# The Bandit's States: Modeling State Selection for Stateful Network Fuzzing as Multi-armed Bandit Problem

1st Anne Borcherding
*Fraunhofer IOSB*
*Karlsruhe, Germany*
*anne.borcherding@iosb.fraunhofer.de*
*0000-0002-8144-2382*

2nd Mark Giraud
*Fraunhofer IOSB*
*Karlsruhe, Germany*
*mark.giraud@iosb.fraunhofer.de*
*0000-0002-2972-2758*

3rd Ian Fitzgerald
*Karlsruhe Institute of Technology*
*Karlsruhe, Germany*
*ian.fitzgerald@student.kit.edu*

4th Jürgen Beyerer
*Fraunhofer IOSB,*
*Karlsruhe Institute of Technology, and*
*KASTEL Security Research Labs,*
*Karlsruhe, Germany*
*juergen.beyerer@iosb.fraunhofer.de*

***Abstract*—Network interfaces of Industrial Control Systems are a common entry point for attackers, and thus need to be thoroughly tested for vulnerabilities. One way to perform such tests is with network fuzzers, which randomly mutate network packets to induce unexpected behavior and vulnerabilities. Highly stateful network protocols pose a particular challenge to fuzzers, since a fuzzer needs to be aware of the states in order to find deep vulnerabilities. Even if a fuzzer is aware of the states of a stateful network protocol, there are still several challenges to overcome. The challenge we focus on is deciding which state to test next. To make this decision, the fuzzer needs to strike a balance between exploiting known states and exploring states not yet tested. We propose to model this exploration versus exploitation dilemma using a Multi-armed Bandit. In this work, we present two modeling approaches and preliminary experiments. We choose to model the state selection problem with (I) a stochastic Multi-armed Bandit, and (II) an adversarial Multi-armed Bandit. The latter takes into account that coverage can only be discovered once, and that the underlying reward probability therefore decreases over time. Although the adversarial Multi-armed Bandit models the state selection problem more accurately, our experiments show that both approaches lead to statistically indistinguishable fuzzer performance. Furthermore, we show that the baseline fuzzer AFLNet leads to significantly better results in terms of coverage. Building on these unintuitive preliminary results, we aim to investigate the behavior of the agents in more detail, to include additional modeling approaches, and to use additional Systems under Test for the evaluation.**

## 1. Introduction

Securing Industrial Control Systems (ICSs) requires to consider security in all steps of the development lifecycle. This includes ensuring the robustness and resilience of the used components, such as Programmable Logic Controllers (PLCs) and Human Machine Interfaces (HMIs). One way to reveal weaknesses in robustness and resilience is to perform security testing. In particular, security tests should include tests against the network interfaces of the System under Test (SUT). On the one hand, this allows to test the SUT from the attacker's perspective in an automated way. On the other hand, it is one of the requirements formulated by IEC 62443, the standard for security in automation and control systems [12, part 4-1]. Amongst various testing techniques, this should include *fuzzing* of the network interfaces. Fuzzing is a testing technique in which random input is generated and sent to the SUT. The behaviour of the SUT in response to the input is analyzed to identify anomalies, potential bugs and vulnerabilities, or to guide the fuzzing process.

Most network protocols, however, are highly stateful. This is especially true for industrial network protocols, such as OPC UA. The stateful nature of network protocols presents a challenge for thorough testing and fuzzing, as vulnerabilities may be hidden in deep states of the SUT. One approach to this issue is *stateful* fuzzing, which uses state models to guide the fuzzing process (see e.g. [3], [7], [14]). Nevertheless, stateful fuzzers have additional challenges to overcome. One of these challenges is to decide which state to select for the next testing phase (see Section 2.1). This decision can be seen as an exploration versus exploitation problem since the probability of success per state is not known a priori. A success would be, for example, to find new coverage in the SUT, or to trigger a crash of the SUT.

