Acta Polytechnica **59**(5):518–526, 2019
DOI:10.14311/AP.2019.59.0518

MODEL-BASED SECURITY ANALYSIS OF FPGA DESIGNS THROUGH REINFORCEMENT LEARNING

Michael Vetter

University of West Bohemia, Faculty of Applied Sciences, Department of Computer Science and Engineering, Technická 8, 301 00 Plzeň, Czech Republic

correspondence: michaelvetter@protonmail.com

Abstract. Finding potential security weaknesses in any complex IT system is an important and often challenging task best started in the early stages of the development process. We present a method that transforms this task for FPGA designs into a reinforcement learning (RL) problem. This paper introduces a method to generate a Markov Decision Process based RL model from a formal, high-level system description (formulated in the domain-specific language) of the system under review and different, quantified assumptions about the system's security. Probabilistic transitions and the reward function can be used to model the varying resilience of different elements against attacks and the capabilities of an attacker. This information is then used to determine a plausible data exfiltration strategy. An example with multiple scenarios illustrates the workflow. A discussion of supplementary techniques like hierarchical learning and deep neural networks concludes this paper.

Keywords: FPGA, IT security, model-driven design, reinforcement learning, machine learning.

1. Introduction

Securing any (non-trivial) computer system against malicious actors is a challenging task. The defender has to identify and protect all possible paths of entry while an attacker has to find only one feasible way to breach the system's security properties. As a system matures, new threats can arise and the strength of defence mechanisms may erode. Current threat modelling approaches are often informal, provide little automation and have a tendency to neglect the defence in-depth aspect of a design by focusing on the attack surface. Attack trees [1] are a common method to model an attacker's options but their creation is often tedious and they provide little information about the most efficient way to breach a system. We address this problem for FPGA [2] based designs through a combination of a text-based Domain Specific Language (DSL) for the design description and a quantified assessment of the security properties. This description is automatically transformed into a processable Markov Decision Process (MDP). An agent is trained on this MDP to exfiltrate all data stored within an FPGA design using the most efficient sequence of predefined actions. This task is performed by well-established reinforcement learning (RL) [3] [4] algorithms. The result of this analysis is one or more attack sequences that provide the user with insights about an attacker's strategy under the described circumstances. We illustrate the feasibility of this approach through a series of experiments, assess its constraints and conclude with a discussion about our model's limits and possible methods to lift them.

2. Related Work

The approach described in this work operates on the intersection between IT security, model-driven development (MDD) and machine learning.

Model-driven development [5] is a common technique to master the complexity of modern systems. General purpose languages like SysML [6] and AADL [7] flatten the learning curve, advance best practices and prevent vendor lock-in, but, to the authors knowledge, no standardized support for a security analysis exists in either of these languages. Proposals like [8] for AADL have been made but not widely adopted. Threat Modelling [9] is often performed informally using tools as simple as paper, whiteboards or general purpose diagram software. Dedicated threat modelling tools, including Microsoft Threat Modelling Tool [10] and OWASP Threat Dragon [11], provide a suitable structure and user interface but little automation.

Machine learning has seen a boom in recent years, mainly powered by a trifecta of big data, parallel data processing power provided by GPUs (Graphics Processing Unit) and new ML architectures that utilize both. [12] discusses the application of machine learning for a malware detection from a practitioner's perspective.[13] presents both a comprehensive review of the scientific literature and a number of research papers with applications ranging from the detection of malware to automatically generated penetration test plans. Machine Learning has also been used to generate malicious inputs with complexity beyond the conventional fuzzing [14] methods. [15] presents a Generative Adversarial Networks (GAN) [16] based attack against a fingerprint scanner. In [17], a similar approach is used against the computer vision system of an electric vehicle. In [18], the authors use Markov

Decision Processes to craft stealthy attack sequences against a cyber-physical system. MDPs can be used for defensive purposes e.g. to detect attacks [19]. The Cyber Grand Challenge 2016 [20] demonstrated that modern computer programs are, in principle, able to detect, exploit and patch previously unknown vulnerabilities in other cyber systems without a human intervention. It is generally assumed that most research in this field is done by military contractors and intelligence agencies and, therefore, a secret [21] Advances in other, more visible, domains like games can provide us with valuable insights about the capabilities and limits of current machine learning systems. Milestones for this progress include Chess with Deep Blue (Chess) [22], Jeopardy with Watson [23], Pacman[24] and diverse ATARI 2600 arcade games [25], the board game GO [26], as well as the real-time strategy game Starcraft II [27] and Quake III [28]. Unresolved, but intensely researched problems, like autonomous driving, indicate the limits of the current technology when the problem space becomes too large.

Advances in other fields of statistics have provided us with methods to determinate casual effects [29][30] in the absence of randomized experiments and the exploration of counterfactual scenarios. Progress in probabilistic programming [31] [32] have eased the usage of Bayesian methods to e.g. incorporate knowledge about a system as informed priors.

We can conclude that the advancement of machine learning in recent years can and has been used to improve both attacks and defences of existing cyber physical systems. The machine learning tools available today must be tailored to the application at hand and used with a careful knowledge of their limits. To the author's knowledge, no such system exists that uses reinforcement learning to determine a systems vulnerabilities in the analysis and design phase.

