that the average loss of a ransomware attack was 4.54M USD [3].

Detecting cyberattacks and ransomware, in particular, is the first step toward mitigating their impact. As outlined by recent work [4], dynamic detection approaches incorporating behavioral data into machine and deep learning (ML and DL) techniques have demonstrated to be highly effective against ransomware. However, while these behavioral-based approaches are not vulnerable to obfuscation (as static approaches), they rely on specific assumptions. In particular, both classification and anomaly detection approaches assume that malicious behavior is stable and static enough to allow ML/DL models to differentiate it from normal or benign behavior [5].

However, integrating Artificial Intelligence (AI) into ransomware samples could change it by adding intelligent dynamicity to encryption behaviors. It would complicate the detection since ransomware samples could learn the encryption rate that maximizes impact while minimizing its detection. Furthermore, the encryption rate could automatically change depending on the device status. Therefore, the literature presents an open challenge that focuses on studying the suitability of integrating AI-based techniques into ransomware to learn how and when to encrypt real devices while staying undetected. This challenge is vital to later, in a second stage, evaluate the detection capabilities of current cybersecurity mechanisms and update them if needed.

This work fills this research gap and proposes *RansomAI*, a framework using Reinforcement Learning (RL) and fingerprinting to maximize the stealth capabilities of ransomware samples while maximizing their damage function. *RansomAI* contains an RL-based agent that learns the optimum ransomware behavior (encryption rate, duration, and algorithm combination) according to the reward received for each ransomware configuration and the device behavior. The reward is calculated according to the output of a fingerprinting and ML-based anomaly detection system and the encryption rate of the selected ransomware configuration. A prototype of *RansomAI* (available in [6]) that implements Deep Q-Learning and Isolation Forest (for the agent and the anomaly detector, respectively) has been integrated into a real ransomware called Ransomware-PoC. The prototype effectiveness has been evaluated in a Raspberry Pi 4, acting as a crowdsensor through a pool of experiments with six configurations of Ransomware-PoC dealing with the encryption algorithm, rate, and duration. The experiments have demonstrated that *RansomAI* evades the detection of Ransomware-PoC in a few minutes and with high accuracy.

The remainder of this paper is structured as follows. Section II introduces the background and related approaches.

Then Section III presents the problem and scenario tackled by Section IV with the framework architecture and its implementation. The performance of the solution is demonstrated in the experiments presented in Section V. Finally, Section VI draws conclusions from the results.

II. RELATED WORK

In contrast to many works that demonstrate the applicability of AI for defense, there are only a few papers discussing the offensive perspective. TABLE I compares the most relevant and recent ones. In [7]–[9], the authors leveraged RL to evade a static detection system. Here, static detection refers to an ML-based system that detects whether a piece of software is malicious based on the static structure of the malware sample. Thus, the three approaches consider RL to find an optimal way to structure the malware sample before executing the payload on the victim's premises. This is achieved by relying on an adversarial technique to evade the ML-based detection system. None of these solutions consider ransomware, and only [8] presents experimental results in a realistic scenario where commercial antivirus systems were deployed in cloud-based virtual machines.

Aside from using RL, [10] presented how Genetic Algorithms can help with byte-level modifications to evade malware detection. [11] proposed an ML model to inject strategic system failure into a cyber-physical system. The model was not used to address evasion but to optimize the time and location of the failure. Therefore, no adversarial techniques were used in that work. *RoboTack* was proposed in [12] to estimate attacks impact instead of evading detection. A neural network with three hidden layers gave insight into the optimal deployment target, time, and strategy. Like the previous two approaches, *RoboTack* was only evaluated in a simulated environment.

While all aforementioned evasion approaches deal with a static detection system, *DeepLocker* [13] is the first generic malware that does not rely on static obfuscation for evasion. In contrast, it achieves evasion by employing dynamic obfuscation of arbitrary sources before the breach event. *DeepLocker* encrypts the payload and injects itself into the target system, making static analysis infeasible. The trigger conditions are transformed into a deep neural network (DNN), encoding it in another black box system. The DNN was trained on multiple target markers (*e.g.,* system-level features, geolocation, input

and output systems). Upon recognizing the target, these markers can be converted into the encryption key, unleashing the payload. *DeepLocker* was evaluated in a real scenario.

