HoneyIoT: Adaptive High-Interaction Honeypot for IoT Devices Through Reinforcement Learning

Chongqi Guan
cxg5270@psu.edu
The Pennsylvania State University
University Park, PA, USA

Heting Liu
hxl476@psu.edu
The Pennsylvania State University
University Park, PA, USA

Guohong Cao
gxc27@psu.edu
The Pennsylvania State University
University Park, PA, USA

Sencun Zhu
sxz16@psu.edu
The Pennsylvania State University
University Park, PA, USA

Thomas La Porta
tfl12@psu.edu
The Pennsylvania State University
University Park, PA, USA

ABSTRACT

As IoT devices are becoming widely deployed, there exist many threats to IoT-based systems due to their inherent vulnerabilities. One effective approach to improving IoT security is to deploy IoT honeypot systems, which can collect attack information and reveal the methods and strategies used by attackers. However, building high-interaction IoT honeypots is challenging due to the heterogeneity of IoT devices. Vulnerabilities in IoT devices typically depend on specific device types or firmware versions, which encourages attackers to perform pre-attack checks to gather device information before launching attacks. Moreover, conventional honeypots are easily detected because their replying logic differs from that of the IoT devices they try to mimic. To address these problems, we develop an adaptive high-interaction honeypot for IoT devices, called *HoneyIoT*. We first build a real device based attack trace collection system to learn how attackers interact with IoT devices. We then model the attack behavior through markov decision process and leverage reinforcement learning techniques to learn the best responses to engage attackers based on the attack trace. We also use differential analysis techniques to mutate response values in some fields to generate high-fidelity responses. HoneyIoT has been deployed on the public Internet. Experimental results show that HoneyIoT can effectively bypass the pre-attack checks and mislead the attackers into uploading malware. Furthermore, HoneyIoT is covert against widely used reconnaissance and honeypot detection tools.

CCS CONCEPTS

• **Security and privacy**; • **Computing methodologies** → *Reinforcement learning*;

KEYWORDS

Honeypot, Internet of Things, Security, Reinforcement Learning

WiSec '23, May 29-June 1, 2023, Guildford, United Kingdom

ACM ISBN 978-1-4503-9859-6/23/05…$15.00
https://doi.org/10.1145/3558482.3590195

ACM Reference Format:
Chongqi Guan, Heting Liu, Guohong Cao, Sencun Zhu, and Thomas La Porta. 2023. HoneyIoT: Adaptive High-Interaction Honeypot for IoT Devices Through Reinforcement Learning . In *Proceedings of the 16th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec '23), May 29-June 1, 2023, Guildford, United Kingdom.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3558482.3590195

1 INTRODUCTION

Internet of Things (IoT) are being widely deployed in a variety of consumer, enterprise, and military settings such as home automation, smart manufacturing, autonomous driving, smart city, and military operations. However, there exist many threats to IoT devices due to their inherent vulnerabilities [1–4], which are caused by outdated or broken security modules, improper patch management, insufficient access control, and inadequate physical security. Many recent attacks [5, 6] utilize these security vulnerabilities to compromise and infect IoT devices. Therefore, there is a pressing need to understand the dynamic threat landscape for IoT devices in order to improve their overall security.

Honeypot is a valuable security tool that has been widely used by security practitioners to gain insight into the dynamic threat landscape. They are decoy systems designed to attract, engage and deceive potential attackers through vulnerable services in a monitored and controlled environment. Typically, they consist of virtual systems that closely mimic real production environments to effectively engage attackers. A successful honeypot can efficiently attract attackers through different vulnerabilities, evade reconnaissance and honeypot detection tools, provide seemingly genuine responses, and collect any attack traces left by the attackers for future analysis. As such, honeypots have the potential to be applied to improve IoT security.

Applying honeypot technology to the IoT domain presents significant challenges. This is primarily due to the heterogeneity of IoT devices, which means that vulnerabilities can vary significantly depending on the specific device brand, model, or firmware version. As a result, attackers typically perform various pre-attack checks to gather information about the target device before launching attacks. For instance, Nmap [7] is a widely used reconnaissance tool that can perform port scans and identify devices based on probing results. Additionally, attackers can use honeypot detection tools such as HoneyScore [8] or HoneypotHunter [9] to determine if a device is a honeypot during their pre-attack checks. Traditional

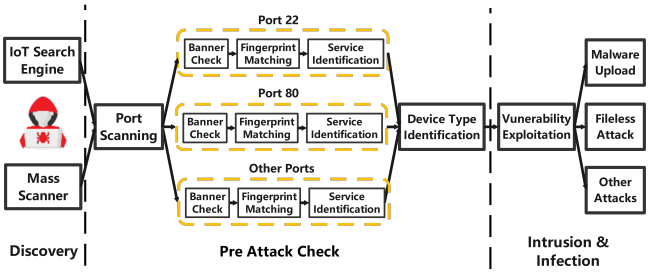


Figure 1: Attacks against IoT devices

IoT honeypots [10] are often limited in terms of interactions due to their fixed replying logic, which makes it difficult for them to deceive attackers during pre-attack checks.

