Reinforcement Learning for Anti-Ransomware Testing

Alexander Adamov
NioGuard Security Lab /
Design Automation Dep.
Kharkiv National University of Radio Electronics
Kharkiv, Ukraine /
Dep. of Software Engineering
Blekinge Institute of Technology
Karlskrona, Sweden
ada@nioguard.com

Anders Carlsson
Dep. of Computer Science
Blekinge Institute of Technology
Karlskrona, Sweden
anders.carlsson@bth.se

***Abstract*—In this paper, we are going to verify the possibility to create a ransomware simulation that will use an arbitrary combination of known tactics and techniques to bypass an anti-malware defense.**

To verify this hypothesis, we conducted an experiment in which an agent was trained with the help of reinforcement learning to run the ransomware simulator in a way that can bypass anti-ransomware solution and encrypt the target files.

The novelty of the proposed method lies in applying reinforcement learning to anti-ransomware testing that may help to identify weaknesses in the anti-ransomware defense and fix them before a real attack happens.

Keywords—ransomware, machine learning, reinforcement learning, artificial intelligence, anti-ransomware testing

I. INTRODUCTION

68% of ransomware attacks go unnoticed according to the latest report by US cybersecurity provider FireEye [1] that draws the cybersecurity experts' attention to this problem.

In the previous work, we already analyzed the LockerGoga ransomware used in the targeted attack against Norsk Hydro in consequence of which the company needed to switch to manual operation mode reducing the production capacity. The discovered techniques included digital signing of ransomware executables and multi process encryption when a single worker process was created and responsible for encryption only one user's file. These techniques helped the ransomware go under the radar. [2]

Another ransomware called Maze discovered in May 2019 and used in the recent attack against Canon on July 30, 2020 that caused the outage of the image.canon cloud service. Before that, the Maze operators published the data stolen from Xerox and LG companies in June 2020.

The author(s) of Maze ransomware complained that it is so easy nowadays to bypass an antivirus protection because they "place a signature on data section in the packer layer" that makes EDR (Endpoint Detection and Response) [3] solutions useless when it comes to detecting targeted ransomware attacks. Because, once repacked, a piece of malware becomes undetectable again.

One more recent ransomware called WastedLocker and operated by the Evil Corp gang has shown the power of bypassing anti-ransomware modules that typically rely detecting anomalous behavior. The ransomware employed *Alternate Data Streams (ADS)* in NTFS to drop the payload and memory-mapped files for encrypting user's data in addition to digital signing employed by LockerGoga and MegaCortex in the beginning of 2019. As a result, WasteLocker managed to encrypt data stored on the Garmin's servers. [4]

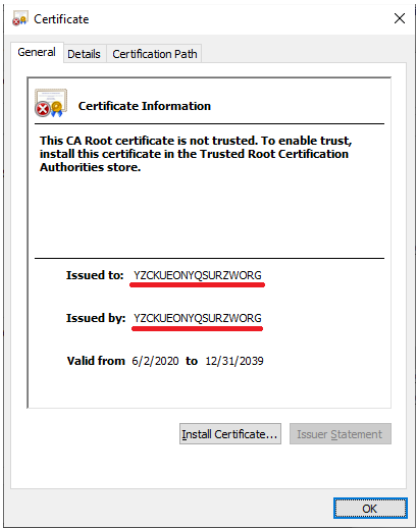

Fig. 1. WastedLocker uses a self-signed digital certificate.

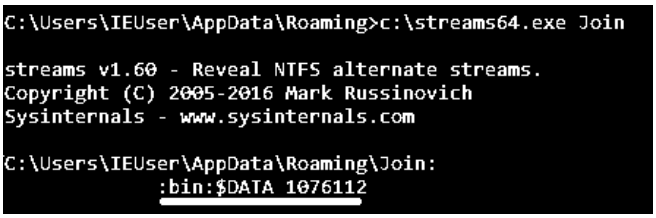

Fig. 2. WastedLocker stores its copy in the ADS 'Join:bin'.

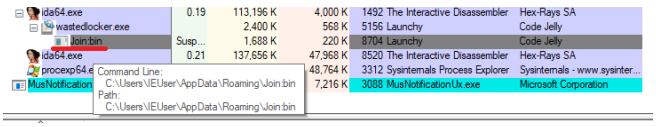

Fig. 3. WastedLocker executes its code from the ADS 'Join:bin'.