In conclusion, while several approaches leverage AI for malware, most assume a static defense model. Furthermore, most solutions rely on adversarial attacks for evasion, thereby focusing on optimizing the malware before its execution. Finally, most works do not present evaluation in realistic scenarios, and no work is focused on ransomware, which is currently one of the most harmful malware families.

III. PROBLEM STATEMENT & SCENARIO DESCRIPTION

Some of the most recent and promising cybersecurity solutions combine fingerprinting and ML to successfully detect anomalies and classify ransomware samples [14]. However, they do not consider ransomware adapting its encryption mechanisms according to specific criteria to stay undetected. Therefore, this work seeks to answer the following question: Is it possible to effectively and automatically evade novel detection systems by incorporating AI-based techniques into ransomware? Here, the main goal of AI would be to learn and select in an online and autonomous fashion what malicious configuration maximizes encryption while minimizing its detection. In addition, if evasion is possible, the next question would be: How much time is needed to adapt the ransomware behavior? This work considers the following scenario to tackle these questions.

- A Raspberry Pi 4 acting as a real sensor of ElectroSense [15], a platform that monitors the radio frequency spectrum. The Raspberry Pi would host the intelligent fingerprinting detection system that must be evaded. More information about the potential detection mechanism can be found in [14].
- An extended version of Ransomware-PoC composed of i) a client running on the Raspberry Pi and encrypting files by using different algorithms, rates, and duration; and ii) a command and control (C&C) server that selects between six different configurations (see TABLE II) the encryption setup executed by the client. It is important to mention that the C&C only has control over the client, so it is not able to change the configuration of the Raspberry Pi or the anomaly detector.

IV. *RansomAI* ARCHITECTURE

This section presents the design and implementation details of *RansomAI*, a framework adding intelligence to existing

TABLE I
RELATED WORK USING AI FOR OFFENSIVE PURPOSES

Paper	Attack	Obf.	Evasion	Tech.	Execution	Eval.
[7] 2018	Adv.	Yes	Static	RL	Offline	Sim.
[8] 2022	Adv.	Yes	Static	RL	Offline	Real
[9] 2021	Adv.	Yes	Static	RL	Offline	Sim.
[10] 2019	Malw.	Yes	Static	GA	Offline	Sim.
[11] 2019	Malw.	No	-	ML	Online	Simul.
[12] 2020	Adv.	No	-	ML	Online	Sim.
[13] 2022	Malw.	Yes	Hybrid	DL	Offline	Real
This	Malw.	No	Dynamic	RL	Online	Real

TABLE II
RANSOMWARE CONFIGURATIONS

Conf.	Algorithm	Rate (B/s)	Burst Duration	Burst Pause (s)
1	ChaCha20	16	1 file	60
2	AES-CTR	565'565.65	unlimited	0
3	Salsa20	632'834.80	unlimited	0
4	AES-CTR	500	10 s	5
5	ChaCha20	200	20 s	40
6	Salsa20	200	120 s	30

Fig. 1. *RansomAI* Architecture

ransomware samples to evade detection systems while maximizing encryption. Fig. 1 shows the main components of *RansomAI*, which is available in [6].

In summary, the Agent uses RL to learn and deploy (interacting with the C&C) the best ransomware configuration (from a list of existing ones, see TABLE II). The learning process is driven by the feedback provided by a Reward function, which prioritizes the ransomware configuration that maximizes the encryption rate while minimizing its detection. For that, the Reward function computes the output of an Anomaly Detector that uses behavioral fingerprinting and ML, and the encryption rate of the selected ransomware configuration. More in detail, the Anomaly Detector (which tries to mimic a genuine system potentially deployed in the Raspberry Pi) compares the current state (behavior) of the Raspberry Pi with its normal one (monitored when it is not under attack). The Fingerprint Monitor is responsible for continuously gathering the Raspberry Pi states. All the previous components are deployed on the server where the C&C runs, except the Fingerprint Monitor, which runs on the Raspberry Pi with the Ransomware client. More details about each component are provided below.

