QRASSH - A self-adaptive SSH Honeypot driven by Q-Learning

Conference Paper · June 2018

DOI: 10.1109/ICComm.2018.8484261

3 authors, including:

Adrian Pauna
Technical Military Academy of Bucharest
7 PUBLICATIONS **107** CITATIONS
SEE PROFILE

Andrei-Constantin Iacob
Universitatea Alexandru Ioan Cuza
2 PUBLICATIONS **35** CITATIONS
SEE PROFILE

QRASSH – A self-adaptive SSH Honeypot driven by Q-Learning

Adrian Pauna[1] , Andrei-Constantin Iacob[2], and Ion Bica [3]
[1] Faculty of Military Electronic and Information Systems, Military Technical Academy, Bucharest, Romania
[2] Faculty of Computer Science, A. I. Cuza University of IAŞI, Iasi, Romania
[3] Faculty of Military Electronic and Information Systems, Military Technical Academy, Bucharest, Romania
Contact author e-mail: adrian.pauna.ro@gmail.com

Abstract— **Developed for the first time in the 80s, honeypot systems research increased tremendously in the last decade. Moving from simple monitored, emulated, Internet Services, towards intelligent systems that autonomously interact with attackers, was a normal engagement in the context of higher development of artificial intelligence as science. In this paper we present a newly developed SSH self-adaptive honeypot system that uses a Deep Q-Learning algorithm in order to decide how to interact with external attackers. The honeypot system is developed in Python and integrates an existing implementation of a Reinforcement Learning algorithm that makes use of neural network (NN).**

Keywords—honeypot systems; neural network; reinforcement learning;information security

I. INTRODUCTION

The increased number of cybersecurity attacks and its ever-growing level of complexity makes the need for an in depth study of them, a real demand. Studies reveal a major switch in the direction of what is called targeted attacks [1]. Compared with traditional malware that spreads independently of the target, these attacks, also known as APT (Advanced Persistent Threats) [2], comprise specific generated exploit vectors and in some cases they necessitate direct human interaction with the scoped systems. In these specific cases, collecting useful information that can be used as Threat intelligence or Indicators of compromises (IOC) [3] becomes a complicated objective.

The system we have developed tries to add a new perspective on the above problem, by providing a solution that is able to log all the actions an attacker does, and forcing him, in an autonomous manner, to download as much as possible malicious software. This way attacks such as APTs can be more thoroughly identified and collected data can be derived in specific IOCs. The idea of having an SSH honeypot capable of interacting with the attacker was initially developed by Wagener [4]. The author has used a modified Linux kernel that was able to block the execution of console commands and take other actions too. Starting from this point, Pauna et al. developed RASSH, a self-adaptive honeypot system that was using SARSA algorithm in order to decide on the action taken [5].

From the perspective of the reinforcement learning algorithm used, the solution proved to be complex as it required a big number of computing resources. This took us to the point of developing a new and improved system that uses the neural networks features used in the Deep Q-Learning algorithm implementation.

The first contribution of this article is represented by the fully-fledged Honeypot system developed that offers the possibility of integration of different Reinforcement Learning algorithms. The honeypot system offers the possibility to visualize all the commands and actions taken in an interactive way.

Secondly, the system models the integration of a Deep Q-Learning algorithm [6]. We have abstracted the general components of the algorithm (Agent, Environment, States, Reward function, Q-values) in order to match the real situation of our SSH Honeypot.

Finally, yet importantly, we have incurred a set of experiments and tests that confirm that our system learns the best policy that will allow reaching the goal of having decoyed the attackers to download as much malicious software as possible.

In **Sect. 2** we make a short debrief of the general aspects as regards honeypot systems and their self-adaptive nature. We present also a short overview of related work on the subject. In **Sect. 3** we present the Deep Q-learning algorithm we have used and insist on its features that recommended it. In **Sect. 4** we present in a concise manner the implementation of QRASSH, marking its features and capabilities. We continue by presenting in **Sect. 5** the experiments and results we have obtained confirming that the developed system fulfils the tasks has been designed for. We close our presentation, in **Sect.6**, with a set of conclusions and remarks as regards future developments that are foreseen.