3. The descriptive model of the FPGA design

It is common wisdom that security problems should best be solved at the early stages of the development process, ideally before any hardware is built or code written. This approach requires a suitable model of the system to be created. The FPGASECML metamodel (Figure 1) defines several components that can be instantiated and parameterized for this purpose. FPGAModules represent the computational parts of the design. They come in two different manifestations: Processing Blocks store and process data while IO Blocks provide an additional connection to the outside world. FPGAModules utilize Slots to perform their tasks. Slots represent disjunctive sets of FPGA primitives like Configurable Logic Blocks and Embedded RAM. Each Slot can be used by only one FPGAModule at a time and all FPGAModules must use the same predefined Slot every time. Communication Networks represent the communication infrastructure between the Slots. They are directed graphs with the Slots as nodes and the possible data flow between them is represented by the edges. Each change in the Slot utilization (partial runtime reconfiguration) is represented by a Reconfiguration Event. Each Reconfiguration Event has a set of FPGAModules that replace the FPGAModule in their respective Slot. The only mandatory event evInit marks the initial configuration of the FPGA and must provide one FPGAModule for each Slot.

4. The MDP-Representation of the FPGA design

Our model-to-model generator translates each valid FPGASECML description into a computable Markov Decision Processes model. This section presents a short introduction into MDPs in general, and the simplifications made to create a suitable representation of the problem at hand. We further discuss the different components - state, action, reward and terminal states - of our model (Figure 2) and its implementation.

4.1. Scope and constraints of the model

We assume that the attacker has only one goal: the exfiltration of all data stored within the FPGA. We further assume that the agent has a full access to all peripheral devices including those containing the FPGAs configuration. Each action in the MDP-model represents a whole class of attacks, many (low-level) operations might be necessary to execute each of these actions in a real-world case. The model does not support side effects of attacks like a damaged configuration. We further grant the attacker total knowledge over the systems state, and this state depends solely on the actions of the attacker. Methods to address these constraints are presented later.

4.2. Generic components of Markov Decision Processes

Each State of the Markov Decision Processes (MDP) contains all the information necessary to determine the next state as well as the agent's reward after an action is taken. Each MDP has one start state and at least one terminal state. The agent can choose from a finite set of actions to change the state of the system. Executing this action results in the transition from the state s to state s'. All actions are atomic and require the same amount of time. The probability $P_a(s, s')$ specifies the likelihood that an action a in the state s leads to a transition into the state s'. The reward $R_a(s, s')$ informs the agent which transitions are desirable and which are not. The agent receives its reward immediately after the transition into s'. Rewards can also be negative (punishments) or zero. γ or Gamma is a discount factor for future rewards (we use the traditional value of $\gamma = 0.7$).

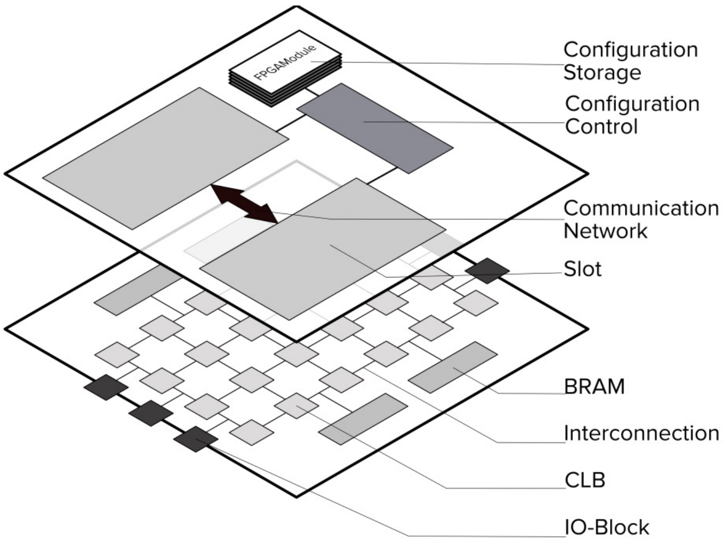

Figure 1. FPGA primitives (bottom layer) and their abstract representation in FPGASECML (top layer).