To address these issues, we first need to learn how an attacker interacts with real IoT devices. In order to model the attacker behavior, we collect attack traces by exposing vulnerable IoT devices to attackers directly. The collected attack traces contain the attack behavioral information such as what IoT device the attacker is targeting, when they send specific pre-attack checks, and what responses can better lead to vulnerability exploitation or malware uploads. Since the attack trace is too complicated to extract meaning information purely based on heuristic based method or human analysis, we model the attack behavior through markov decision process (MDP) and leverage reinforcement learning techniques to learn the best responses to engage the attackers based on the collected trace. In addition, differential analysis techniques are used to mutate the response values in certain fields (e.g., date, time, sensor values, etc.) in order to generate more authentic and convincing responses.

The main contributions of this paper are as follows:

- We propose HoneyIoT, an adaptive high-interaction honeypot for IoT devices through reinforcement learning.
- We develop techniques to model the interactions between attackers and honeypot as a markov decision process base on the collected attack trace and leverage reinforcement learning to engage attackers.
- We identify the mutation fields in the response through differential analysis and update them in real time to provide high fidelity responses.
- We evaluate the effectiveness and robustness of HoneyIoT by deploying the system on the public Internet. Our evaluation results show that HoneyIoT is covert against widely used reconnaissance and honeypot detection tools, and it can mislead attackers into uploading malware.

2 BACKGROUND

In this section, we provide background information about the evolving threat landscape for IoT devices and explain why we need to build a more deceptive and interactive IoT honeypot.

IoT devices have long been valuable targets to attackers due to their inherent vulnerabilities [1–4]. For example, the storm of Mirai botnet [5] has overwhelmed several high-profile targets since late 2016. The original Mirai botnet compromises IoT devices through brute-force login against the telnet port. After a successful login, Mirai bots try to perform a series of operations to infect target devices and corral them into a botnet. The Mirai botnet was able to infect hundreds of thousands of IoT devices, which were used to perform DDos attacks against different targets. However, with the upgrading of IoT devices over recent years, simple brute-force login attacks against telnet ports are no longer effective. Some IoT device manufactures choose to use random default passwords to mitigate password cracking, and others choose to disable telnet and ssh services to avoid being discovered by the Mirai botnet.

Meanwhile, the attack against IoT devices is evolving. Instead of password cracking, attackers nowadays are targeting various types of vulnerabilities on IoT devices. For example, by exploiting the Remote Code Execution (RCE) vulnerabilities, attackers can inject malicious code to corral the target devices into their botnet. Since the vulnerabilities usually depend on the type, brand, and model of the target IoT devices, attackers nowadays tend to perform several pre-attack checks on the device before injecting malicious code to increase the attack success rate.

Figure 1 shows a typical attack process against IoT devices. The attacker first uses some reconnaissance tools such as Mass scan [11] or IoT search engine [12] to locate the victim device. Then, the attacker scans and probes the open ports of the target device to gather more information. The responses from the remote host can be used to match known fingerprints of existing honeypots [13–16]. For example, many open-source honeypots offer a limited set of hard-coded banners or static http responses which can be used as fingerprints to identify the remote hosts. During these pre-attack checks, if the attacker observes any honeypot fingerprint, or finds out inconsistency between the simulated device and the provided service, he will suspect that he is interacting with a honeypot. The attacker will either evade these honeypots by blacklisting their IP addresses, or takes down these honeypots through DDos attacks. The responses can also be used to identify the service running on the remote host, which can further be leveraged to pinpoint the type of the victim IoT device [17]. For example, if the port scan results indicate that the target is providing a video streaming service through real time streaming protocol (RTSP) on port 554, and a Web server for camera control on port 80, the target device is most likely an IoT camera. After identifying the remote host, the attacker can speculate the types of vulnerabilities existing on the target IoT device and then launch the exploitation attack. If the attack is successful, the attacker may launch various types of follow-up attacks such as uploading malware or paralyzing the device.

Honeypots have been used to defend against attacks on IoT devices. Conventional IoT honeypots mainly focus on emulating specific protocols such as telnet or ssh [10, 18]. The attacker may notice that certain service is missing and suspect that he is not interacting with a real IoT device. In addition, the honeypot only provides limited level of interaction to attackers with some fixed replying logic (i.e., fixed answer to various requests). Such behavioral fingerprints may have been recorded by the scanners [7, 11] used by the attackers to identify the honeypots. Therefore, conventional IoT honeypots are not effective against the latest attackers. There is a strong need to build adaptive high-interaction IoT honeypot which can interact with the attackers, bypass their pre-attack checks, and mislead them to upload their malicious codes.

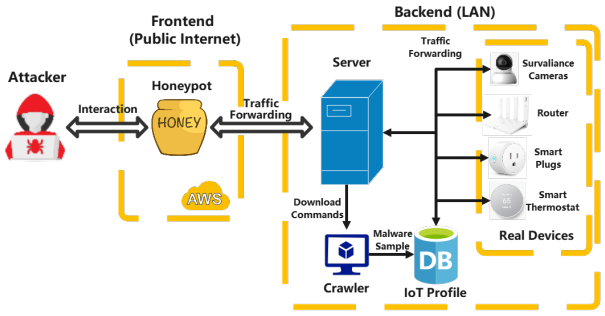


Figure 2: Attack trace collection based on real IoT devices

3 ATTACK TRACE COLLECTION

In this section, we present a system to collect attack traces, which can help us learn how the attackers interact with real IoT devices. We also show some analysis results based on the attack traces and generate attack graphs for IoT devices.