II. HONEYPOT SYSTEMS

Initially developed in the 80s, as a solution to monitor the activity of malicious intruders, the Honeypot systems became, nowadays, a valuable resource of gained knowledge as regards the cyber-attacks [7].

In its purest form, a honeypot system is an information system resource (Internet services or even fully active computers) exposed over Internet with the main purpose of being attacked, exploited and therefore logging all information about the interactions with the perpetrators.

A. Short classification

From its inception, honeypot systems were developed with less interaction capabilities and therefore this was called Gen I and named low interaction honeypot [8]. In most of the cases we were talking about common Internet services such as SSH, SFTP, NetBIOS that were emulated and offered to be attacked over Internet. A good example of such system is honeyd [9].

As the needs evolved and the complexity of attacks burst, honeypot systems evolved to what was named Gen II, high interaction systems. These ones were, in most of the cases, fully operational systems (Linux, Microsoft), exposed over Internet that were monitored from "behind the scenes" without the attacker noticing [8].

As Spitzner [10], a pioneer in the research of honeypot systems, stated, the obvious road of honeypot systems was reaching a greater level of interaction with the attackers and the surrounding environment by implementing dynamic features. In his view, a honeypot system should be capable of self-configuring based on the surrounding environment where it was placed, therefore capable to blend-in. These newly developed systems are called dynamic honeypots.

In the latter years a new development occurred, which involved using machine learning algorithms that offer possibility to directly interact with attackers. These systems were classified as self-adaptive. Pauna et.al [5] proposed a honeypot system that emulates a SSH service and interacts directly with the commands inserted by the attackers blocking their execution, or delaying it or even substituting the legitimate output with a fake one.

B. Self-adaptive Honeypot

Wagener et al. [11], proposed a self-adaptive honeypot system using game theory as basis for the decision of actions to be taken while interacting with the attackers. The system used a Linux platform exposing a SSH services and the interaction was possible by using a modified kernel module (LKM). A further development was the used or machine learning SARSA algorithm and the KIPPO SSH platform. Pauna et al. implemented as a standalone project, RASSH, a honeypot system that used the code of KIPPO and updated it with two extensions: an Actions module and a Reinforcement Learning one [5].

C. Related work

Gutierrez et al. [12] present a research on the strategic selection of various honeypot configurations in order to adapt to an intelligent attacker. They use a model mapped to a Multi-Armed Bandit (MAB) problem.

Mehta et al. [13] present an architecture of frameworks that can be used on collected honeypot systems data in order to make predictions. These predictions are the result of using various machine learning algorithms and produced as signatures that can be used by specific detection tools.

Fraunholz et al. [14] tackle in their article the method of identification of the machines and devices in a network using clustering. This way they decide in an intelligent manner how to deploy honeypot systems.

Wang et al. [15] makes an analysis of interactions between attackers and defenders in a Smart Grid environment. They derive optimal strategies for both sides and prove the existence of Bayesian Nash equilibrium in honeypot game.

D. La et al. [16] uses the same approach of looking at the model of deception involving an attacker and a defender from a game-theoretic perspective. But, instead of Smart Grid environment he uses IoT as area of study.

Hayatle et al. [17] presents a solution in order to have high interaction honeypots part of botnet network for a longer period. The author presents a Bayesian Game theoretic framework that models the interaction between honeypots and botmasters.

As we review the existing literature, we understand that not much work was done for development of honeypot systems that leverage reinforcement learning algorithms.

As most of the papers are focused on how to model the interaction between attackers and defenders using game-theoretic approach in different context.

For the moment the work of Wagener [4], [11] and Pauna [5] remains the only one that presents implemented systems that do apply the honeypot and machine learning concepts.

III. Deep Q-Learning

As part of Machine Learning domain, Reinforcement Learning (RL) is inspired by behavioral psychology. At its essence the theory behind is concerned with how a software agent takes actions in an environment in order to maximize a long-term reward.