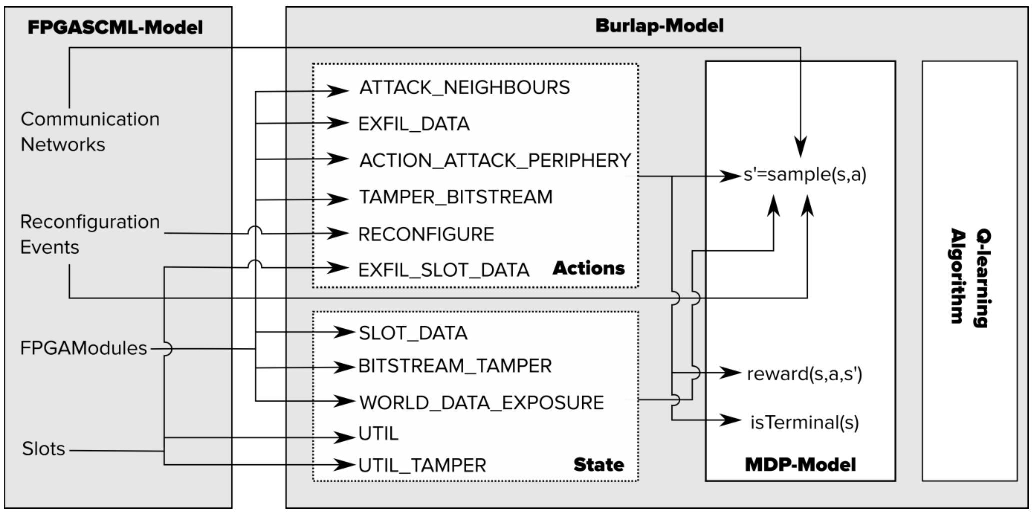

Figure 2. Transformation flow from the FPGASECML to the BURLAP based reinforcement learning model.

4.3. The domain-specific Markov Decision Process

This section describes how the FPGASECML meta-model is translated into the required MDP components.

4.3.1. State

The state contains the current configuration of the FPGA, which FPGAModule is under the control of the attacker and which Slot contains data already exfiltrated from another FPGAModule (chain exfiltration). The state further encodes which FPGAModules storage has been tampered with and which data has already been leaked into the outside world. All the information are stored in integer or Boolean variables. The FPGA starts in its initial configuration (as defined by evInit) and all FPGAModules untampered with. Terminal states are, by default, all states where the agent has exfiltrated the data of each FPGAModule into the outside world. The code generator does not validate whether the agent can reach a terminal state.

4.3.2. Actions

The agent can invoke one of the (pre-defined) actions to get and extend control over the system. Possible actions include the reconfiguration of the system, tampering with a distinct FPGAModule (inside the FPGA or in its storage element) and the extraction of data from a tampered-with FPGAModule. The attacker can further transfer these extracted data to other Slots or the outside world (if the data is in an IOBlock that the attacker has already tampered with.) The MDP moves into the new state if the action is successful and remains in the current state ($s' = s$) otherwise.

4.3.3. Reward

The agent receives a positive reward for reaching a terminal state and either none or a negative reward (punishment) for every other step. This penalty encourages the agent to use shorter sequences of low-cost actions and constraints the search space. It can be the same for all actions but it is also possible to penalize the usage of distinct actions.

4.4. Reinforcement Learning Scenarios

Multiple experiments with different parameters must be performed for a meaningful analysis. Each scenario is a combination of MDP parameters like the reward, the success likelihood and costs of actions as well as a hyperparameter like the learning rate. FPGASECML allows the definition of scenarios that allow the independent definition and evaluation of these parameters. All scenarios share a set of common parameter like γ and the exploration-exploitation tradeoff ϵ set. The example in the next section illustrates the application of scenarios.

4.5. Implementation

Our proof of concept implementation transforms, as mentioned earlier, any valid FPGASECML-model into Java code. This generated code is meant to be linked against the Brown-UMBC reinforcement learning and Planning (BURLAP) library [33], a JAVA framework that provides a generic reinforcement learning functionality. Where the algorithmic flexibility provided by BURLAP is not required, a reimplementation in e.g., C++,Rust or Go should lead to a better performance. The memory consumption could be reduced through an optimized state encoding at the expense of readability and extensible - the implementation used here is optimized for a simple code generation process.

5. Example

An FPGA design consists of three Slots with two FPGAModules each (Figure 3). FPGAModule A is the only IO Block. The run ends the data of all six Modules (A-F) is exfiltrated. The Q-learning algorithm is used to find the best sequence of actions. The heap memory of the JAVA virtual machine (JVM) had to be extended to up to 6 GB to accommodate the Qtable as well as the telemetric data. Each scenario is executed (ran) once for each epsilon (0.1, 0.05 and 0.25 in our example). Each run consists of 3 million episodes, and, in each episode, the agent starts at the initial state and ends in a terminal state. Histograms (truncated to the right) are used to compare the agent's learning performance for the different *epsilon* and scenarios.

The FPGASECML-model, as well as the generated JAVA code and reports, for this example, can be found in the accompanying files.

5.1. Scenario 1: Baseline

Scenario 1 establishes a baseline of the model through a deterministic MDP (the success rate of every action is 1) and a fixed reward/punishment assignment. The agent gets a reward of 200.000 for reaching a terminal state and a punishment of -1.000 for every other action. All three runs return a minimal solution with 24 actions, the histogram (Figure 5) of all three actions shows that the epsilon of 0.1 has the most episodes with the minimal steps followed by 0.05 (low exploration rate requires more steps to find a solution) and 0.25 (too much exploration hinders the efficient exploitation of the available data.) The learning process starts with episodes well over 2.000 steps (Figure 4) and provides better results as the Qtable gets filled. For the *epsilon* of 0.1, the first minimal solution is found after just 303.323 Episodes - indicating that the agent has either found the best possible solution or is stuck in a local minimum with little chances to break out. It is also notable that all three solutions make an extensive use of the tamper−bitstream action, but what if this action is unlikely to succeed or prohibitively expensive?