3.1 Real-Device Based Attack Trace Collection

We build a system to collect attack traces based on the interactions between attackers and real IoT devices. As shown in Figure 2, the system consists of a frontend virtual machine running on AWS, a backend server for traffic forwarding and preliminary traffic analysis, and a few IoT devices including various models of IoT cameras, routers and smart plugs. The detailed list of IoT devices and their corresponding vulnerabilities are shown in Table 1. The system interacts with the attacker by forwarding the received packet to one of the IoT devices to learn what the attacker will do next. Based on the attacker's request, the corresponding IoT device sends the necessary files or responses so that the attacker can continue to interact with the corresponding IoT device. This process continues until the attacker uploads some exploit code or stops interacting with the IoT devices. Our system maintains the log traces and may have to be rebooted in some cases to recover from the attacks. Then, a new cycle starts which may select a different IoT device for a different attacker. By doing this, our system can obtain different attacker traces, targeting different kinds of IoT devices, or targeting different kinds of protocols. We use the open source project SysFlow [19] to monitor the traffic between the attacker and the IoT devices and perform event-driven analysis to classify requests. We filter out any commands containing download instructions such as Wget or Curl, and forward them to a crawler in a sandbox to automatically collect malware from the attacker's control and command server.

3.2 Preliminary Analysis on Attack Traces

Our attack trace collection system was deployed on AWS from June 17 to September 1, 2022. During this period, the attack trace was collected to model the attack behavior and gain insights on how to effectively engage with attackers. An initial analysis of the trace indicates that it encompasses all known remotely exploitable CVEs associated with these IoT devices. In order to better understand the attacker behavior against IoT devices through the attack traces, we generate attack graphs for IoT devices based on the interactions between the attackers and the IoT devices. Figure 3 shows two attack graphs against TPlink NC220 cameras and Reolink cameras

Device Model	Manufacture	Device Type	Vulnerability ID
NC220	TP-Link	Camera	CVE-2020-12109, etc
RLC-410W	Reolink	Camera	CVE-2021-44402, etc
E1 Zoom	Reolink	Camera	CVE-2021-40149
Home	YI	Camera	CVE-2018-3928, etc
DS-2CD2183G	Hikvision	Camera	CVE-2021-36260
Insight	Wemo	Smart Plug	CVE-2018-6692
Mini	Wemo	Smart Plug	CVE-2018-6692
HS103-P4	TPlink	Smart Plug	CVE-2019-15745
ISP5	iHome	Smart Plug	RCE [1]
ISP6	iHome	Smart Plug	RCE [1]
VMB3000	Netgear	Router	CVE-2019-3949, etc
DGN2220	Netgear	Router	CVE-2020-35577, etc
TL-WR840N	TP-Link	Router	CVE-2018-14436, etc
DIR-3040	D-Link	Router	CVE-2021-21819
WS5200	Huawei	Router	CVE-2019-5268, etc
WS7200	Huawei	Router	N/A

Table 1: IoT devices used in our attack trace collection system

over HTTP ports. As the whole graph is way too big, we only show a partial attack graph emphasizing specific vulnerabilities exploited and the attack behavior.

In the attack graph, a node represents the attacker's action such as probing a directory, accessing a resource, exploiting a certain vulnerability or uploading a malware. For example, the node with '/' indicates that the attacker probes the root directory. The node with '/favicon.ico' indicates that the attacker tries to access the favicon file which is usually a small icon indicating the device type or manufacturer. The node with 'CVE-2020-12109' means that the attacker exploits a specific vulnerability with Common Vulnerabilities and Exposures (CVE) ID 2020-12109, i.e., a publicly disclosed IoT security flaw on TPlink web camera where no format check is enforced when the attacker sends malicious HTTP request through '/setsysname.fcgi'. The edges connecting two nodes indicate that some attackers have taken another action after receiving the previous response from the IoT device.

From the attack graph, we can see that the attacker conducts various types of pre-attack checks to gather information from the remote host before launching attacks. The attacker may choose different follow up attacks based on the responses from the IoT devices. For example, as shown in Figure 3(a), some of the attackers first access the favicon file to identify that this is a TPlink camera, by matching the MD5 hash of favicon or by analyzing its image. Then, they decide to exploit the 'CVE-2020-12109' vulnerability by sending various requests. On the other hand, if the attacker notices that the remote host is not a TPlink camera (the MD5 has does not match), he may not proceed with follow up attacks. As shown in Figure 3(b), the attack trace collected in the Reolink camera does not have such attack behavior because the favicon of Reolink camera is different from the favicon of TPlink camera.

The collected attack trace contains valuable attacker behavioral information which can be used to answer questions such as what IoT device the attacker is targeting, when the attacker sends specific pre-attack checks, and which response can better lead to a vulnerability exploitation or malware upload. The answers to these questions are critical to build high-interaction honeypots for IoT devices. In order to effectively learn the attack behavior through these collected attack traces, HoneyIoT leverages reinforcement learning techniques.