In this context, Reinforcement Learning also uses an environment typically formulated as a Markov Decision Process (MDP). MDP is a discrete time stochastic control process [17], which assumes that at every time step we have some state ***s***, and the decision maker can take any action ***a*** that is available in state ***s.***

As depicted in Fig.1, at each step t the agent input is represented by the current state S_t and it takes action A_t . Than it receives a reward R_{t+1} and the next state S_{t+1}. The decision on what action to take is based on some policy $\pi: A_t = \pi(S_t)$. In the end the goal is to find a policy that maximizes the long-run cumulative reward $\sum R_t$

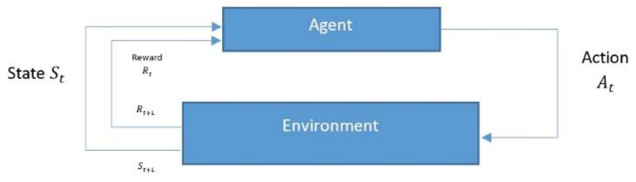


Fig. 1. Reinforcement Learning

Compared to supervised learning methods, RL assumes the target outputs are not known and the evaluation of the performance of an agent is quantified in terms of a reinforcement signal [18]. This means that throughout the entire cycle, the focus is not on finding correct input/output pairs but instead the focus is in finding a balance between exploration and exploitation.

To better understand the exploration-exploitation trade-off let's imagine a child that learns to walk. At the begging he makes random moves, which is exploration, but after discovering that specific ones that allow him to advance in the desired direction, he chooses those more often and that's exploitation.

A. Q-Learning

As part of Reinforcement Learning methods, Q-learning is a temporal difference (TD) algorithm [19], meaning that the time differences between actions taken and rewards received are involved. Its main objective is to calculate state-action values, also called Q-values based on the following formula (1):

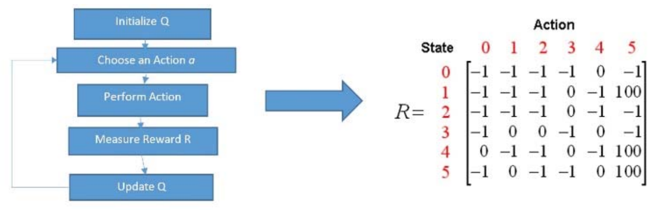
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_t + \gamma maxQ(S_{t+1}, A_{t+1}) - Q(S_t, A_t)] \quad (1)$$

Where:

- α is a parameter 0→1 called learning rate and determines the speed with which learning can occur;
- γ is a parameter 0→1 called discount factor and it determines the way future rewards are used in updating Q-values.

B. Deep Q-Learning

A simple description of Q-learning goes like this:



$$R = \begin{array}{c|cccccc} & 0 & 1 & 2 & 3 & 4 & 5 \\ \hline 0 & -1 & -1 & -1 & -1 & 0 & -1 \\ 1 & -1 & -1 & -1 & 0 & -1 & 100 \\ 2 & -1 & -1 & -1 & 0 & -1 & -1 \\ 3 & -1 & 0 & 0 & -1 & 0 & -1 \\ 4 & 0 & -1 & -1 & 0 & -1 & 100 \\ 5 & -1 & 0 & -1 & -1 & 0 & 100 \end{array}$$

Fig. 2. Q-Learning process

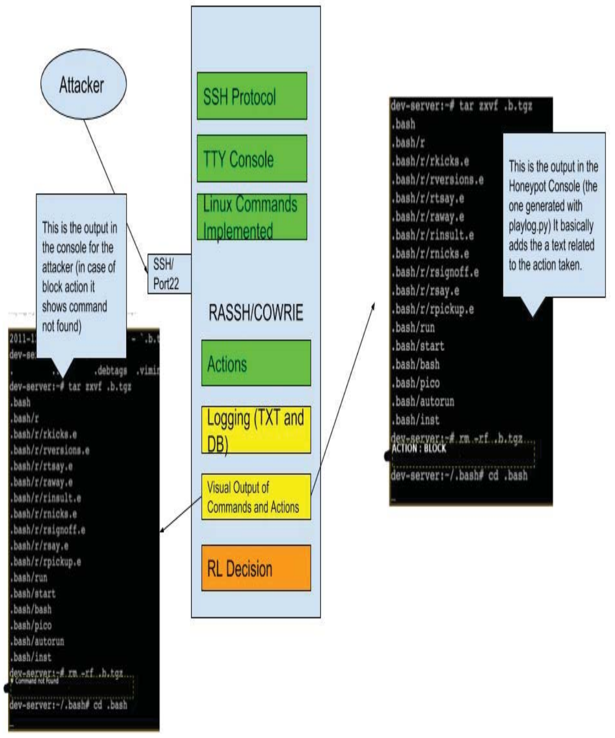
The implementation assumes using a lookup-table that stores all the Q-values for each (state, action) pair. An article published in 2015 [20], by Google DeepMind, explains how an agent can learn to play games by just observing the screen. At its core stands the Deep Q Network (DQN) introduced in 2013 [21]. In a simplistic approach using neural networks as a type of function approximation for the Q(s,a) function is the solution presented in these articles.