**Approach.** An instance of the exploration versus exploitation problem is the Multi-armed Bandit (MaB) problem, where a non-contextual reinforcement learning agent repeatedly needs to choose between several actions. Each action results in a certain reward for the agent. The goal of the agent is to maximize their reward (see Section 2.2). Various algorithms for agents solving the MaB problem have already been proposed [19]. Thus, we propose to model the state selection problem as a MaB problem, and to leverage these existing algorithms.

**Contributions.** This work makes two main contributions: (I) We present a novel approach to model the state selection problem of stateful network fuzzers as an instance of the MaB problem, and (II) we conduct preliminary experiments analyzing whether this improves the fuzzing performance.

**Research Goals.** The overall goal of our work is to understand whether the state selection problem can be successfully modeled as a MaB problem and whether this improves the performance of a stateful fuzzer. We formulate the following two research questions:

RQ1 Does the modeling approach for the MaB problem affect the performance of stateful fuzzers based on a MaB state selection?

RQ2 How do stateful fuzzers using a MaB based state selection compare to the state of the art fuzzer AFLNet?

**Preliminary Results.** For our preliminary experiments, we implemented four graybox MaB fuzzers using two different modeling approaches and used the stateful network protocol OPC UA as fuzzing target (see Section 5). These experiments showed the following two main insights: On the one hand, the experiments suggest that the four MaB based fuzzers show no statistically significant performance increase over one another (RQ1). On the other hand, the baseline fuzzer AFLNet performs significantly better than all MaB fuzzers (RQ2). These preliminary results support results from literature [14], and build a starting point for further investigations on modeling the state selection problem as MaB problem. These further investigations could include, in particular, an analysis of the learning behavior of the agents, and the use of different modeling approaches and SUTs (see Section 6).

## 2. Background and Related Work

Our work combines two different domains: state selection in stateful network fuzzing and the MaB problem. The following provides an overview of those two domains.

### 2.1. State Selection in Stateful Network Fuzzing

Generally, fuzzing can be divided into blackbox, graybox, and whitebox fuzzing, depending on the information that is available to the fuzzer [15]. Our work is concerned with graybox fuzzers. Thus, we assume that the fuzzer has access to the information which code was covered by a certain test case (*code coverage*). The goal of stateful network fuzzing is to improve the fuzzing of highly stateful systems such as network protocols. To achieve this, state of the art fuzzers extract a state model and use this model to guide the fuzzing process [18]. Several approaches to represent and extract the state model have been proposed (e.g. [3], [17]). Next to other challenges that need to be solved for stateful fuzzing, the fuzzer needs to decide which state should be used for testing in the next fuzzing phase (*state selection*) [14]. Liu et al. show that current state selection algorithms do not increase performance in a significant manner [14]. In our work, we therefore investigate whether this is also the case if the state selection is modeled as MaB problem.

### 2.2. Multi-armed Bandit Problem

Figuratively speaking, in the MaB problem, an agent is presented with multiple slot machines (one-armed bandits) to choose from and has the goal of maximizing the total winnings over several rounds. To be more formal, in the MaB problem, a non-contextual agent is repeatedly presented with a choice between several actions. These actions lead to rewards which are drawn from a probability distribution that depends on the chosen action. The aim of the agent is to maximize the expected total reward [20]. As the agent has no prior knowledge of the probability distributions of the rewards, they must decide whether to exploit known actions or to explore unknown actions. Note that the agent is non-contextual, i.e. the environment in which the agent operates has no state. Thus, the agent only receives the information about the reward and no additional information about the environment, the fuzzer, or the SUT. Several approaches to formulate MaB problems have been proposed (see e.g. [2], [19], [20]). For our setting, we choose two different approaches: a *stochastic* MaB and an *adversarial* MaB. The main difference is that for a stochastic MaB, the reward is based on stationary probability distributions whereas for an adversarial MaB these distributions can change over time.