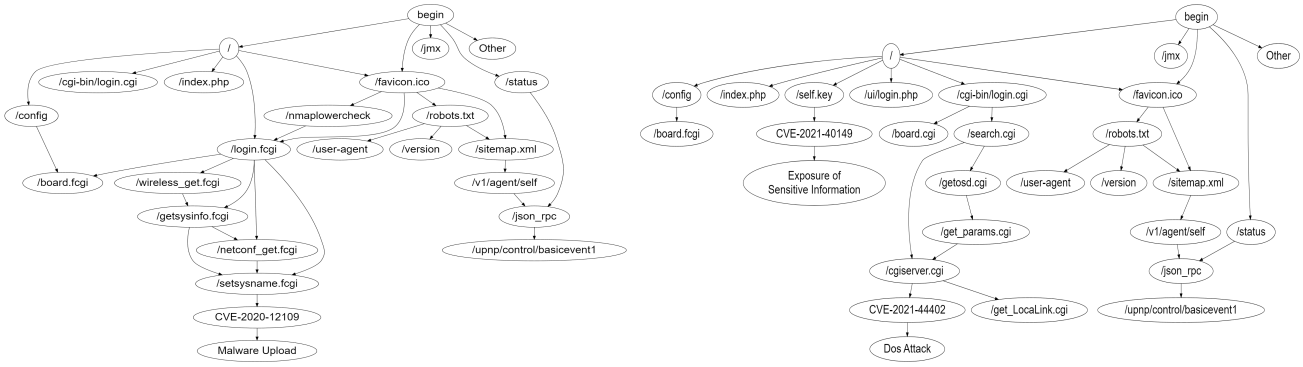


Figure 3: Partial Attack graph against HTTP port: (a) TPlink TC220 camera (b) Reolink Camera

4 ADAPTIVE HIGH-INTERACTION HONEYPOT FOR IOT DEVICES

In this section, we introduce HoneyIoT, an adaptive high-interaction honeypot for IoT devices. We first formulate the interactions between honeypots and attackers as a Markov Decision Process (MDP), and then leverage reinforcement learning to build an agent which can adaptively interact with the attacker by selecting proper responses. We also propose a differential analysis based content mutation method to effectively update some fields of the response at run time. At last, we present the general system design of HoneyIoT.

4.1 Problem Formulation

After taking a close look at the collected attack traces, we notice that there is a strong correlation between the attacker's requests and the responses from the IoT devices. As discussed in Section 3.2, some of the attacker's request packets look for the type and model information of the target device, and others try to collect the service and protocol information of that device. These packets together can be classified as "pre-attack checks", which are used to identify the target IoT device. Since attackers performing different pre-attack checks may be targeting different IoT devices and expecting different responses, our honeypot should adaptively choose the responses which can better mislead the attackers to perform follow-up attacks. In HoneyIoT, we formulate the interactions between the honeypot and the attacker as a MDP and leverage reinforcement learning algorithms to select the proper responses.

In a typical MDP, the agent interacts with the outside world called environment through a series of actions. At each step, the agent observes the environment state and takes an action. As a result of the action, the environment makes corresponding changes and transits to another state. Meanwhile, the agent may receive a reward generated by the environment. The agent will continue this process, and get accumulated rewards after every action until the session is over. The agent seeks to maximize the accumulated rewards by taking proper actions.

As shown in Figure 4, in our case, the honeypot is the agent that interacts with the environment containing different attackers. At each step, the honeypot agent observes the environment state by analyzing the attacker's request packet. The agent then takes an action to choose a proper response to fulfill the attacker's request.

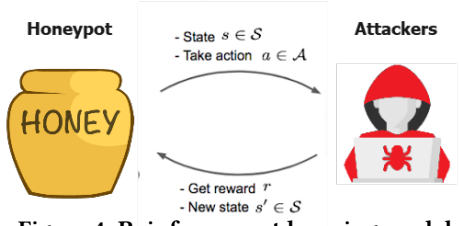


Figure 4: Reinforcement learning model

After receiving the response, the attacker may send another request that will cause the agent to enter a new state or terminate the session by stopping sending any requests. A reward is collected whenever the attacker tries to exploit a vulnerability, upload a malware or terminate the attack session. In the following subsection, we describe the state space, the action space, the state transition probabilities, and the reward function of our MDP model in detail.

4.2 MDP Model Formulation

State Space. The state of MDP model should represent the current situation of the reinforcement learning agent. In our case, it should contain the interactions between the attacker and the honeypot. In order to clearly represent the attacker's session history, we define the state as a series of packets received from the attackers. Let s_c denote the state space, and $s_c = H(p_1, p_2, \ldots, p_c)$, where s_c denotes the state space, and p_i denotes the i^{th} request packets received from the attacker. However, directly using the packets from the attack trace may lead to sparse state space since the same type of request packets from different attackers may only vary slightly. To address this problem, some states may be aggregated, i.e., request packets with the same path and query strings are represented by a single state. In addition, we manually add a terminating state if no new packet is received from the attacker after a certain amount of time to indicate the end of the session.

Action Space. Each response from the IoT device is labeled as a discrete action for the honeypot agent. That is, $a \in A = \{q_1, q_2, \cdots\}$, where A is the action space which is a set of all possible responses from IoT devices, and q_i denotes a specific response from certain IoT devices. Since some IoT devices may not be able to respond to certain requests as they do not provide certain services, only a subset of actions is available at a given state in our MDP model. Therefore, the action space for our MDP model is essentially a multi-discrete action space. When a specific action is taken by the

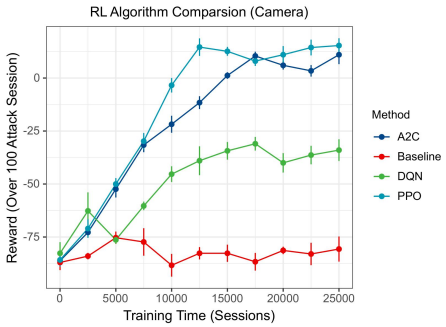


Figure 5: Comparison of different learning algorithms

honeypot agent, the corresponding response will be rewritten (i.e. the date value) and forwarded to the attacker in order to provide a real-time high-fidelity response. The details on packet modification will be discussed in Section 4.3.