Instead of using a neural network architecture that has as input states and actions and as output the value of a single pair, DeepMind network accepts states and outputs separate Q-values for each possible action. This approach is used also for our honeypot as we need to get **max(Q(s,a))**[maximum of Q-values for every possible action in the next state **s**]

IV. QRASSH Implementation

A. The Adaptive honeypot

Our solution is based on an existing medium interaction Honeypot system called ***Cowrie*** [22]. Cowrie is a branch of a former project named Kippo that was not updated in the last years. The honeypot system uses twistd library in order to emulate a fully-fledged SSH server [23]. It also provides a full set of Linux commands emulated in order to give the outside attackers the possibility to pursue their goals. In principle the main focus of the system is to collect all the information that an attack would produce, such as: credentials used in order to brute force the system access, continuing with logging all the commands and attacker would insert in the console and ending with storing all the data an attacker would download on the system.


Attacker
This is the output in the console for the attacker (in case of block action it shows command not found)
SSH/ Port22
SSH Protocol
TTY Console
Linux Commands Implemented
RASSH/COWRIE
Actions
Logging (TXT and DB)
Visual Output of Commands and Actions
RL Decision
This is the output in the Honeypot Console (the one generated with playlog.py) It basically adds the a text related to the action taken.

```
2011-1                       - `.b.t
dev-se
.         ..          .debtags  .vimi
dev-server:~# tar zxvf .b.tgz
.bash
.bash/r
.bash/r/rkicks.e
.bash/r/rversions.e
.bash/r/rtsay.e
.bash/r/raway.e
.bash/r/rinsult.e
.bash/r/rnicks.e
.bash/r/rsignoff.e
.bash/r/rsay.e
.bash/r/rpickup.e
.bash/run
.bash/start
.bash/bash
.bash/pico
.bash/autorun
.bash/inst
dev-server:~# rm -rf .b.tgz
# Command not found
dev-server:~/.bash# cd .bash
```

```
dev-server:~# tar zxvf .b.tgz
.bash
.bash/r
.bash/r/rkicks.e
.bash/r/rversions.e
.bash/r/rtsay.e
.bash/r/raway.e
.bash/r/rinsult.e
.bash/r/rnicks.e
.bash/r/rsignoff.e
.bash/r/rsay.e
.bash/r/rpickup.e
.bash/run
.bash/start
.bash/bash
.bash/pico
.bash/autorun
.bash/inst
dev-server:~# rm -rf .b.tgz
ACTION : BLOCK
dev-server:~/.bash# cd .bash
```

Fig. 3. QRASSH block schema

We have modified cowrie and added an Actions Module that makes it possible to interact inside the console directly with the attackers. The actions implemented are similar to the ones developed by Pauna in RASSH system [5] and they contain:

- ***allow***: it allows the execution of a Linux command in console
- ***block***: it blocks the execution of a Linux command in console
- ***delay***: it delays the execution of a Linux command in console
- ***fake*** : it modifies the output of a Linux command in console
- ***insult***: it modifies the output of a Linux command in console by showing an insult message in a language corresponding to the source IP address

B. *The DQN Module*

For the development of the DQN module, we have used as reference the implementation of Harvey [23]. This implementation has as scope providing means of reinforcement learning for a real car, so to be able to move around a room. It uses several existing libraries that offer machine learning features such as Keras with Theano backend [24].