A. State

A state is the agent vision of the environment (Raspberry Pi) at a given time. Modeling states precisely is crucial to allow the agent to understand the impact of each action on the environment and learn proper actions.

RansomAI proposes behavioral fingerprinting to represent states. In particular, it uses the *perf* Linux command to collect different events from *system calls*, *CPU*, *device drivers*, *scheduler*, *network*, *file system*, *virtual memory*, and *random numbers* families. The reason for selecting these heterogeneous data sources is to detect small perturbations produced by encryption phases and later evade robust detection systems. More in detail, 50 features were chosen from the previous families. For this selection, 103 features were initially monitored in time windows of 5 s (decided according to previous work [14]) during 8 hours of Raspberry Pi normal behavior (sensor without being attacked). Then, duplicated, temporal, and constant features were removed during a data cleaning

process. All remaining features were plotted, and 28 features whose normal behavior was volatile overall fingerprints were manually identified and subsequently dropped. Finally, features with more than 99% correlation were removed. After the whole process and as mentioned, 50 features were selected to create the device fingerprints.

B. Action

Actions are the way in which the agent interacts with the environment. In *RansomAI*, actions correspond to the execution of the six ransomware configurations created in the extended version of Ransomware-PoC (see TABLE II). It is important to mention that the original version of Ransomware-PoC provides a fixed encryption configuration (in terms of algorithm and rate), and this work extended it with new functionality in terms of different algorithms, encryption rates, and pauses between bursts.

C. Anomaly Detector

The Anomaly Detector component is critical for *RansomAI* since it tries to mimic the functionality of existing novel detection systems that might be deployed on the Raspberry Pi to detect ransomware attacks. Specifically, its outputs are used by the Reward function to provide positive or negative feedback for each encryption configuration.

RansomAI proposes the combination of unsupervised ML and behavioral fingerprint. More in detail, it considers an anomaly detection system to model the normal behavior of the Raspberry Pi (acting as a sensor of ElectroSense) and detect anomalies produced by each ransomware configuration. The behavior of the device is represented by the 50 features selected to model the environment state (previously described). These features cover as many different data sources as possible to detect changes in device resource usage. Then, different unsupervised ML models such as Isolation Forest, Autoencoder, Local Outlier Factor, and One Class-Support Vector Machine were trained with the normal behavior of the device (running for 8 hours) and evaluated with the normal and six ransomware configurations. Isolation Forest was selected for the prototype implementation after analyzing the performance of each model when detecting normal and under-attack behaviors. More in detail and as can be seen in TABLE III, normal behavior and configurations 2 and 3 were correctly detected with almost 89% True Negative Rate (TNR) and 0% False Negative Rate (FNR), respectively. Then, configurations 1, 4, 5, and 6 were incorrectly detected as normal behavior with 77-91% FNR. These results were achieved with a 5% contamination factor in Isolation Forest hyperparameters. At this point, it can be concluded that it is feasible to encrypt files without being detected by novel works. However, evaluating if *RansomAI* can automatically learn the optimum encryption configuration within an acceptable time is still essential.

D. Reward

Positive and negative rewards allow the Agent to learn if selected ransomware configurations are good or bad. *RansomAI*

provides rewards according to the output of the anomaly detector and the encryption rate of the selected configuration. In other words, if one ransomware configuration is not detected, a higher encryption rate should be more favorable than a lower encryption rate. Moreover, if the ransomware configuration is detected, a lower encryption rate is worse than a higher one. In summary, encryption time is important and considered by the reward mechanism.

RansomAI proposes two separate reward functions, one for hidden and another for detected encryption. After several experiments and fine-tuning, $R_{hid}(r) = 10 * ln(r + 1) + h$ and $R_{det}(r) = \frac{-d}{max(r,1)} - d$ are the two proposed reward functions. In both functions, h and d are constants aiming to distinguish clearly between rewards for hidden and detected behaviors, and r is the current encryption rate. The constants were set as $h = 0$ and $d = 20$ to avoid impacting the weights in the network with unnecessarily high rewards. Therefore, with the reward functions and the probability of being detected (see TABLE III), the Agent should learn that configuration 4 is the most convenient because it achieves the best expected average reward $(0.8018*62.166+0.1982*(-20.04) = 45.87)$. More in detail, following configuration 4, Ransomware-PoC could encrypt approximately 200 KB per minute, 12 MB per hour, 288 MB per day, and 8.6 GB per month without being detected.