5.2. Scenario 2: Tamper resistance storage prevents most attacks

The second scenario assumes that only one in a thousand attacks against tamper-resistant bitstream storage is successful (relative to all other attacks as success factors are normalized.) The three runs return a minimal sequence of 25 actions each and all three of them avoid the tamper−bitsteam actions completely. The histogram (Figure 5) shows a smaller spread than that of the first scenario and the median of the episodes for each run has shrunk from (29,39,44) to (28,26,35).

5.3. Scenario 3: Faulty encryption eases attacks

We assume that a weaker tamper protection of the bitstream storage and a weakness in the bitstream encryption mechanism increases the success rate of tamper−bitstream actions. The minimal sequences returned require 27 steps for epsilon of 0.1, 26 for 0.05 and 30 for 0.25 - a significantly worse performance than in the previous scenarios. The histogram of all three runs also skews much more to the right than those of the preceding scenarios. Worse, the three attack sequences have all 4 of the 6 bitstreams tampered with. The success rate of one in a thousand was, apparently, low enough to discourage the agent from using these actions while the "one in five" -chance is too weak to achieve a similar effect. The low success rate is, however, strong enough to prolong the learning process as the median number of steps per episode rises from previously (28,26,35) to (154,155,152). The learning progress (Figure 7) is much noisier than it was in the baseline scenario (Figure 4.)

5.4. Scenario 4: Physical attacks against storage devices are expensive

Not all attacks require the same amount of resources - some take longer than others, a few require costly

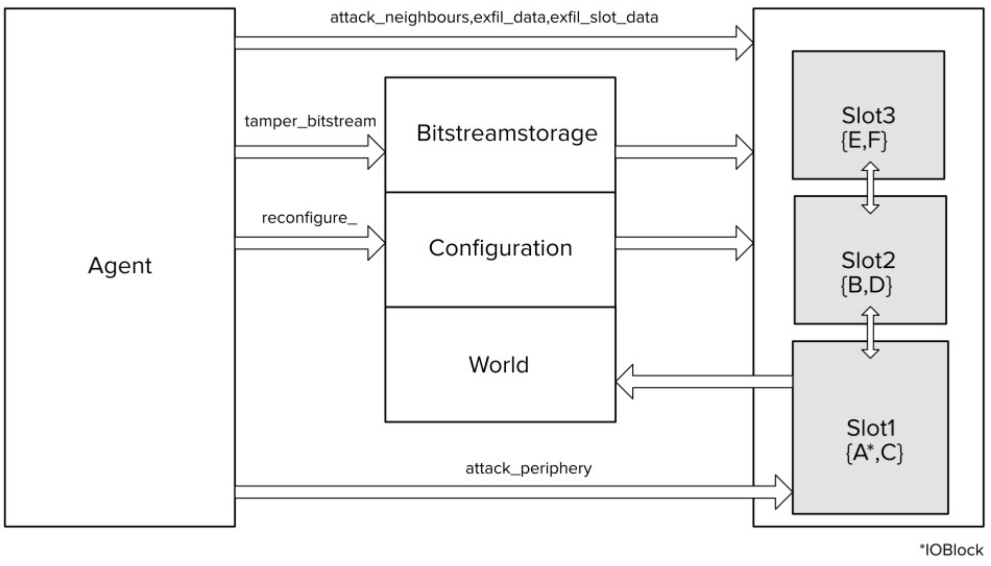

Figure 3. Schematic of the FPGASECML to burlap example.