WastedLocker also took advantage of the Lazy Writer function of the Windows Cache Manager. This function is used by the Windows Cache Manager to reduce the overall number of disk I/O operations. The file's data is stored in cache pages in the memory and then written to disk by the Cache Manager allowing to accumulate file changes in the memory and, only then, flush them all at once. [5]

This is enabled with memory-mapped files. In a regular way, we need first to open a file for modification and get its size to know how many memory pages to allocate. Then, we call CreateFileMapping() and MapViewOfFile() to load the file's content to the memory. After, modifying data in the memory, a program usually calls UnmapViewOfFile(), closes a handle of the memory-mapped file and, only then, a file handle.

To force Lazy Writing on behalf of the System process, WastedLocker closes a file handle right after mapping the file

978-1-7281-9899-6/20/$31.00 ©2020 IEEE

but before encrypting the data to let the Cache Manager write back the encrypted data later on behalf of the operating system.

TABLE I
THE FILE MEMORY MAPPING EXECUTION

#	Regular file memory mapping	Wastedlocker's way
1	CreateFile() - open file	CreateFile() - open file
2	GetFileSize()	GetFileSize()
3	CreateFileMapping()	CreateFileMapping()
4	MapViewOfFile()	MapViewOfFile()
5	Modify mapped data	**CloseHandle(file)**
6	UnmapViewOfFile()	**Encrypt mapped data**
7	CloseHandle(file map)	UnmapViewOfFile()
8	CloseHandle(file)	CloseHandle(file map)

As a result, the encrypted data is flushed by the System process after the ransomware process has already closed the file handle.

Fig. 4. System process writes back the cached encrypted data to the user's file after the file handle has been already closed by the ransomware.

As we can see, the described techniques have been evolving and may help attackers to elude anti-ransomware protection. Therefore, it is essential to act proactively and test anti-malware solutions (EDR) if they can provide an adequate response to the modern ransomware defense evasion techniques. Attack simulation based on the discovered techniques can be

In 2017 [6] we already made the first attempt to evaluate the quality of anti-malware solutions by conducting the Anti-Ransomware Test using the Ransomware Simulator with the limited set of basic ransomware techniques to simulate behavior of popular ransomware families. The test results revealed that just a few products had managed to detect a simulation of the known ransomware attacks.

In this paper, we explain how we applied Reinforcement Learning (RL) approach to anti-ransomware testing. We created a simulation environment that includes two major components: Ransomware Simulator (an attacker) and Ransomware Detector (a defender). We also introduced an Agent that can learn with the help of RL how to perform the ransomware attack in an optimal way bypassing Ransomware Detector.

II. RANSOMWARE ATTACK SIMULATION

To be able to test if antiviruses (EDR solutions) can detect an unknown ransomware attack, we propose the attack simulation. An example of such approach is MITRE ATT&CK Evaluation project [7]. The first evaluation of EDR solutions was performed based on the discovered attacks by APT29 group attributed to Russian Intelligence Service. The attack simulation included tactics and techniques of this hacking group [8].

Similarly, we designed a ransomware simulation tool (Ransomware Simulator) to imitate techniques employed by a ransomware. For this experiment, only three parameters were chosen:

1) adding extension to an encrypted file, e.g. '.enc';

2) encoding the encrypted data with Base64 that helps to reduce an entropy level;

3) the number of files to be encrypted per step.

The simulator uses AES-256 for encryption and targets documents, multimedia files, and archives that are typically encrypted by ransomware. The goal of the Ransomware Simulator is to encrypt the maximum number of files in the minimal number of steps on the target system.

TABLE II
THE CHOSEN PARAMETERS OF THE RANSOMWARE SIMULATOR

Parameter	Value 0	Value 1	Value 2	Value 3
Adding the extension	no	yes		
Base64 encoding	no	yes		
The number of encrypted files per action	1	2	5	10

III. RANSOMWARE DEFENSE SIMULATION

To counteract the Ransomware Simulator, we created Ransomware Detector. The detector implements three methods to detect the ransomware activity in correspondence with the Ransomware Simulator's parameters:

1) checking if the second extension exists;

2) entropy level evaluation;

3) detection of anomalous modification time of the files (e.g. 8 files have been modified within 1 second).

The detection *Threshold* was set to 8, which means if the Ransomware Detector sees 8 files with one of the following anomalies: 1) second extension; 2) high level of entropy that indicates that data are encrypted; 3) similar modification time then it triggers an alert 'Ransomware Detected' and block the attack. The Ransomware Simulator failed and the game (attack) is over (blocked).