E. Agent

The agent of *RansomAI* considers RL and learns following a trial-and-error approach. When the agent takes one action (selects a ransomware configuration), the environment (Raspberry Pi) changes to a new state (behavioral fingerprint called afterstate) and receives a reward. This loop is repeated, and sequences of the previous steps (state, action, afterstate, and reward) are called episodes. An episode concludes when no more actions can be taken.

In *RansomAI*, since the state space is vast due to the number of features (50) modeling the behavior fingerprint and their continuous values, a tabular approach for the RL model is not feasible. Instead, a neural network is used together with the Deep Q-Learning algorithm. Q-learning is an off-policy temporal difference control algorithm defined by $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma\ max_a\ Q(S_{t+1}, a) - Q(S_t, A_t)]$. Q is the learned action-value function that approximates the optimal action-value function based on value estimates of state-action pairs (S, A) and reward R independent of the followed policy. In addition, the agent follows an epsilon-greedy policy that decides with probability ϵ to take a random action, which may not coincide with the current estimated optimal action. This policy ensures the agent explores different

TABLE III
TPR AND FNR OF ISOLATION FOREST

Normal	Conf. 1	Conf. 2	Conf. 3	Conf. 4	Conf. 5	Conf. 6
88.94	91.62	0.62	0.21	80.18	82.05	77.38

actions to find the optimum one. Regarding the Agent neural network, it has three fully-connected layers. The input layer has 50 neurons (matching the features of the fingerprint), the hidden layer has 25 neurons, and the output layer size is 6, corresponding to the number of ransomware configurations. The details regarding the hyperparameters selection, such as exploration versus exploitation (ϵ and δ), learning rate (α), discount factor (γ), hidden neurons, activation function, and weights initialization, are provided in Section V.

V. EXPERIMENTS

This section evaluates the performance of *RansomAI* while learning the optimum configuration to encrypt the Raspberry Pi presented in Section III. For that, the First experiment performs an individual search of hyperparameters to optimize the agent accuracy and learning time. Then, the second shows the learning accuracy across time.

A. Hyperparameters Search

The goal of this experiment is to find the optimum configuration of hyperparameters to maximize accuracy and reduce the training time of *RansomAI*. In this sense, the following hyperparameters of the RL-based Agent are considered: i) exploration settings (ϵ, δ, α, and γ), ii) number of hidden neurons in the neural network, iii) activation functions used in the forward and backward pass of the neural network, and iv) method for weight initialization in the neural network. The following baseline configuration is fixed while one hyperparameter per test is fine-tuned. Episodes = 5,000, $\epsilon = 0.4$, $\delta = 0.01$, $\alpha = 0.0050$, $\gamma = 0.10$, hidden neurons = 40, activation = *Log and SiLU*, weights = *He*.

1) Exploration vs. Exploitation: It presents the trade-off between exploring new actions to discover new states and exploiting learned optimal actions. For different exploration and exploitation configurations, TABLE IV shows the average learning time of the Agent, the accuracy after training, the number of episodes required to find the best encryption configuration, and the average Q-value difference (AQD). As can be seen, the exploration does not significantly impact the training or learning duration, as most values for the average training time are similar. Instead, the impact is manifested in the final accuracy as well as the number of episodes and AQD. The best setting ($\epsilon = 0.20, \delta = 0.01$) has the highest performance considering the trade-off for exploration and exploitation. It offers great accuracy, 99.63%, close to the maximum of 99.84%, needs one of the lowest numbers of episodes, and achieves high differences between the Q-values of configuration 4 and the closest ones (5 and 6).

2) Learning Rate: It represents the weight given to the update of the Q-values. If the step size is too small, the current Q-value estimates will likely get stuck in a local maximum. In contrast, when α is too large, convergence is impossible in most cases. TABLE V shows that adjusting the learning rate causes significant differences in the average learning time. In this sense, it was infeasible to do a complete hyperparameter exploration given the infinite value space.