The adversarial approach is especially interesting for the modeling of the state selection in fuzzing. In general, the success in fuzzing is based on the new coverage or new bugs the fuzzing test discovers. Note that this explicitly refers to *new* coverage or bugs. Finding the same coverage or bugs several times will not lead to new insights and will consequently usually not lead to a reward for the agent. Thus, the probability distribution for the reward changes over time, and an adversarial MaB should be able to model the state selection problem more accurately.

For both modeling approaches, various algorithms for the agent's interaction with the MaB have been proposed. In our work, we choose the following common algorithms. For the stochastic MaB, we choose

(1) the $\epsilon$-Greedy algorithm (EPS), which chooses the action with the highest expected reward with a probability of $\epsilon$, and a random action with the probability $1 - \epsilon$ [20],

(2) the UCB1 algorithm (UCB), which is based on the *optimism under uncertainty* principle and chooses the best action based on optimistic estimates [20], and

(3) a tuned version of UCB1 (UCBT) which additionally takes the variance of an action into account [1].

For the adversarial MaB, we choose

(1) the Exp3 algorithm (EXP) presented by Auer et al. which makes no statistical assumptions about the distributions of rewards [2].

### 2.3. Related Work

The MaB problem or reinforcement learning in general has been successfully used to model various other parts of the fuzzing process (see e.g. [22] for an overview). For example, it has been used for seed selection [8], [23], [24], coverage metric selection [21], and mutation selection [4], [5]. All of these publications show an improvement of the fuzzing process by leveraging reinforcement learning for one or several decisions. However, to our knowledge, there is no publication trying to pass the state selection problem to a reinforcement learning agent.

## 3. Methodology

The goal of our work is to understand how the state selection can be modeled as a MaB problem and whether this modeling improves the overall fuzzer performance. Liu et al. showed that current approaches to state selection do not increase the overall performance [14]. However, approaches based on reinforcement learning and the MaB problem showed good performance in other decisions in fuzzing (see Section 2.3), and provide a good approach to the exploitation versus exploration problem. Based on these observations, we aim to investigate whether modeling the state selection using the MaB problem can improve the overall fuzzer performance. Especially, we aim to focus on different modeling approaches, including stochastic and adversarial approaches, and on how the modeling choices affect the fuzzer's performance.

### 3.1. Research Questions

We formulate two research questions. The first research question is concerned with the choice of the modeling approach for the MaB problem.

`RQ1` Does the modeling approach for the MaB problem affect the performance of stateful fuzzers based on a MaB state selection?

As presented in Section 2.2, one can choose from various approaches to model the MaB problem. For this work, we choose a stochastic approach and an adversarial approach. One would expect the agent in the adversarial setting to perform better since it takes the changing reward distributions into account. The rewards distributions change over time since coverage that has been found earlier in the fuzzing process will not be rewarded again. In addition to the general modeling approach, the reward and the hyperparameters of the agents need to be modeled (see Section 4). To evaluate the performance of the various configurations for the agents, we measure the code coverage a fuzzer based on the respective agent achieves during fuzzing.

The second research question is concerned with the overall performance of fuzzers which base their state selection on a MaB agent.

`RQ2` How do stateful fuzzers using a MaB based state selection compare to the state of the art fuzzer AFLNet?

As a baseline for our experiments, we choose AFLNet, a state of the art stateful graybox network fuzzer [18]. AFLNet is based on AFL[1] and extends it with a state machine. The state machine identifies states via the response codes to messages that were sent by the fuzzer. Before fuzzing, the SUT is set to a specific state by sending the messages required to reach this specific state first. After this, the mutated input is sent to the SUT. The input mutation is almost identical to AFL, except for a new mutation operation that concatenates different network messages as a whole. AFLNet requires a trace of a full network communication with the target in order to extract a reasonable starting state machine and valid inputs. For state selection, AFLNet by default uses its own schedule based on a manually crafted heuristic based on the number of times a state has been used and on how successful the fuzzer was in finding new coverage in this state [18]. For the experiments, we use the coverage to compare the performance of the fuzzers again.