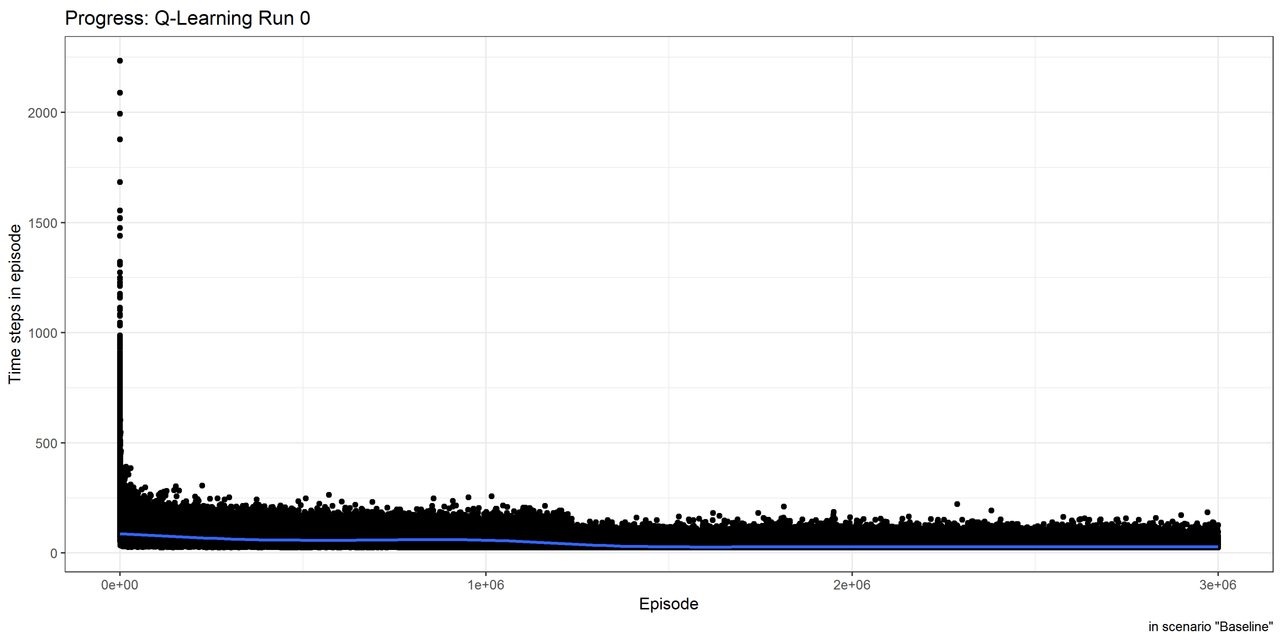

Figure 4. Number of actions necessary to reach a terminal state for each Episode in the Baseline scenario (the line at the bottom represents the running average).

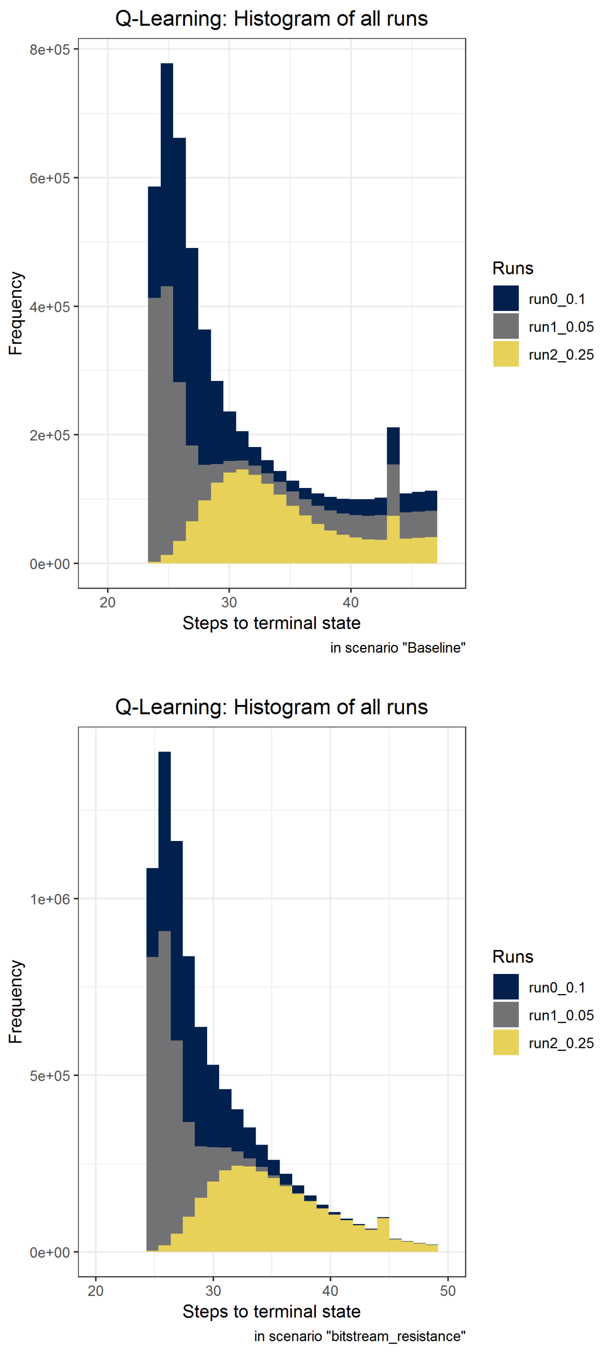

Figure 5. Histogram of multiple runs with different epsilons for scenario 1 (top) and 2 (bottom).