TABLE IV
AGENT PERFORMANCE FOR DIFFERENT EXPLORATION SETTINGS

Settings	Learning (min)	Accuracy	Episodes	AQD
$\epsilon = 0.10, \delta = 0.01$	140.2	99.84%	50	80.1
$\epsilon = 0.20, \delta = 0.01$	141.8	99.63%	40	125.1
$\epsilon = 0.30, \delta = 0.01$	139.1	99.34%	40	60.5
$\epsilon = 0.40, \delta = 0.01$	139.4	99.24%	150	60.5
$\epsilon = 0.50, \delta = 0.01$	126.7	99.24%	300	59.5
$\epsilon = 0.10, \delta = 0.001$	127.0	98.75%	400	33.3
$\epsilon = 0.20, \delta = 0.001$	126.0	97.19%	120	55.3
$\epsilon = 0.30, \delta = 0.001$	131.4	52.43%	260	13.5
$\epsilon = 0.40, \delta = 0.001$	130.1	94.64%	900	61.4
$\epsilon = 0.50, \delta = 0.001$	124.7	92.81%	1400	27.0

However, $\alpha = 0.0050$ is the best configuration considering the average training time and episodes to learn the best encryption configuration (configuration 4). Although other variants achieved a higher AQD, they required more episodes and time to find configuration 4 as the best. In summary, it trades the speed against distinction capabilities as the Q-values for configuration 4 are very close to those of configurations 5 and 6. Nevertheless, given the presented selection of variants, $\alpha = 0.0050$ is considered the best approximation of the optimal setting due to its clear advantage in speed and accuracy.

TABLE V
AGENT PERFORMANCE FOR DIFFERENT LEARNING RATES

Learning rate	Learning (min)	Accuracy	Episodes	AQD
$\alpha = 0.0001$	331.3	00.10%	–	0.7
$\alpha = 0.0005$	124.9	99.24%	320	69.7
$\alpha = 0.0010$	149.5	99.34%	200	74.0
$\alpha = 0.0050$	139.4	99.24%	150	60.5
$\alpha = 0.0100$	162.1	99.25%	150	45.7

3) Discount Factor: It controls the importance of future estimations over the current one. With γ approaching 1, the agent considers future value estimations more strongly. In contrast, with $\gamma = 0$, the agent only maximizes immediate estimates. TABLE VI shows the impact of the discount factor on the average learning time, the accuracy, the number of episodes, and the AQD values to find configuration 4 as the best one. Interestingly, the accuracy remains relatively stable for a discount factor between 0.2 and 0.6. Analyzing the results, the setting $\gamma = 0.10$ is considered best as it has the shortest average training time and the third lowest number of episodes. In addition, it provides a reasonable distinction of Q-values from configuration 4 (the best) and the rest.

4) Hidden Neurons: TABLE VII shows how the number of neurons in the hidden layer affects the learning time, accuracy, number of episodes, and AQD of the agent. Sizes from 10 to 50 neurons were tested, achieving the best setup configuration with 25 hidden neurons.

5) Activation Function: TABLE VIII lists the activation functions evaluated in the first and second layers of the neural network and their impact on the learning process. As can be seen, the activation function significantly impacts the average training time. More in detail, the Log-SiLU setting obtains

TABLE VI
AGENT PERFORMANCE FOR DIFFERENT DISCOUNT FACTORS

Discount	Learning (min)	Accuracy	Episodes	AQD
$\gamma = 0.00$	154.1	54.03%	110	18.7
$\gamma = 0.10$	139.4	99.24%	150	60.5
$\gamma = 0.20$	161.8	99.22%	260	86.1
$\gamma = 0.30$	158.1	99.29%	130	81.5
$\gamma = 0.40$	163.2	99.18%	130	102.5
$\gamma = 0.50$	162.3	99.29%	300	34.4
$\gamma = 0.60$	152.0	99.21%	320	71.8
$\gamma = 0.70$	156.5	51.80%	400	63.8
$\gamma = 0.80$	187.0	51.89%	410	29.8
$\gamma = 0.90$	207.6	51.79%	300	07.3
$\gamma = 1.00$	253.0	49.69%	760	83.2

TABLE VII
AGENT PERFORMANCE FOR DIFFERENT HIDDEN NEURONS

Neurons	Learning (min)	Accuracy	Episodes	AQD
10	155.3	99.46%	220	44.6
20	155.4	99.45%	150	45.2
25	195.8	99.52%	110	63.9
30	198.7	99.42%	240	51.4
35	200.4	99.33%	100	56.6
40	139.4	99.24%	150	60.5
45	195.5	99.26%	130	42.5
50	204.0	99.21%	580	53.2

the best results because, although Log-ReLU achieves slightly better accuracy, many Q-values are equal to zero due to the dying ReLU problem.