IV. ENVIRONMENT

The recent targeted ransomware attacks such as Maze, WastedLocker, Netwalker, Clop, and others target Windows OS. Therefore, we used a Windows 10 virtual machine [9] as a simulation environment where we placed the folder with ten files that include documents, multimedia files, archives that are typically encrypted by ransomware.

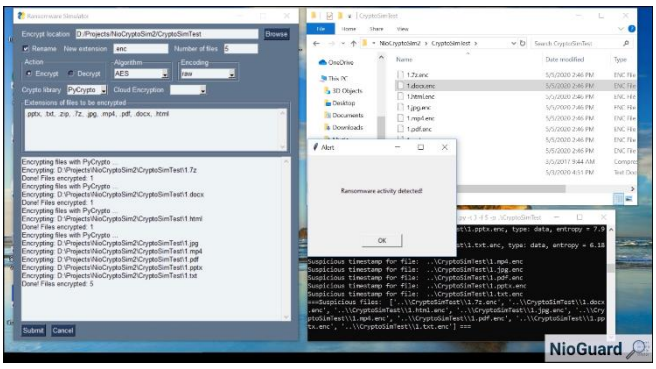

Fig. 5. A screenshot of the simulation environment showing the Ransomware Simulator (on the left), target user's files and the Ransomware Detector (on the right).

V. Reinforcement Learning

Ransomware Simulator is a tool that requires a set of parameters as an input to operate. The problem is how to find these parameters that will allow the Ransomware Simulator to bypass the Ransomware Detector. To address this problem, we came to the idea of adding Artificial Intelligence (AI) to the Ransomware Simulator with the help of RL that should be well known to the players of the Real-Time Strategy (RTS) games, such as StarCraft.

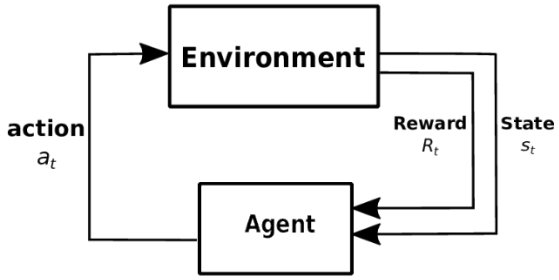

Fig. 6. Reinforcement learning process.

The key advantage of RL is that it does not require training data or specific expertise in the domain. It needs only a goal to be specified and the Agent finds the optimal way (a policy) to achieve that goal using the trial and error method.

In RL, we have the Agent and Environment. The Agent performs actions that affect in some way the Environment and receives the new state of the Environment and the Reward (a numerical score) that evaluates how good the previous action was in terms of leading the Agent to maximization of the total reward in the long run.

In our case the Environment can be a user's Windows OS with antivirus (the Ransomware Detector). The Actions performed by the Agent are ransomware attempts to encrypt user's files. After executing an Action, the Agent receives information about the new state of the Environment and how successful the attempt was. The state of the Environment is evaluated based on the number of encrypted files [10].

Q-learning algorithm is commonly used to solve RL tasks. Q-learning is a reinforcement learning algorithm defined over Finite Markov Decision Process (FMDP) which does not require creating a model of the Environment. The algorithm calculates the quality (Q) of a combination of a state (S) and action (A) based on a reward value (R):

$$Q: S \times A \rightarrow R \qquad (1)$$

The Q values are updated based on the reward the Agent got and the reward the Agent expects next.

$$Q(s_t, a_t) \coloneqq Q(s_t, a_t) + \\ + \alpha \cdot [r_{t+1} + \lambda \cdot max_a Q(s_t, a) - Q(s_t, a_t)] \qquad (2)$$

, where

α – learning rate;

r – Reward;

λ – discount – how the future Action value is weighted over the one at present;

$max_a Q(s_t, a)$ – the estimated Reward from the next Action.

The algorithm requires the definition of the possible states, actions, and reward function.

Set of States. There are 11 states in our model that represents the number of encrypted files in the target folder from 0 to 10.

Set of Actions. There are 16 possible actions that are the combinations of three Ransomware Simulator's parameters.

TABLE III
The Actions encoding table

Action code	Extension (code)	Base64 (code)	Number of files (code)
0	0	0	0
1	0	0	1
2	0	0	2
3	0	0	3
4	0	1	0
5	0	1	1
6	0	1	2
7	0	1	3
8	1	0	0
9	1	0	1
10	1	0	2
11	1	0	3
12	1	1	0
13	1	1	1
14	1	1	2
15	1	1	3

See Table 1 to translate the codes to the actual values of the parameters.