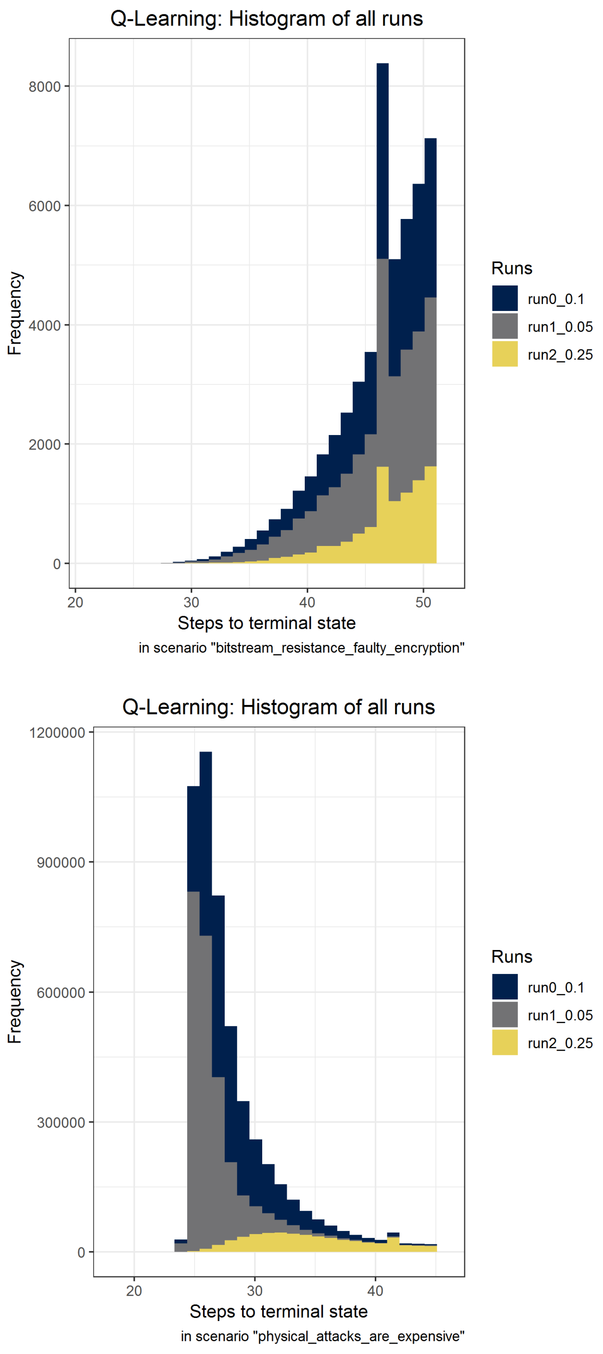

Figure 6. Histogram of multiple runs with different epsilons for scenario 3 (top) and 4 (bottom).

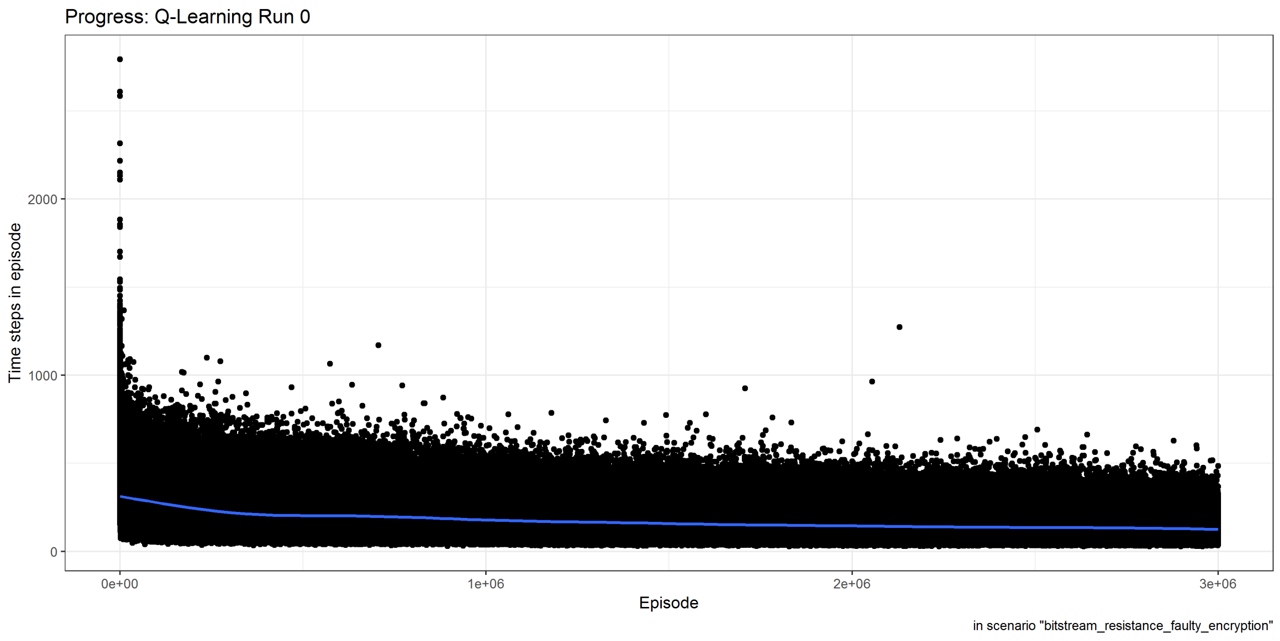

Figure 7. Training progess for scenario 3 - faulty encryption eases attacks.

equipment and highly specialized threat agents. Reducing the success rate of an attack makes it implicitly more expensive (as, on average, more attacks must be performed before succeeding) but with several million trials per run, there is a good chance that at least one sequence of highly improbable actions will succeed at least once. A real attacker might not want to take this risk. We convert our MDP back into a DMDP and assign the tamper−bitstream action an additional cost weight of 100.000 (this cost is subtracted from the default reward of -1.000.) The minimal sequence for the epsilons 0.1 and 0.25 are 24, but the low exploration rate of 0.05 has led to a global minimum of 23. It is also notable that this global minimum is not reached until the 2.543.660th episode and that the agent tampers with two bitstream storages while the other two use this expensive operation only once.

Imposing even higher costs (bitstream tampering is extremely expensive) on the tamper−bitstream operation had no positive effect with all three runs returning a sequence of the length 24.