As depicted above the Q-learning algorithm calculates a set of Q-values that can be disposed in a lookup-table. To improve its capabilities, DQN uses a neural network as an approximation function. As our research was not about deep learning or neural networks we have considered neural network as a black box algorithm with its only purpose being to approximately map inputs to outputs.

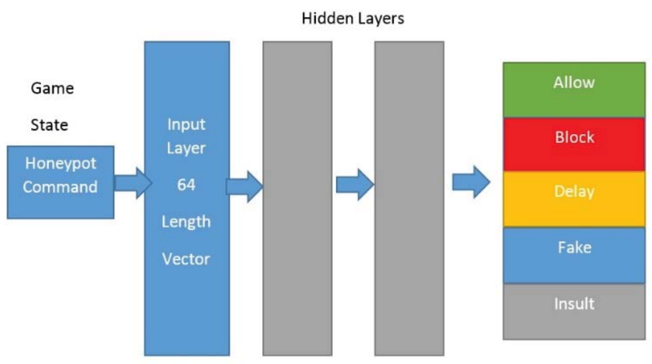


Fig. 4. Neural Network

We have used a neural network with an input layer, two hidden layers and an output layer similar to the one depicted in Fig. 4. With Keras it is easy to implement such a basic neural network as you can see in the code bellow:

```
# Sequential() creates the foundation of the layers.
     model = Sequential()
        # First layer.
        model.add(Embedding(100,64,
input_length=input_length))
        if input_length==1:
    # 'Dense' is the basic form of a neural network layer
            model.add(Dense(params[0],
init='lecun_uniform'))
            model.add(Activation('relu'))
        else:
            model.add(Bidirectional(LSTM(params[0])))
        model.add(Dropout(0.2))
        # Second layer.
        model.add(Dense(params[1], init='lecun_uniform'))
        model.add(Activation('relu'))
        model.add(Dropout(0.2))
        # Output layer.
        model.add(Dense(number_of_actions,
init='lecun_uniform'))
        model.add(Activation('linear'))
        optimizer = Adam()
    # Create the model based on the information above
        model.compile(loss='mse', optimizer=optimizer)
```

As per our implementation we consider the following [27]:

- Environment has 57 states comprised by the 56 Linux commands implemented plus the ***exit*** command
- Actions set is comprised by the 5 actions presented above
- Reward function R defined so to match our goal which is to have the attacker download malicious software as much as possible

and the workflow, as presented below:

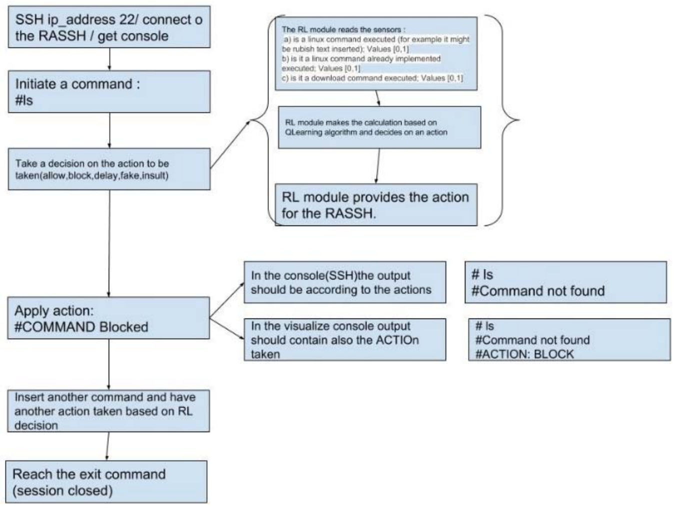

SSH ip_address 22/ connect o the RASSH / get console
Initiate a command : #ls
Take a decision on the action to be taken(allow,block,delay,fake,insult)
The RL module reads the sensors : a) is a linux command executed (for example it might be rubish text inserted); Values [0,1] b) is it a linux command already implemented executed; Values [0,1] c) is it a download command executed; Values [0,1]
RL module makes the calculation based on QLearning algorithm and decides on an action
RL module provides the action for the RASSH.
Apply action: #COMMAND Blocked
In the console(SSH)the output should be according to the actions
# ls #Command not found
In the visualize console output should contain also the ACTIOn taken
# ls #Command not found #ACTION: BLOCK
Insert another command and have another action taken based on RL decision
Reach the exit command (session closed)