Reward. The Reward is calculated using the following formula:

$$\text{Rewards} = N_{enc} * 2 - 1, \qquad (2)$$

where

N_{enc} – is the number of encrypted files as the result of an action.

In other words, for one encrypted file the Agent earns 2 points, and one action costs 1 point.

Learning strategy. The algorithm starts with the random (*exploration*) policy and then slowly reduce the probability of random choice for an action from 1 in the beginning to 0 at the end of the learning process (*exploitation*). Discount = 0.95. Learning rate = 0.1.

VI. Results

After 2000 iterations (595 games have been played), we obtained the following results.

As shown on Figure 7, the Reward values goes up through the training session starting from 0 and ending in 15 – the highest possible reward value to achieve out of 20 with the given Ransomware Detector configuration. The Agent gets 20 points for 10 encrypted files and spends 5 points to execute the ransomware simulator five times changing the options in order to bypass the Ransomware Detector.

The same trend can be seen on Figures 8 and 9 where the number of wins (Ransomware Simulator encrypts all the files without being detected) grows exponentially with the number of played games.

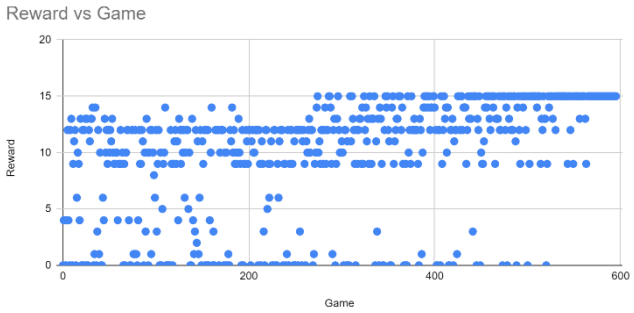

Fig. 7. Learning progress (Reward vs. Game rate).

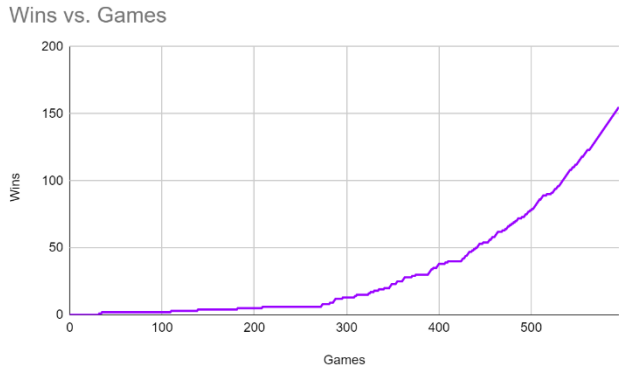

Fig. 8. Learning curve (Wins vs. Games rate).

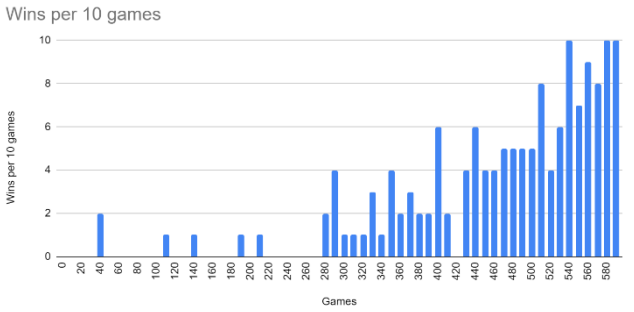

Fig. 9. The number of Wins per 10 games.