5.5. Results

We can conclude that our approach found a reasonable solution to this problem within a feasible amount of computations. The noisy progress of the RL algorithm makes it impossible to determine whether a local or global minimum has been found. Scenarios can be used to assess the impact of high costs and low success rates and to find better solutions by constraining the size of the search space. A plausibility check of the results (here performed on the generated sequence) is mandatory, as for all machine learning methods. The number of training episodes required for this small example indicates challenges for intricate designs and models with a richer state, action representation. The memory consumption of the program, presumably driven by the expanding Qtable, supports this assumption. Finding the appropriate cost/reward structure remains a challenge and any analysis should include multiple scenarios with different cost and success rate values.

6. Limits and restrictions of the MDP based approach and possible solutions

The MDP based model presented here relies on certain constraints that could be lifted by supplementing the proposed model. Hierarchical learning [34] can be used to combine this high-level model with the low-level activities needed to execute the strategy. It is also plausible that the attacker does not know the state of the system but has to guess it from a limited set of, presumably noisy, indicators available (like physical side channels [35][36]). The reinforcement learning equivalent of this effect is a Partially Observable MDP (POMDP [37] - BURLAP provides limited support for POMDP) Transforming the MDP into a stochastic game allows the integration of multiple actors (e.g. an active defence mechanism.)

Navigating the vast search space remains the main problem of the reinforcement learning. Limiting the content of the state and the number of actions eases this problem but restricts the expressiveness of the model. Constraining the search space through transition probabilities and costs is another method but increases the number of experiments. A functional approximation of the Qtable could provide an avenue towards an improved RL based weakness analysis. One candidate for this approximation are deep neural networks, whose general feasibility has been demonstrated by the DeepQ [38] algorithm. This more recent technique replaces the Qtable with a neuronal network that approximates the value of a given state-action combination. Suitable neural networks mitigate the state explosion problem and should be able to detect patterns. They may, therefore, be used to extract and abstract information from a feature-rich state, action representation in the same way convolution networks extract information e.g from a picture [16]. Policy shaping [39] or reward learning [40] provide other avenues to explore. A sufficiently competent and autonomous reinforcement learning system could be pitted against real-life systems (preferably in a laboratory environment) for further refinement without a human interference (similar to AlphaZero [41].) These proposals require at least a rudimentary implementation of the design and the insight gained from attacking this system could be transferred to other systems still in the design and analysis phase.

Finding the best parameters for the reward and success rates represents an additional challenge. Different domain experts may have different assessments of the threat landscape or varying confidence in their assumptions. Running a large number of scenarios with different parameters also increases the chance to find an efficient and robust solution. In later stages of the development cycle, data from penetration tests and from security incidents can be used to verify the validity of the assumptions made. To simplify the creation of scenarios, the point values for the cost and success rate could be replaced with distributions (similar to hierarchical learning in probabilistic programming) where the actual values for each run are drawn from.

7. Conclusion

Reinforcement learning can be used to identify potential weaknesses in an FPGA design and the steps a reasonable attacker may take to exploit them. We presented a method to generate a Markov Decision Process based reinforcement learning model from a formal, high-level system description (formulated in the domain-specific language FPGASECML.) The automatic model-to-model translation reduces the developer's workload, decreases the risk of errors and helps to keep both models in sync. Probabilistic transitions and the reward function can be used to model the varying resilience of different elements against attacks and the capabilities of an attacker. Supplementary techniques like hierarchical learning, Partially Observable MDPs, and stochastic games can be used to extend the scope of the model.

References

[1] B. Schneier. Academic: Attack Trees: Schneier on Security. `https://www.schneier.com/academic/archives/1999/12/attack_trees.html`.

[2] S. M. Trimberger, J. J. Moore. FPGA Security: Motivations, Features, and Applications. *Proceedings of the IEEE* **102**(8):1248–1265, 2014. doi:10.1109/JPROC.2014.2331672.

[3] C. Isbell, M. Littman. Reinforcement learning - udacity. `https://classroom.udacity.com/courses/ud600`.

[4] F. Akhtar. *Practical Reinforcment Learning.* Packt Publishing Limited, 2017.

[5] M. Brambilla, J. Cabot, M. Wimmer. *Model-driven software engineering in practice.* Synthesis lectures on software engineering. Morgan & Claypool Publishers, second edition edn., 2017.

[6] L. Delligatti. *SysML distilled: A brief guide to the systems modeling language.* Addison-Wesley Professional, 2014.

[7] J. Delange. *AADL in practice.* Reblechon Development Co, 2017.

[8] Robert Ellison, Allen Householder, John Hudak, Rik Kazman, Carol Woody. Extending AADL for security design assurance of cyber-physical systems. `https://resources.sei.cmu.edu/asset_files/TechnicalReport/2015_005_001_449522.pdf`.

[9] F. Swiderski, W. Snyder. *Threat modeling.* Microsoft Press, Redmond and Wash, 2004.

[10] JEGEIB. Getting started - microsoft threat modeling tool - azure. `https://docs.microsoft.com/en-us/azure/security/azure-security-threat-modeling-tool-getting-started`.

[11] Threat Dragon. `https://threatdragon.org/login`.