State Transition Probabilities. In our model, the state transition probability (Equation 1) can be described as a transition function $T(s, a, s')$ where s is the current state of the environment, a is the action taken by the agent, and s' is the next state of the environment. In our case, the next state $s' = H(p_1, \ldots, p_c, p_{c+1})$ refers to the packet received after taking action a at the current state $s = H(p_1, \ldots, p_c)$. To better model the attacker's behavior, we utilize the attack trace collected through our system. We calculate the percentage of the occurrences of (s, a, s') over the occurrences of all combinations containing the current state s and action a. The percentage is then used as the state transition probability.

$$T(s, a, s') = P(S_t = s' \mid S_{t-1} = s, a_t = a) = \frac{C(s, a, s')}{\sum_{x \in S} C(s, a, x)} \quad (1)$$

Reward Function. Since the ultimate goal of our honeypot is to mislead the attacker to upload the malicious code, we should in general reward the responses that lead to a real attack while punishing those that cause the attacker to end the current session. Typically, an attacker will exploit a vulnerability before uploading malware, so we assign a moderate reward for vulnerability exploitation, while reserving the highest reward for uploading malicious code. The intermediate reward reflects the potential progress we make when selecting a response in a given state.

In our context, the reward function can be described as $R(s, a, s')$, where s is the current state of the environment, a is the action that our honeypot agent takes and s' is the next state of the environment. In particular, we set the intermediate reward $R(s, a, s')$ for each valid state transition to 0 unless the last packet in the next state s' is vulnerability exploitation, malware upload or terminating state. If the attacker terminates the session without sending any malware download command or exploiting any vulnerabilities, we assign a negative reward. If any vulnerabilities are exploited, we add a positive intermediate reward. If the attacker uploads malware, we assign a large positive reward to this session.

Algorithm Selection. In recent years, many reinforcement learning algorithms such as Proximal Policy Optimization (PPO) [20], Deep Q Network (DQN) [21] and Advantage Actor Critic (A2C) [22] have been proposed. In particular, PPO is a policy based method. A reinforcement learning policy is a mapping from the current environment observation to a probability distribution of the actions to be taken. In PPO, we build and update a policy to maximize the long-term reward while learning. DQN is a value based method where we store and update the value function for each state and action pair, and use the value function to deduce the best action at a given state. A2C is a hybrid of policy based method and value based method. It consists of a policy based actor which controls how the agent acts, and a value based critic that measures the effectiveness of the agent's action.

In order to choose the most suitable algorithm for HoneyIoT, we first turn the collected attack trace into the MDP model defined in section 4.2 and then use the Open AI Gym environment [23] and stable baseline3 [24] to test the effectiveness of different reinforcement learning algorithms. For comparison purposes, we also set up the baseline approach where no reinforcement learning algorithm is used. The baseline agent chooses each valid candidate response with equal probability. In our setting, a positive reward of 5 and 1 is given when the agent receives a malware upload or vulnerability exploitation in an attack session, and a negative reward of -1 is given otherwise. In order to minimize the variance, we evaluate the reward over 100 random attack sessions multiple times for different RL algorithms with different training times. The results in Figure 5 show that PPO has the fastest convergence rate and the best average reward compared to other algorithms. Therefore, HoneyIoT uses PPO as the learning algorithm.

4.3 Differential Analysis based Content Mutation against Fingerprinting Attacks

Although the reinforcement learning model can select a valid response when receiving the request, the attackers may still be able to detect the honeypot through fingerprinting attacks. Fingerprinting attacks [13–16] have long been an effective tool to detect and label honeypots. They have been widely applied in cyber attacks mainly because they can greatly improve the success rate of intrusion. In a typical fingerprinting attack, the attacker first collects information from remote hosts through active probing or passive eavesdropping. The attacker then uses some heuristic rules [17, 25] to match the responses with existing fingerprint database and analyzes the inconsistency. Most of the existing open-source honeypots are not effective against fingerprint attacks [14] due to their open nature. For example, Dionaeca can be identified through its fixed banner information, Cowrie can be detected through its error messages and Glastopf can be identified by analyzing its HTTP responses. Without proper content mutation, the attacker may identify the fixed replying logic of the remote host through fingerprinting attacks and detect the honeypot.

To address these fingerprinting attacks, we need to provide responses that are unique to different attackers and follow the internal logic of IoT devices. We need to modify the packet selected by the RL model before returning it to the attacker. However, it is a challenge to locate the field that needs to be updated and figure out how we should update these values. As IoT devices from different manufactures have different internal logic, generating responses by reverse engineering each IoT device can be time-consuming. To address this problem, we apply differential analysis based method to analyze the responses and extract their mutation fields and mutation logic.

Differential analysis has been applied to IoT security and privacy analysis domain as it can help researchers glean valuable insight by analyzing closely related inputs. For example, IotSpotter [26] uses differential analysis to identify IoT-specific libraries among different mobile apps. Continella *et al.* [27] proposed a privacy leak detection method for mobile apps by analyzing the difference among the network traces which are generated based on users' requests containing various private information (e.g. location). However, none of them has been applied to IoT honeypot design.