1. https://github.com/google/AFL

### 3.2. Potential Impact

Our work will provide insights on how to model the state selection problem of a stateful network fuzzer as a MaB problem. In addition, our work will provide insights on what agents can learn regarding the state selection and how this information can be leveraged to provide a more efficient state selection. With a more efficient state selection, fuzzers can find vulnerabilities in different depths of a SUT faster and more accurately. This can improve the overall performance of a fuzzer regarding code coverage and found vulnerabilities. As a result, vulnerabilities can be found faster and closed earlier, and thus the affected SUT can be made more secure.

### 3.3. Evaluation Strategy

In general, we base our evaluation on the recommendations by Klees et al. [13]. This includes executing each of the fuzzing runs for at least 24 hours, and to repeat each configuration at least 30 times to account for the randomness of the fuzzer and the agents. To ensure transferable and robust results, the fuzzers will be evaluated against several stateful network protocols. As a baseline, the state of the art graybox fuzzer AFLNet will be used.

This work includes preliminary experiments of the two research questions. Note that this only includes four different agents, one stateful network protocol as SUT, and 10 repetitions of each configuration. AFLNet is used as a baseline for the evaluations. As has been stated, we will expand our evaluation by including more agents and a broader set of evaluation SUTs in the future. Note that a state machine for each of the SUTs needs to be constructed first.

## 4. Modeling

We model the state selection problem in stateful network fuzzing as MaB problem to be able to leverage existing algorithms. The agent's task is to select the state of the SUT to be tested in the next testing phase. Overall, the goal is to find new coverage and crashes. For the preliminary evaluation presented in this work, we decided on a modeling approach that is presented in the following. In the future, we plan to propose and evaluate more approaches, especially for modeling the reward.

**Actions.** For the definition of actions in the MaB problem, we assume that

(1) a deterministic finite-state machine with states $S$ of the SUT is available,

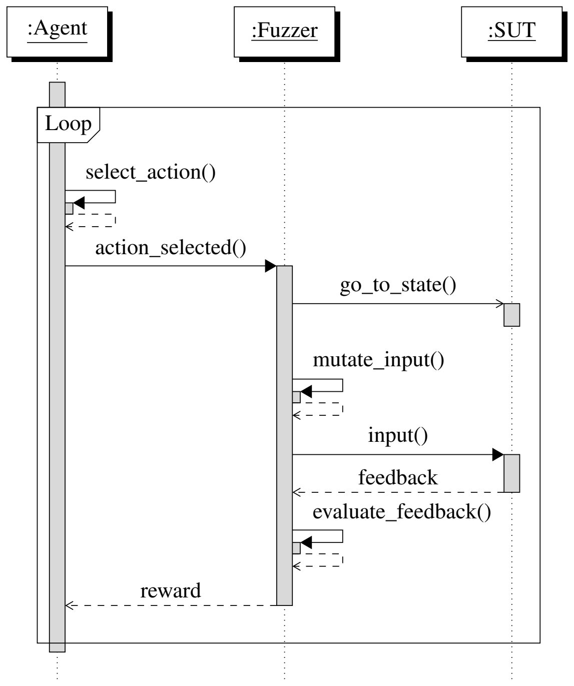

Figure 1. Abstract sequence diagram of the interaction between the agent, the fuzzer, and the SUT.

(2) each state in $S$ is reachable from the start state, and

(3) that the SUT can be coerced into a specific state.

With this, the specifics of state machine itself can be transparent to the agent. Consequently, we define the set of possible actions $A$ to be the set of states $S$. Note that $S$ represents the states of the SUT, whereas the agent itself is stateless since it is non-contextual. In each round, the agent selects one of the states of the SUT's state machine by choosing one action. This state is then used by the fuzzer in the next fuzzing iteration.