Q (S, A)	States								Threshold		
Actions	0	1	2	3	4	5	6	7	8	9	10
0	3.742632	0.787156	0.50037	0	0	1.007811	0.119928	2.686471	1.771443	0	
1	7.105196	1.013578	1.252902	0.625865	0	5.491767	0	0.110781	0.020577	0	
2	7.545032	3.370516	1.118979	0.296561	0	1.170941	0.170201	0.147976	0.215866	0.763711	
3	6.681859	1.036887	1.431128	0	0	1.178761	0.10031	0.526232	0.019609	0.1805	
4	3.948145	0	0	0.231532	0.120758	0.790898	0.599471	0.871498	0.483216	0	
5	5.816442	0	1.158697	0.693802	0	1.778738	0.715084	0.205816	0.029846	0	
6	3.734055	8.923577	2.616297	0	0	0.747448	0.252802	0.236518	0.020577	0	
7	7.022468	0.056858	1.931174	0.563466	0	1.003211	0.213173	0.198259	0.377146	0	
8	4.158711	0.963861	1.045184	0	0	0.459586	0.590549	0.028423	0.28698	0.01805	
9	5.872631	0.463353	0.387103	0	1.524341	1.143632	0.358465	0.771859	0.112953	0.119148	
10	5.959267	0.9285	8.530923	1.739734	0	1.55761	0.141551	0.244138	0.172354	0	
11	6.141237	0	0.768712	0	0.144812	1.640483	0.149433	0.418004	0.055891	0.068169	
12	5.179697	1.992647	0.713766	0	0.443053	0.652553	0.226234	0.173608	0.372956	0.089291	
13	3.181544	1.945088	0.629294	0	0.422005	1.051774	2.283579	0.231025	0.166056	0	
14	14.11996	2.516235	1.889826	3.106087	0.087103	0.705648	0.275448	0.247926	0.20654	0	
15	5.828471	0.885469	0	0.163353	0	1.260362	0.080788	0.186429	0.010403	0.07444	

All the files are encrypted

Fig. 10. Q-matrix with the optimal policy highlighted.

VII. Analysis of Results

After training, the Agent found the *optimal policy* that allowed it to encrypt all ten files in the target folder:

1. *State 0: Action 14* - encrypt 5 files adding the extension and apply Base64 encoding to reduce entropy.
2. *State 5: Action 1* – encrypt 2 files without adding the extension and Base64 encoding.
3. *State 7*: *Action 0* – encrypt 1 file without adding the extension and Base64 encoding.
4. *State 8*: *Action 0* – encrypt 1 file without adding the extension and Base64 encoding.
5. *State 9*: *Action 2* – encrypt 5 files without adding the extension and Base64 encoding.

At State 9, any Action would lead to win because only one file left unencrypted. Moreover, at State 9, we have:

- 5 (or 7 – two consequent iterations may result in the encrypted files having similar modification time within Δt) files modified at the same time (within Δt = 1 sec),
- 5 files with the extension,
- 4 files with high entropy.

None of these exceeds the detection threshold equal to 8.

VIII. Conclusions

The obtained results during the experiments show that RL can help to discover an attack strategy that can overcome a behavior-based anti-ransomware protection. It is worth noting, that the experiment was conducted on the Ransomware Detector that only represents the limited number of detection methods imitating the behavior of a real EDR solution. However, the presented results look promising and the proposed RL approach for anti-ransomware testing can be further applied to the anti-malware products with anti-ransomware modules available on the market that mostly rely on behavior analysis.

The future work can be also conducted on applying the RL approach to network penetration testing. So, the Agent can learn how to discover the optimal attack path that can be used by an ethical hacker in security testing of a service or product.

References

[1] Security effectiveness Report 2020. Deep dive into cyber reality, Mandiant, 2020.

[2] A. Adamov, A. Carlsson and T. Surmacz. An Analysis of LockerGoga Ransomware, 2019 IEEE East-West Design & Test Symposium (EWDTS), Batumi, Georgia, 2019, pp. 1-5, doi: 10.1109/EWDTS.2019.8884472.

[3] A. Chuvakin. Named: Endpoint Threat Detection & Response, Gatner, 2013, available at https://blogs.gartner.com/anton-chuvakin/2013/07/26/named-endpoint-threat-detection-response/

[4] WastedLocker's techniques point to a familiar heritage, Sophos, 2020, available at https://news.sophos.com/en-us/2020/08/04/wastedlocker-techniques-point-to-a-familiar-heritage/

[5] Russinovich M. E., Solomon D. A., A. Ionescu. Windows Internals, Part 2 (6th ed.), Pearson Education; September 2012, ISBN: 9780735677289.

[6] Ransomware Protection Test, April 2017, NioGuard Security Lab, available at https://www.nioguard.com/2017/05/ransomware-protection-test-april-2017.html.

[7] Methodology Overview: Adversary Emulation. MITRE, 2020, available at https://attackevals.mitre.org/adversary-emulation.html

[8] APT29 Emulation. MITRE, 2020, available at https://attackevals.mitre.org/APT29/

[9] Windows virtual machines, 2020, available at https://developer.microsoft.com/en-us/microsoft-edge/tools/vms/

[10] F. Bach, R. S. Sutton, A. G. Barto. Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning series). Second edition, A Bradford Book, 2018.