We identify the *mutation fields* by leveraging differential analysis techniques as follow. Based on the collected trace, we find responses generated by the same request (i.e., with the same request path and query string) to the same IoT device at different time. By comparing these responses, we find that some fields do not change (e.g., static web content) and some fields change (e.g., date, time, sensor readings). The fields that change are the *mutation fields*.

Since different attackers may send the same request to the same IoT device at different time, the response from the IoT device can be directly used for differential analysis. For some rare request that is only sent by few attackers, we replay the attack request offline towards the IoT device multiple times to collect enough responses for differential analysis. We then extract the responses and use the Needleman-Wunsch algorithm [28] to identify the mutation fields. These mutation fields can be classified into three categories based on the cause of the mutation:

- **Timing**: The mutation field is affected by the current time of the IoT device. These mutation fields are usually in HTTP headers, device diagnostic logs, or user interface.
- **System**: The mutation field is determined by the system and physical status of the IoT device such as the pan and tilt angle of an IoT camera, the temperature value of a thermostat, the switch status (on, off) of a smart plug, etc.
- **Random**: The mutation field does not have a determined value. These mutation fields may be session identifiers, encryption value, or random values caused by the software non-determinism.

For different categories of mutation fields, we have different rules on updating their values to provide high-fidelity responses. First, since the collected responses may be weeks before the real HoneyIoT deployment, we need to update the time related mutation fields to avoid inconsistency. In general, we generate some simple heuristic rules for time related requests to rewrite these mutation fields. For example, we replace the date field of the HTTP header with the current server time and change the time related themes in the user interface accordingly.

Second, for system category, since the physical or system status of the IoT device may change over time, we need to provide different system values to the attackers at different time. The system value should be bounded by the physical capability of the IoT device (e.g., pan and tilt angle of a camera) or the environment (e.g., temperature of a thermostat). Since the values generated by the IoT devices in the past are valid, we store these valid system values in a database. At run time, HoneyIoT will randomly select a system value from the database to rewrite the corresponding mutation field.

Third, for mutation fields that do not have a determined value, the attacker will not be able to detect the inconsistency as long as we replace it with a different random value. For HoneyIoT, we record the features (e.g., length, maximum, minimum) of these random values, and then generate random values based on these features for the corresponding mutation field.

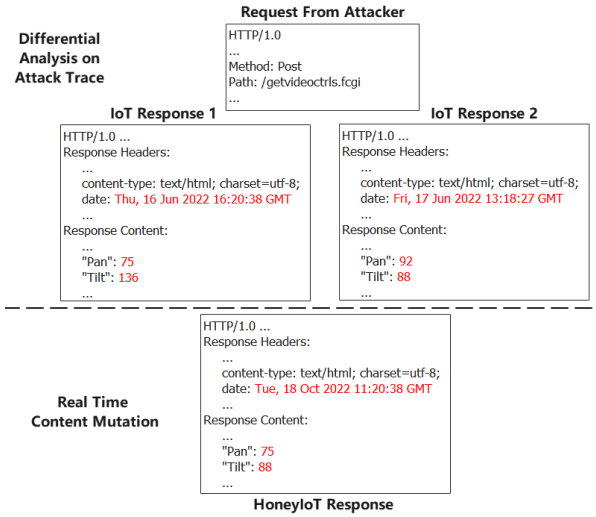


Figure 6: An example of using differential analysis to update the mutation field

Figure 6 shows how to update the mutation fields with a simple example based on attacks against an NC220 web camera. This "/getvideoctrls.fcgi" request allows the attacker to acquire video control related information, which is determined by the orientation of the camera. As shown in the figure, there are three mutation fields, time related mutation field "date", system related fields "Pan" and "Tilt". For the time related field "date", we rewrite it with the current time of the HoneyIoT server.The "Pan" and "Tilt" fields represent the horizontal and vertical angles of the IoT camera. As they are related to the physical orientation of the device, we classify them as the system category and store them in a database. At run time, HoneyIoT selects 75 for "Pan" and 88 for "Tilt".

By employing differential analysis based content mutation, HoneyIoT can effectively mitigate fingerprinting attacks and mislead attackers to launch followup attacks and upload malware. As future work, we aim to enhance the content mutation module by conducting more in-depth analysis of the correlations among different mutation fields, with the aim of generating more realistic and consistent responses that can better deceive evolving attackers.

4.4 System Design

As shown in Figure 7, HoneyIoT consists of two main components: the frontend and the reinforcement learning agent.

The frontend is a virtual machine which opens some ports to provide services similar to a real IoT device. It needs to parse and analyze attackers' request packets efficiently. Once the frontend receives an attacker's request packet, it extracts the crucial attributes of the data (i.e. source IP, target port, request path, query string, etc) and forwards them to the reinforcement learning agent. If any malware downloading command (e.g., wget) is detected, the frontend forwards the command to a separate crawler in a sandbox. The crawler will parse the obtained command and establish a separate outbound connection to the attackers' Command and Control

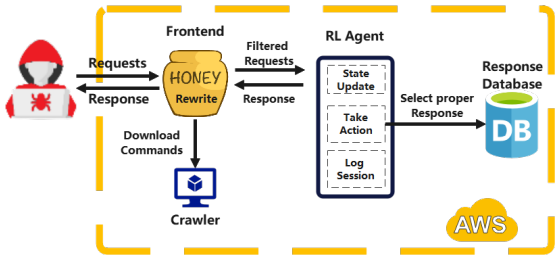


Figure 7: HoneyIoT system structure

(C&C) server and automatically collect malware for further analysis. The frontend is also responsible for assembling and rewriting the response selected by the reinforcement learning agent. As discussed in the previous section, HoneyIoT will use different logic to mutate the value based on the type of mutation field in the response. For system value, HoneyIoT randomly selects a valid value from the database to rewrite the field. For timing value, HoneyIoT combines the session information and heuristic rules to generate the mutation value. For mutation field that does not have a determined value, HoneyIoT uses some random number and string generator to rewrite the value. Some HTTP header information will also be updated in order to ensure the fidelity of the packet.