The interaction between the agent, the fuzzer, and the SUT is shown in Figure 1, and a corresponding description follows. By selecting an action, the agent selects the next state of the SUT to be tested. After the agent's decision, the SUT is put into the chosen state. In our setting, this task is done by the fuzzer, but in principle it could be done directly by the agent. Next, the fuzzer generates the fuzzing input and sends it to the SUT. The SUT executes based on the input, and it measures the code coverage and crashes produced by the fuzzing input. The code coverage and crashes are the feedback that is sent back to the fuzzer. Note that we assume a graybox setting in which the fuzzer has access to code coverage information during fuzzing. Based on the feedback, the fuzzer calculates the reward for the agent and rewards the agent accordingly. After that, one round of the fuzzing process is finished and the agent starts again with selecting the next state. Note that this process is non-deterministic since the fuzzer performs non-deterministic mutations and the agent might make non-deterministic decisions.

**Reward.** In the abstract sequence diagram (see Figure 1), the reward for the agent is calculated by the fuzzer, based on the feedback of the SUT. The agent receives a reward of 1 if new coverage was achieved, a reward of 10 if a new crash was observed, and 0 otherwise. Note that the agent is only rewarded if it finds *previously unseen* coverage or crashes, resulting in non-stationary probability distributions for the rewards.

## 5. Preliminary Experiments

In order to gain first insights into the topic and the research questions, we performed preliminary experiments. These experiments include three stochastic MaB agents, one adversarial MaB agent, and one SUT. In future work, we aim to extend the variation of the agents as well as the number of SUTs (see Section 6 for details).

### 5.1. Setup

Based on AFL++ [9] and the algorithms discussed in Section 2.2, we implemented the four MaB based fuzzers `EPS`, `EXP`, `UCB`, and `UCBT`. The three fuzzers `EPS`, `UCB`, and `UCBT` are fuzzers using a stochastic MaB, whereas `EXP` is a fuzzer using an adversarial MaB (see Section 2.2 for details on the algorithms). As baseline, we used the state of the art stateful greybox fuzzer AFLNet [18]. We executed each of the five fuzzers 10 times for 24 hours to accommodate for the fuzzers' randomness. We sped up the resulting 1200 hours of runtime by using docker to enable parallel evaluation runs. We ran our evaluation on an Ubuntu 20.04 LTS Server with an Intel® Xeon® CPU ES-1650 v3 @ 3.50GHz (12 physical cores) with 64 GB of RAM. As SUT, we chose open62541[2], an open source implementation of the stateful network protocol OPC UA. For our experiments, we included and modeled five different states of the OPC UA protocol (see e.g. the OPC UA standard[3] for more details on OPC UA). We chose the five states that are needed to establish a connection successfully and model them as the state machine shown in Figure 2. The SUT starts in an initial state. By sending OPC UA messages to the SUT, the internal state of the SUT can be changed. Since this state machine is only used to coerce the SUT into a specific state instead of representing the full behavior of the SUT, it does not need to include all possible error cases and error transitions. Note that the fuzzers also implement the corresponding closing messages which allow to go back in the state machine. However, for our preliminary experiments, we chose to only include the establishment of the connection in the intended order.

### 5.2. Results

The results of the evaluation are presented in Figures 3 and 4. Figure 3 shows the final coverage which has been achieved by 10 runs of the MaB fuzzers on the SUT open62541 as a box plot. Based on the box plot, one can come to the conclusion that the MaB fuzzers show no statistically significant performance increase over one another (`RQ1`). To verify this, we present the p-values calculated using a Mann-Whitney U Test in Table 1. Being greater than 0.05, these values indicate that we cannot reject the null hypothesis that all fuzzers result in the

2. https://www.open62541.org/
3. https://reference.opcfoundation.org/

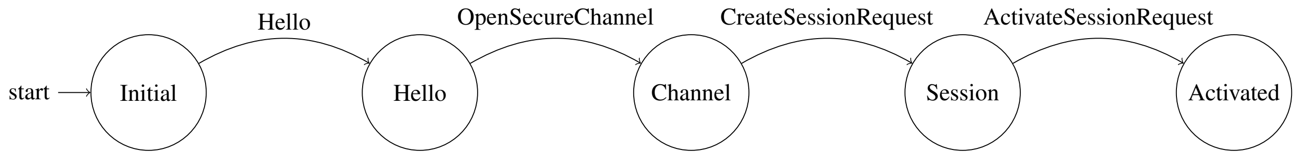

Figure 2. States of the OPC UA protocol used during the preliminary experiments. The text on the transitions represent the OPC UA message needed for the transition. The MaB agent selects one of the states for the next fuzzing phase and the fuzzer send the necessary messages to the SUT before sending the mutated fuzzing input.