Fig. 5. QRASSH workflow

V. Experiments and Results

Usually in games, the reward is directly related to the score of the game. In our case if we consider the situation in which the attacker introduces the sftp command in the console if we allow it the reward will be higher than if we delay it as the attacker will feel more confident on the execution of the sftp command. In order to logically represent this intuition and train it, we need to develop a formula that we can optimize on.

For this purpose, we have used the Taehoon Kim [26] mathematical representation of Q-learning and define our loss function as bellow:

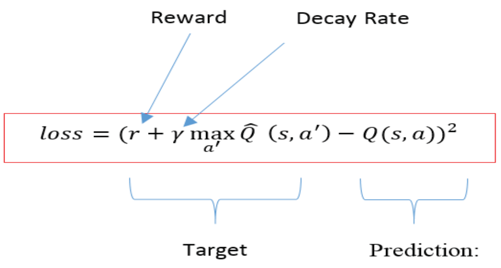


Fig. 6. Mathematical representation of Q-learning

In the formula, the goal is to decrease the gap between predictions and target (loss) [20].

In order to calculate the loss we:

1. Take an action ***a*** (allow, block, delay, fake, insult)
2. Calculate the reward ***r***
3. We observe the resulting new state ***s'***
4. We calculate the maximum target Q and discount it with Gamma (as future reward worth less than immediate)
5. We calculate target value by adding the current reward with the discounted future reward
6. We obtain loss by subtracting our current prediction from the target.
7. We square the obtained value so to punish large loss value and treat negative and positive values, the same

By using an exposed over Internet honeypot system and two, different implemented reward functions we have observed two situations. Due to a limited time, we have collected not more than 20 episodes in which attackers have got inside our honeypot system. For testing purposes we have simulated same behavior the attackers exposed, meaning the same sequence of commands introduced in the SSH console.

They are two different metrics to evaluate a model's performance usually being used in different phases: loss and accuracy.

The accuracy of a model is determined based on the fixed learned model parameters. Basically when learning has ended.

At this point test samples are fed to the model and the number of mistakes (zero-one loss) the model makes are recorded, after comparison to the true targets. Then is calculated the misclassification percentage which represents the accuracy.

Loss is often used in the training process to find the optimal parameter values for a model. It is what we try to optimize in the training by updating weights. At each training step the neural network measures the loss.

A better model means a lower loss. A model is trained and validated. The interpretation of loss is how well the model acts for these two sets and can be considered an accumulation of the errors made for each example in training and validation sets.

Loss is mostly used in the training process in order to optimize it by updating weights. In the graphs presented in Fig. 7 and Fig. 8 on x- axis we have the epoch and on the y-axis we have the loss value which unlike accuracy is not expressed in percentages.

A. *Simple Reward Function*

As our goal is to decoy the attackers to download as much as possible malicious code on our honeypot, we have prepared a simple reward function that offer:

- a value of **500** when a download command is executed;
- a **0** value for all the other commands and
- **-500** for when the exit command is being executed.

The results as presented in Fig.7 show that the neural network learns very quickly therefore the loss decreases in a few frames.

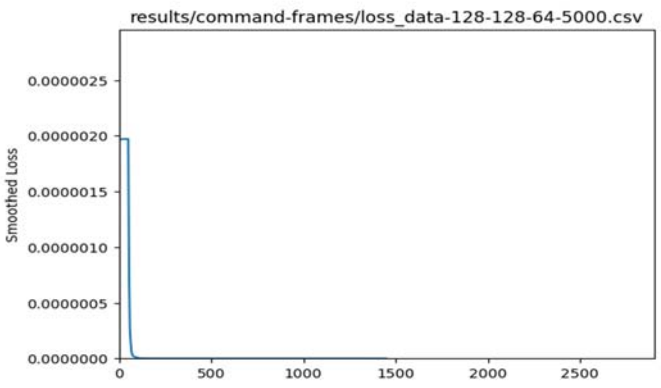


Fig. 7. Loss with simple reward function

B. *A more complex Reward Function*

In order to test our implementation we have developed a more complex reward function that recompenses as follows:

- **500** when a download command is executed
- **200** when a hacking command (nmap, metasploit, nc , etc..) is being tried
- **0** when a Linux command from the ones implemented but not a download one is executed
- **-200** when a Linux command not yet implemented is being tried
- **-500** when ***exit*** command is executed or session closed

With this new reward function, the results show that QRASSH takes much longer to learn as depicted in Fig. 8.

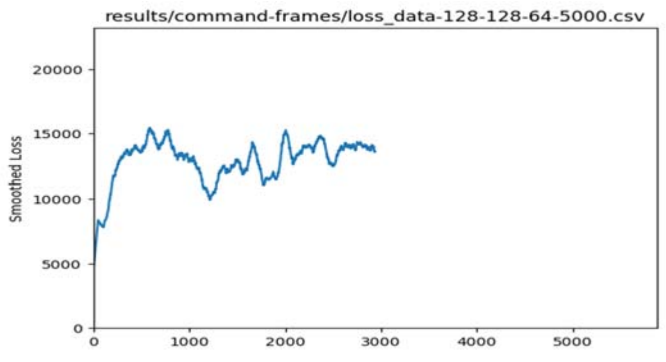


Fig. 8. Loss with a more complex reward function

VI. Conclusions and Future Work

In this paper we have presented the implementation of a SSH honeypot system that uses Deep Q-learning algorithm in order to interact with the attacker [27]. The goal of the implementation is to have a system capable to learn the behavior of the attacker so to be able to generate a policy π, defined as a series of (state, action) pairs that in the end maximize a long term reward. Based on our experiments results that the complexity of the defined reward function used makes it harder for the neural network to learn such a policy. Being given that the reward functions we have defined are purely subjective we consider relevant for future research the usage of Inverse Reinforcement learning algorithms in order to define better reward functions. Also approaches other than DQN for learning the Q matrix such as A3C [28] and Evolving Strategies [29] are promising areas for study.

Acknowledgment

This work was supported by a grant of the Romanian Ministry of Research and Innovation, CCCDI – UEFISCDI, project number PN-III-P1-1.2-PCCDI-2017-0272 / Avant-garde Technology Hub for Advanced Security (ATLAS), within PNCDI III.

References

[1] B. I. D. Messaoud, K. Guennoun, M. Wahbi and M. Sadik, "Advanced Persistent Threat: New analysis driven by life cycle phases and their challenges," 2016 International Conference on Advanced Communication Systems and Information Security (ACOSIS), Marrakesh, 2016, pp. 1-6.

[2] M. Ussath, D. Jaeger, Feng Cheng and C. Meinel, "Advanced persistent threats: Behind the scenes," 2016 Annual Conference on Information Science and Systems (CISS), Princeton, NJ, 2016, pp. 181-186.doi: 10.1109/CISS.2016.7460498.

[3] D. Rhoades, "Machine actionable indicators of compromise," 2014 International Carnahan Conference on Security Technology (ICCST), Rome, 2014, pp. 1-5.doi: 10.1109/CCST.2014.6987016

[4] G. Wagener, R. State, T. Engel and A. Dulaunoy, "Adaptive and self-configurable honeypots," 12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops, Dublin, 2011, pp. 345-352.doi: 10.1109/INM.2011.5990710.

[5] A. Pauna and I. Bica, "RASSH - Reinforced adaptive SSH honeypot," 2014 10th International Conference on Communications (COMM), Bucharest, 2014, pp. 1-6.doi: 10.1109/ICComm.2014.6866707.