The reinforcement learning agent is responsible for emulating IoT devices by selecting a proper response based on the attacker's request. Upon receiving the attacker's request from the frontend, the reinforcement learning agent updates the state by appending the packet to the attacker's session history. It then takes an action by fetching a specific IoT device's response from the response database following the reinforcement learning algorithm and updates the reward based on the reward function. During the interaction process, HoneyIoT also logs all attack traffic for further analysis.

5 PERFORMANCE EVALUATIONS

In this section, we evaluate the performance of HoneyIoT and compare it to other IoT honeypots in various aspects.

5.1 Experiment Setup

We have implemented HoneyIoT and deployed it on the public Internet. Specifically, it runs on AWS using t2.small instance with 2G memory and one vCPU core. The frontend is mainly written in python and shell scripts. It opens ports identical to the real IoT devices it tries to emulate. For example, to emulate an IoT camera, the frontend opens the HTTP port 80, HTTPS port 443, RTSP port 554 and RTMP port 1935. We notice that different brands and models of IoT devices usually open some device specific ports providing services like mobile app control or firmware update. The frontend also opens these ports as there are attack traces collected over them.

Upon receiving a request from an attacker, HoneyIoT processes the request and forwards it to the RL Agent. After the RL Agent chooses a valid response, the frontend rewrites the corresponding mutation field based on their categories (in Section 4.3). We implement a separate crawler module to automatically collect malware from the attacker's control and command server once the frontend receives a downloading command. The malware sample is uploaded to VirusTotal[29] for malware classification and analysis.

In our implementation of the RL agent, we use stable baseline3 [24] to generate the RL model. As discussed in Section 4.2, we choose PPO as the RL algorithm in our model. We also implement a log module to store the interactions between the RL agent and the attackers for further analysis. The interaction log contains information such as the attacker's IP address, the request packets and reinforcement learning model's action at each step, etc.

We compare HoneyIoT with the following IoT honeypots:

Our Baseline: It is identical to HoneyIoT except that it does not use any RL algorithm. Instead, we implement a simple heuristic algorithm to let our baseline honeypot select responses based on the MDP model directly. In the algorithm, each response has an equal probability to be chosen regardless of the attacker's session history.

Existing Honeypot: We deploy an existing open-source honeypot called Snare & Tanner [30], because it can effectively mimic the Web service of IoT devices opened on HTTP ports. Since the majority of the vulnerabilities of the emulated IoT devices are over HTTP ports, Snare & Tanner can, at some degree, emulate these IoT devices.

5.2 Evaluation Results

We trained the HoneyIoT agent using our collected attack trace and then deployed HoneyIoT, Our Baseline, and Snare & Tanner on AWS for two month (Oct. 2022- Dec. 2023) to evaluate their performance. In particular, we build HoneyIoT-camera based on the attack trace of five different cameras (as shown in Table 1) and present the basic statistics in Section 5.2.1. Besides IoT camera, we also apply the idea of HoneyIoT to other IoT devices to test the extensibility of HoneyIoT. Specifically, we used the attack traces of six routers and five smart plugs to build HoneyIoT-router and HoneyIoT-smartplug and present their results in Section 5.2.2. In the remaining subsections, we evaluate the performance of HoneyIoT in terms of covertness and scalability.

5.2.1 Basic Statistics. As shown in Table 2, HoneyIoT-camera received 29,467 attack sessions, 2963 vulnerability exploitations, and 467 malware uploads from the attackers. Our baseline honeypot received 26,301 attack sessions, 843 vulnerability exploitations, and 92 malware uploads. Snare & Tanner received 28,660 attack sessions, 731 vulnerability exploitations and collected 14 malware. In order to estimate the capability of engaging attackers, we calculate the attack-session length by counting the request packets sent by attackers during each attack session. For better estimation, we excluded those sessions done by massive scanners which only perform a probe and then leave. The average session length is 7.57, 4.97, and 4.02 for HoneyIoT-camera, our baseline honeypot and Snare & Tanner, respectively. These results indicate that HoneyIoT is more effective in engaging and misleading the attackers to upload their malicious code compared to our baseline and the existing honeypot.