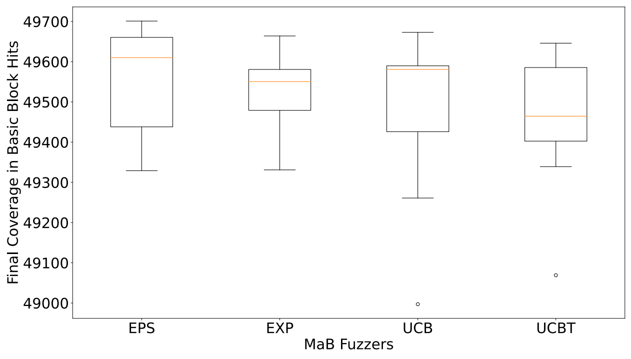

Figure 3. Box plot of the final coverages achieved by 10 runs of the MaB fuzzers. The boxes extend from the lower quartile to the upper quartile, the line in the boxes corresponds to the median, the whiskers represent 1.5 times the interquartile range, and outliers are shown as single circles. No MaB fuzzer leads to significantly different results.

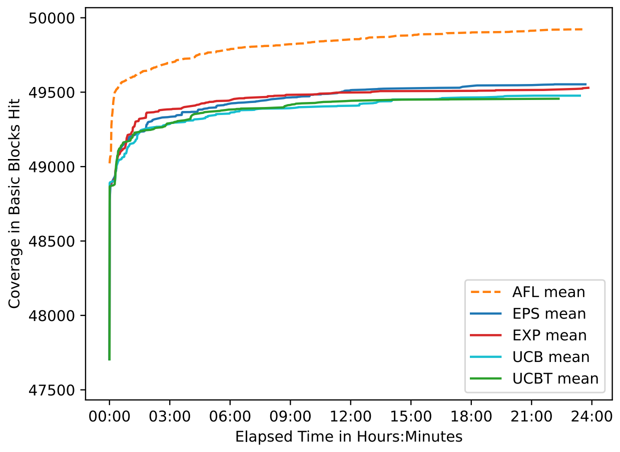

Figure 4. Coverage comparison of the MaB fuzzers and AFLNet (mean of 10 runs). The MaB fuzzers perform similarly, while AFLNet performs significantly better.

same performance. This result is interesting, since the adversarial MaB models the state selection problem more accurately, and we should thus expect the corresponding algorithm (`EXP`) to lead to better results. Nevertheless, this result is in line with the results by Liu et al. [14]. Liu et al. show in their work that previously existing approaches for state selection as well as the new approach they propose lead to indistinguishable fuzzing performances in most cases.

The plot in Figure 4 depicts the mean of the coverage in basic blocks of the different MaB fuzzers in comparison to AFLNet on the SUT open62541 over time. The orange dotted line is the baseline AFLNet, whereas the solid lines correspond to the different fuzzers implementing the MaB

TABLE 1. RESULTING P-VALUES OF A MANN-WHITNEY U TEST REGARDING THE STATISTICAL SIGNIFICANCE OF THE DISTANCE OF THE RESULTING COVERAGES. NO ALGORITHM PERFORMS SIGNIFICANTLY BETTER.

| | UCB | UCBT | EPS | EXP |
|---|---|---|---|---|
| UCB | - | 0.545 | 0.364 | 0.879 |
| UCBT | 0.545 | - | 0.198 | 0.449 |
| EPS | 0.364 | 0.198 | - | 0.542 |
| EXP | 0.879 | 0.449 | 0.542 | - |

algorithms. Overall, AFLNet performs significantly better than all the MaB fuzzers (`RQ2`). Our analysis shows that one reason for this is the difference in inputs. AFLNet requires one full sequence of input messages, while the MaB fuzzers only receive information about individual states (see Section 2.3). This knowledge allows AFLNet to start analyzing deep states directly, while the MaB fuzzers need to identify the deep states first. This insight represents one starting point for further research and on how to make MaB fuzzers more efficient.