TABLE VIII
AGENT PERFORMANCE FOR DIFFERENT ACTIVATION FUNCTIONS

Activation Func.	Learning (min)	Accuracy	Episodes	AQD
Log - Log	343.6	00.02%	–	-0.0022
Log - ReLU	110.9	99.32%	350	49.9
Log - SiLU	139.4	99.24%	150	60.5
ReLU - Log	109.2	00.25%	–	-0.3
ReLU - ReLU	334.3	00.07%	–	-10.5
ReLU - SiLU	151.4	00.18%	–	-27.9
SiLU - Log	137.2	00.27%	–	-3300
SiLU - ReLU	361.0	00.11%	–	0
SiLU - SiLU	171.5	00.12%	–	-36.7

6) Weights Initialization: Plain random-uniform distribution, Xavier uniform distribution, and He initialization were tested. The plain uniform initialization randomly selects weights from a uniform distribution in the $[0, 1]$ range. In the Xavier initialization, the distribution is dynamically scaled according to the dimensions of the previous layer. Lastly, He initialization generates random numbers selected from a standard normal distribution. After evaluating the three of them, Xavier provided the best learning time (105.4 min), accuracy (99.31%), and the number of episodes (90).

B. Agent Final Performance

According to the results obtained in the previous experiment, the Agent of *RansomAI* is configured with the following configuration of hyperparameters: $\epsilon = 0.20$, $\delta = 0.01$,

α = 0.005, γ = 0.30, hidden neurons = 25, activation functions = *Log-SiLU*. TABLE IX shows the accuracy with which the Agent selects configuration 4 for different episodes and therefore learning times. As can be seen, >96% accuracy is obtained in less than 10 minutes.

TABLE IX
AGENT PERFORMANCE WITH BEST HYPERPARMETER CONFIGURATION

Episodes	Learning (min)	Accuracy
100	2.0	91.43%
200	5.1	94.86%
300	6.5	96.32%
400	8.1	96.21%
1'000	23.9	98.61%
2'000	66.4	99.07%
5'000	172.2	99.71%

In conclusion, the obtained results demonstrated that *RansomAI* is able to learn how to evade intelligent detection systems in just a few minutes and with promising accuracy. Therefore, more efforts are needed to improve detection systems against intelligent ransomware samples.

VI. SUMMARY AND FUTURE WORK

This work proposes *RansomAI*, a framework able to intelligently and automatically adapt the encryption behaviors of ransomware and avoid being detected by dynamic defense mechanisms. The main contribution of *RansomAI* is an Agent that combines RL and device fingerprinting to learn the encryption rate, duration, and algorithm combination that maximizes encryption and minimizes detection. The learning task is driven by a reward mechanism that prioritizes the encryption rate and stealth capabilities of ransomware configurations. Stealth is evaluated using an anomaly detection system that uses ML and fingerprinting, as proposed in the literature.

RansomAI has been deployed in a real scenario composed of a Raspberry Pi 4 acting as a crowd-sensor affected by a recent ransomware called Ransomware-PoC. More in detail, the components of *RansomAI* have been integrated into Ransomware-PoC, which has been modified to dynamically adapt its algorithms, encryption rates, and duration. A pool of experiments combining Deep Q-Learning and Isolation Forest (in the Agent and detection system, respectively) has demonstrated that *RansomAI* evades the detection of Ransomware-PoC affecting the Raspberry Pi 4 in 2 minutes with >90% accuracy.