[6] H. Li, T. Wei, A. Ren, Q. Zhu and Y. Wang, "Deep reinforcement learning: Framework, applications, and embedded implementations: Invited paper," 2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), Irvine, CA, 2017, pp. 847-854.doi: 10.1109/ICCAD.2017.8203866

[7] Marcus Gutierrez and Christopher Kiekintveld. 2017. Adapting with Honeypot Configurations to Detect Evolving Exploits. In Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems (AAMAS '17). International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 1565-1567

[8] Wira Zanoramy Ansiry Zakaria, Miss Laiha Mat Kiah "A review of dynamic and intelligent honeypots." ScienceAsia 39S (2013): 1–5

[9] W. Fan, Z. Du and D. Fernández, "Taxonomy of honeynet solutions," 2015 SAI Intelligent Systems Conference (IntelliSys), London, 2015, pp. 1002-1009.doi: 10.1109/IntelliSys.2015.7361266

[10] Lance Spitzner : https://www.symantec.com/connect/articles/dynamic-

[11] Wagener, Gerard Thesis: Self-Adaptive Honeypots Coercing and Assessing Attacker Behaviour http://hdl.handle.net/10993/15673

[12] V. Mehta, P. Bahadur, M. Kapoor, P. Singh and S. Rajpoot, "Threat prediction using honeypot and machine learning," 2015 International Conference on Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE), Noida, 2015, pp. 278-282.doi: 10.1109/ABLAZE.2015.7155011

[13] D. Fraunholz, M. Zimmermann and H. D. Schotten, "An adaptive honeypot configuration, deployment and maintenance strategy," 2017 19th International Conference on Advanced Communication Technology (ICACT), Bongpyeong, 2017, pp. 53-57.doi: 10.23919/ICACT.2017.7890056

[14] K. Wang, M. Du, S. Maharjan and Y. Sun, "Strategic Honeypot Game Model for Distributed Denial of Service Attacks in the Smart Grid," in IEEE Transactions on Smart Grid, vol. 8, no. 5, pp. 2474-2482, Sept. 2017.doi: 10.1109/TSG.2017.2670144

[15] Q. D. La, T. Q. S. Quek and J. Lee, "A game theoretic model for enabling honeypots in IoT networks," 2016 IEEE International Conference on Communications (ICC), Kuala Lumpur, 2016, pp. 1-6.doi: 10.1109/ICC.2016.7510833

[16] O. Hayatle, H. Otrok and A. Youssef, "A game theoretic investigation for high interaction honeypots," 2012 IEEE International Conference on Communications (ICC), Ottawa, ON, 2012, pp. 6662-6667.doi: 10.1109/ICC.2012.6364760

[17] Bellman, R. (1957). "A Markovian Decision Process". Journal of Mathematics and Mechanics. 6.A

[18] S. Nagendra, N. Podila, R. Ugarakhod and K. George, "Comparison of reinforcement learning algorithms applied to the cart-pole problem," 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Udupi, 2017, pp. 26-32.doi: 10.1109/ICACCI.2017.8125811A

[19] Human-level control through deep reinforcement learning ,V Mnih, K Kavukcuoglu, D Silver, AA Rusu, J Veness, MG Bellemare, ...Nature 518 (7540), 529-533

[20] Playing Atari with Deep Reinforcement Learning V Mnih, K Kavukcuoglu, D Silver, A Graves, I Antonoglou, D Wierstra, arXiv:1312.5602

[21] Cowrie : https://github.com/micheloosterhof/cowrie

[22] Twistd: https://twistedmatrix.com/documents/17.5.0/core/howto/

[23] Matt Harvey: https://github.com/harvitronix/reinforcement-learning-car

[24] https://keras.io/

[25] https://stackoverflow.com/questions/34518656/how-to-interpret-loss-and-accuracy-for-a-machine-learning-model

[26] Taehoon Kim: https://www.slideshare.net/carpedm20/ai-67616630

[27] QRASSH: http://github.com/qrassh

[28] V. Mnih, A. P.Badia, M. Mirza, A. Graves, T. Harley, T. P. Lillicrap, D.Silver, and K. Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48 (ICML'16), Maria Florina Balcan and Kilian Q. Weinberger (Eds.), Vol. 48. JMLR.org1928-1937

[29] Tim Salimans, Jonathan Ho, Xi Chen, Ilya Sutskever Evolution strategies as a scalable alternative to reinforcement learning,arXiv preprintarXiv:1703.03864,201