## 6. Future Work

Based on the preliminary experiments, we aim to extend the approaches and experiments to provide deeper insights into the modeling of the state selection problem as a MaB problem. On the one hand, this will include different approaches on modeling the reward, and choosing the MaB algorithm and inputs for the agent. In particular, we aim to further investigate why the adversarial MaB approach did not perform better than the stochastic MaB models. In order to provide a modular structure for the various approaches, we aim to re-implement our approaches building upon the modular fuzzing framework LibAFL [10]. On the other hand, we aim to perform the evaluation with more SUTs. This will include, amongst others, SUTs similar to the ones used by Liu et al. [14], in order to achieve comparable results. In addition, this will provide the possibility to analyze the protocol-depenent behavior of the agents in more detail. In order to gain deeper insights into how and what the agents learn, we also aim to perform in-depth analysis of the agents and to perform hyperparameter tuning.

The next step, building upon the insights on modeling the state selection problem as MaB, would be to combine the state selection with an automated learning of the state graph. Automated learning of a protocol state graph has been done for graybox settings (e.g. [16]), and for blackbox settings (e.g. [6], [7]). With this, we could omit the assumption of a given state graph. On top of this, one could look into other modeling approaches for the states of the SUT (e.g. Markov chains [11]).

## 7. Conclusions

Our work contributes further to the question of whether or not the state selection algorithm has a significant impact on the results achieved by the fuzzer. Our preliminary experiments show that the choice of the agent's algorithm has no statistically significant impact, which supports results from literature. Nevertheless, we plan to analyze the modeling using MaB in greater depth to fully understand its implications and potentials. For this, we plan to perform in-depth hyperparameter tuning to understand the impact of hyperparameters on the fuzzer's performance. This includes providing additional knowledge, such as the depth of states, to the MaB fuzzers. In addition, we aim to analyze the learning phase of the agents to see what they learn during their interaction with the SUT. To go a step further, state selection could be combined with state graph learning, omitting the assumption of a given state graph.

With this, we contribute to more efficient stateful network fuzzing approaches. As a result, fuzzing can find vulnerabilities in ICS faster and thus the vulnerabilities can be closed earlier.

## References

[1] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47:235–256, 2002.

[2] Peter Auer, Nicolo Cesa-Bianchi, Yoav Freund, and Robert E Schapire. The nonstochastic multiarmed bandit problem. *SIAM journal on computing*, 32(1):48–77, 2002.

[3] Jinsheng Ba, Marcel Böhme, Zahra Mirzamomen, and Abhik Roychoudhury. Stateful greybox fuzzing. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 3255–3272, 2022.

[4] Lorenzo Binosi, Luca Rullo, Mario Polino, Michele Carminati, Stefano Zanero, et al. Rainfuzz: Reinforcement-learning driven heatmaps for boosting coverage-guided fuzzing. In *Proceedings of the 12th International Conference on Pattern Recognition Applications and Methods-ICPRAM*, pages 39–50. SciTePress, 2023.

[5] Konstantin Böttinger, Patrice Godefroid, and Rishabh Singh. Deep reinforcement fuzzing. In *2018 IEEE Security and Privacy Workshops (SPW)*, 2018.

[6] Joeri De Ruiter and Erik Poll. Protocol state fuzzing of TLS implementations. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 193–206, 2015.

[7] Adam Doupé, Ludovico Cavedon, Christopher Kruegel, and Giovanni Vigna. Enemy of the state: A state-aware black-box web vulnerability scanner. In *Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)*, pages 523–538, 2012.