Future work plans to evaluate the functionality of *RansomAI* with different benign device behaviors to show its adaptability. It is also planned to try other malware samples, such as backdoors leaking sensitive data. Finally, it is expected to work on intelligent and adaptive detection mechanisms to detect *RansomAI*.

REFERENCES

[1] World Economic Forum, "The Global Risks Report 2023," 2023, https://www3.weforum.org/docs/WEF_Global_Risks_Report_2023.pdf, Last Visit January 2023.

[2] J. Gillum, "Ransomware Attacks on Industrial Firms Increased by 87% in 2022," https://www.bnnbloomberg.ca/ransomware-attacks-on-industrial-firms-increased-by-87-in-2022-1.1883569, 2023, last Visit March 2023.

[3] IBM, "Cost of a data breach 2022," https://www.ibm.com/reports/data-breach, 2023, last Visit March 2023.

[4] P. M. S. Sánchez, J. M. J. Valero, A. H. Celdrán, G. Bovet, M. G. Pérez, and G. M. Pérez, "A Survey on Device Behavior Fingerprinting: Data Sources, Techniques, Application Scenarios, and Datasets," *IEEE Communications Surveys & Tutorials*, vol. 23, pp. 1048–1077, 2021.

[5] P. M. S. Sanchez, A. H. Celdran, G. Bovet, G. M. Perez, and B. Stiller, "Specforce: A framework to secure iot spectrum sensors in the internet of battlefield things," *IEEE Communications Magazine*, pp. 1–7, 2022.

[6] J. Lüchinger, "RansomAI source code," https://github.com/jluech/roar_client, 2023, last Visit March 2023.

[7] H. S. Anderson, A. Kharkar, B. Filar, D. Evans, and P. Roth, "Learning to evade static PE machine learning malware models via reinforcement learning," 2018. [Online]. Available: https://arxiv.org/abs/1801.08917

[8] W. Song, X. Li, S. Afroz, D. Garg, D. Kuznetsov, and H. Yin, "Mab-malware: A reinforcement learning framework for blackbox generation of adversarial malware," in *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, ser. ASIA CCS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 990–1003. [Online]. Available: https://doi.org/10.1145/3488932.3497768

[9] R. Labaca-Castro, S. Franz, and G. D. Rodosek, "Aimed-rl: Exploring adversarial malware examples with reinforcement learning," in *Machine Learning and Knowledge Discovery in Databases. Applied Data Science Track*, Y. Dong, N. Kourtellis, B. Hammer, and J. A. Lozano, Eds. Cham: Springer International Publishing, 2021, pp. 37–52.

[10] R. L. Castro, C. Schmitt, and G. Dreo, "Aimed: Evolving malware with genetic programming to evade detection," in *2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, 2019, pp. 240–247.

[11] K. Chung, Z. T. Kalbarczyk, and R. K. Iyer, "Availability attacks on computing systems through alteration of environmental control: Smart malware approach," in *Proceedings of the 10th ACM/IEEE International Conference on Cyber-Physical Systems*, ser. ICCPS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1–12. [Online]. Available: https://doi.org/10.1145/3302509.3311041

[12] S. Jha, S. Cui, S. Banerjee, J. Cyriac, T. Tsai, Z. Kalbarczyk, and R. K. Iyer, "ML-driven malware that targets AV safety," in *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2020, pp. 113–124.

[13] M. P. Stoecklin, "Deeplocker: How AI can power a stealthy new breed of malware," *Security Intelligence, August*, vol. 8, 2018, accessed: 2022-10-26. [Online]. Available: https://securityintelligence.com/deeplocker-how-ai-can-power-a-stealthy-new-breed-of-malware/

[14] A. Huertas Celdrán, P. M. Sánchez Sánchez, M. Azorín Castillo, G. Bovet, G. Martínez Pérez, and B. Stiller, "Intelligent and behavioral-based detection of malware in iot spectrum sensors," *International Journal of Information Security*, pp. 1–21, 2022.

[15] S. Rajendran, R. Calvo-Palomino, M. Fuchs, B. Van den Bergh, H. Cordobés, D. Giustiniano, S. Pollin, and V. Lenders, "Electrosense: Open and big spectrum data," *IEEE Communications Magazine*, vol. 56, no. 1, pp. 210–217, 2017.