[12] J. Saxe, H. Sanders. *Malware Data Science: Attack Detection and Attribution.* No Starch Press Incorporated, San Francisco, CA, 2018.

[13] S. Parkinson, A. Crampton, R. Hill. *Guide to Vulnerability Analysis for Computer Networks and Systems: An Artificial Intelligence Approach.* Computer Communications and Networks. Springer, 2018.

[14] M. Sutton, A. Greene, P. Amini. *Fuzzing: Brute force vulnerabilty discovery.* Addison-Wesley, Upper Saddle River, N.J., 2007.

[15] P. Bontrager, A. Roy, J. Togelius, et al. DeepMasterPrints: Generating MasterPrints for Dictionary Attacks via Latent Variable Evolution. `https://arxiv.org/pdf/1705.07386.pdf`.

[16] I. Goodfellow, Y. Bengio, A. Courville. *Deep learning.* MIT Press, Cambridge, Massachusetts and London, England, 2016.

[17] Tencent Keen Security Lab. Autopilot. `https://keenlab.tencent.com/en/whitepapers/Experimental_Security_Research_of_Tesla_Autopilot.pdf`.

[18] S. Lakshminarayana, T. Z. Teng, D. K. Y. Yau, R. Tan. Optimal attack against cyber-physical control systems with reactive attack mitigation. *Proceedings of the Eighth International Conference on Future Energy Systems* **2017**, 2017. doi:10.1145/3077839.3077852.

[19] H. Koduvely. Anomaly detection through reinforcement learning. `http://blog.zighra.com/anomaly-detection-and-reinforcement-learning`.

[20] Cyber Grand Challenge (CGC). https://www.darpa.mil/program/cyber-grand-challenge.

[21] Mayhem, the tech behind the DARPA Grand Challenge winner, now used by the Pentagon - CyberScoop. https://www.cyberscoop.com/mayhem-darpa-cyber-grand-challenge-dod-voltron/.

[22] Chandrasekaran, Rajiv. Washingtonpost.com: Deep blue defeats Kasparov in game 2. https://www.washingtonpost.com/wp-srv/tech/analysis/kasparov/kasparov.htm?

[23] N. Jones. Quiz-playing computer system could revolutionize research. https://www.nature.com/news/2011/110215/full/news.2011.95.html.

[24] H. van Seijen, M. Fatemi, J. Romoff, et al. Hybrid reward architecture for reinforcement learning. *CoRR* **abs/1706.04208**, 2017.

[25] V. Mnih, K. Kavukcuoglu, D. Silver, et al. Playing Atari with Deep Reinforcement Learning.

[26] AlphaGo: using machine learning to master the ancient game of Go. https://blog.google/topics/machine-learning/alphago-machine-learning-game-go/.

[27] O. Vinyals, I. Babuschkin, J. Chung, et al. AlphaStar: Mastering the Real-Time Strategy Game StarCraft II. https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/.

[28] Capture the Flag: the emergence of complex cooperative agents | DeepMind. https://deepmind.com/blog/capture-the-flag-science/.

[29] J. Pearl. Causal inference in statistics: An overview. *Statistics Surveys* **3**(0):96–146, 2009. DOI:10.1214/09-SS057.

[30] J. Pearl, D. Mackenzie. *The book of why: The new science of cause and effect.* First edition edn., 2018.

[31] C. Davidson-Pilon. *Bayesian methods for hackers: Probabilistic programming and Bayesian methods.* Addison-Wesley data and analytics series. Addison-Wesley, New York, 2016.

[32] O. Martin. *Bayesian Analysis with Python: Introduction to Statistical Modeling and Probabilistic Programming Using PyMC3 and ArviZ, 2nd Edition.* Packt Publishing Ltd, Birmingham, 2nd edn., 2018.

[33] BURLAP. http://burlap.cs.brown.edu/.

[34] R. S. Sutton, D. Precup, S. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence* **112**(1-2):181–211, 1999. DOI:10.1016/S0004-3702(99)00052-1.

[35] S. Mangard, E. Oswald, T. Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards.* Springer eBook Collection, Computer Science [Dig. Serial], Springer-11645 [Dig. Serial]. Springer Science+Business Media, LLC, Boston, MA, 2007. DOI:10.1007/978-0-387-38162-6.

[36] Meltdown and spectre. https://meltdownattack.com/.

[37] Planning and acting in partially observable stochastic domains. https://www.sciencedirect.com/science/article/pii/S000437029800023X?via%3Dihub.

[38] V. Mnih, K. Kavukcuoglu, D. Silver, et al. Human-level control through deep reinforcement learning. *Nature* **518**(7540):529–533, 2015. DOI:10.1038/nature14236.

[39] S. Griffith, K. Subramanian, J. Scholz, et al. Policy shaping: Integrating human feedback with reinforcement learning. In *In Advances in Neural Information Processing Systems.* 2013.

[40] A. Gonfalonieri. Inverse reinforcement learning – towards data science. https://towardsdatascience.com/inverse-reinforcement-learning-6453b7cdc90d.

[41] AlphaZero: Shedding new light on the grand games of chess, shogi and Go | DeepMind. https://deepmind.com/blog/alphazero-shedding-new-light-grand-games-chess-shogi-and-go/.