[8] Kaiming Fang and Guanhua Yan. Emulation-instrumented fuzz testing of 4g/lte android mobile devices guided by reinforcement learning. In *Computer Security: 23rd European Symposium on Research in Computer Security, ESORICS 2018, Barcelona, Spain, September 3-7, 2018, Proceedings, Part II 23*, pages 20–40. Springer, 2018.

[9] Andrea Fioraldi, Dominik Maier, Heiko Eißfeldt, and Marc Heuse. Afl++ combining incremental steps of fuzzing research. In *Proceedings of the 14th USENIX Conference on Offensive Technologies*, pages 10–10, 2020.

[10] Andrea Fioraldi, Dominik Christian Maier, Dongjia Zhang, and Davide Balzarotti. Libafl: A framework to build modular and reusable fuzzers. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 1051–1065, 2022.

[11] Hugo Gascon, Christian Wressnegger, Fabian Yamaguchi, Daniel Arp, and Konrad Rieck. Pulsar: Stateful black-box fuzzing of proprietary network protocols. In *Security and Privacy in Communication Networks: 11th EAI International Conference, SecureComm 2015, Dallas, TX, USA, October 26-29, 2015, Proceedings 11*, pages 330–347. Springer, 2015.

[12] IEC 62443 Security for Industrial Automation and Control Systems. Standard, Internation Electrotechnical Commission, Geneva, CH, 2009-2020.

[13] George Klees, Andrew Ruef, Benji Cooper, Shiyi Wei, and Michael Hicks. Evaluating fuzz testing. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, pages 2123–2138, 2018.

[14] Dongge Liu, Van-Thuan Pham, Gidon Ernst, Toby Murray, and Benjamin IP Rubinstein. State selection algorithms and their impact on the performance of stateful network protocol fuzzing. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 720–730. IEEE, 2022.

[15] Valentin JM Manès, HyungSeok Han, Choongwoo Han, Sang Kil Cha, Manuel Egele, Edward J Schwartz, and Maverick Woo. The art, science, and engineering of fuzzing: A survey. *IEEE Transactions on Software Engineering*, 47(11):2312–2331, 2019.

[16] Chris McMahon Stone, Sam L Thomas, Mathy Vanhoef, James Henderson, Nicolas Bailluet, and Tom Chothia. The closer you look, the more you learn: A grey-box approach to protocol state machine learning. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 2265–2278, 2022.

[17] Roberto Natella. Stateafl: Greybox fuzzing for stateful network servers. *Empirical Software Engineering*, 27(7):191, 2022.

[18] Van-Thuan Pham, Marcel Böhme, and Abhik Roychoudhury. Aflnet: A greybox fuzzer for network protocols. In *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*, pages 460–465, 2020.

[19] Aleksandrs Slivkins. Introduction to multi-armed bandits. *Foundations and Trends® in Machine Learning*, 12(1-2):1–286, 2019.

[20] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[21] Jinghan Wang, Chengyu Song, and Heng Yin. Reinforcement learning-based hierarchical seed scheduling for greybox fuzzing. In *Network and Distributed Systems Security (NDSS) Symposium 2021*, 2021.

[22] Yan Wang, Peng Jia, Luping Liu, Cheng Huang, and Zhonglin Liu. A systematic review of fuzzing based on machine learning techniques. *PloS one*, 15(8):e0237749, 2020.

[23] Maverick Woo, Sang Kil Cha, Samantha Gottlieb, and David Brumley. Scheduling black-box mutational fuzzing. In *2013 ACM SIGSAC conference on Computer & communications security*, pages 511–522, 2013.

[24] Tai Yue, Pengfei Wang, Yong Tang, Enze Wang, Bo Yu, Kai Lu, and Xu Zhou. EcoFuzz: Adaptive Energy-Saving greybox fuzzing as a variant of the adversarial Multi-Armed bandit. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 2307–2324, 2020.