Figure 8 shows the vulnerability exploitation distribution of different honeypots. HoneyIoT-camera can successfully interact with the attackers and mislead them to exploit different vulnerabilities of all five IoT cameras, while our baseline honeypot and snare & tanner only mislead attackers to the vulnerabilities of Reolink E1 zoom, TPlink NC220 and Hikvision cameras. In fact, most of the attacks collected by our baseline honeypot and snare & tanner are

Table 2: Basic Statistics

	HoneyIoT-camera	Our Baseline	Existing Honeypot (Snare & Tanner)
Attack Sessions	29467	26301	28660
Average Session Length	7.57	4.97	4.02
Vulnerability Exploit	2963	843	731
Malware Collected (Total)	467	92	14

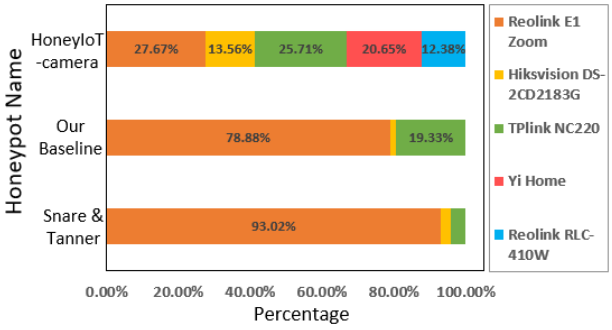


Figure 8: Vulnerability exploitation distribution

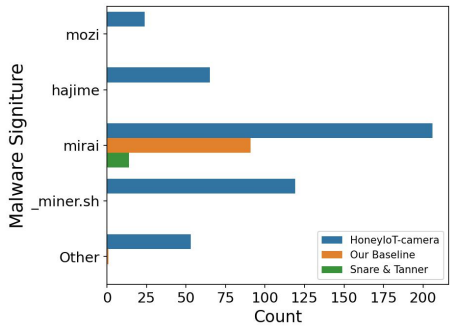


Figure 9: Malware uploaded

through a vulnerability with CVE ID 2021-40149 on the Reolink E1 Zoom camera. It is a simple exploitation where attackers can obtain the SSL private key of the camera by launching a directory traversal attack. As shown in Figure 3(b), compared to other sophisticated remote code execution attacks, this exploitation usually has a very short attack path and thus does not require pre-attack checks. These results indicate that HoneyIoT-camera can effectively emulate the vulnerabilities of different IoT cameras and mislead attackers.

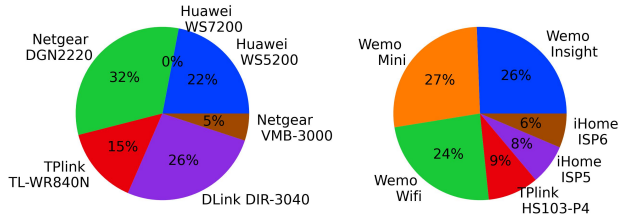
Figure 9 shows the malware collected by HoneyIoT-camera, our baseline honeypot, and snare & tanner, which are identified through a malware analysis website called VirusTotal [29]. Most of the malware collected by the HoneyIoT-camera can be classified as Mirai botnet and it's variants. In particular, HoneyIoT-camera collected 467 botnet malware samples, where 24 are classified as Mozi [6], 65 are classified as Hajime and 206 are classified as Mirai. Other than the botnet malware, HoneyIoT-camera also collected 119 miner malware that aim to mine cryptocurrency through the infected devices. Other 53 files collected from the attackers cannot be identified by VirusTotal, and thus we categorize them as "other" malware. Although our baseline honeypot caught 92 malware and Snare & Tanner caught 14 malware, they are mostly in the category of Mirai. These results demonstrate that HoneyIoT can effectively mislead different attackers to upload their malicious codes.

5.2.2 Extensibility. In order to evaluate whether HoneyIoT can be extended for emulating other types of IoT devices besides IoT cameras, we trained reinforcement learning models (called HoneyIoT-router and HoneyIoT-smartplug) using the attack trace of six routers and six smart plugs (as shown in Table 1) and deployed them on AWS. Within two months, the HoneyIoT-router received 40,531 attack sessions, 3675 vulnerability exploitation, collected 780 malware, and the average session length was 8.12. HoneyIoT-smartplug received 22,104 attack sessions, 1973 vulnerability exploitation, collected 186 malware, and the average session length was 6.97.

As shown in Figure 10(a), HoneyIoT-router successfully misleads attackers to exploit the vulnerabilities of all emulated routers except Huawei WS7200, probably because it has no publicly exposed vulnerability. After analyzing the log, we notice that the reinforcement learning model avoids using the response of Huawei WS7200 in most cases as it does not provide positive rewards. On the other hand, HoneyIoT-smartplug successfully misleads attackers to exploit the vulnerabilities of all six simulated smart plugs as shown in Figure 10(b),

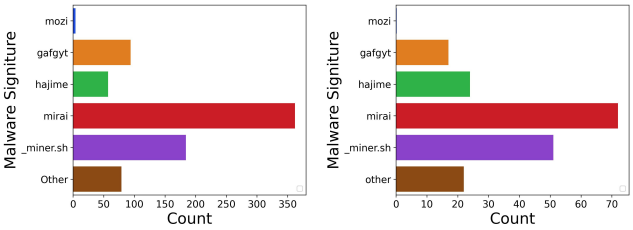
Figure 11(a) shows the malware collected by HoneyIoT-router. For the 780 collected malware, 4 are classified as mozi, 94 are classified as Gafgyt, 57 are classified as hajime, 362 are classified as Mirai, 184 are classified as miner, and 79 cannot be identified by VirusTotal. Figure 11(b) shows the malware collected by HoneyIoT-smartplug, where 17 are classified as Gafgyt, 24 are classified as hajime, 72 are classified as Mirai, 51 are classified as miner, and 22 cannot be identified by VirusTotal. Currently, HoneyIoT relies solely on VirusTotal to perform malware classification and analysis. Although most of the collected malware samples can be identified by VirusTotal, some files cannot be identified by it. Some of them are essentially empty files with randomly generated file names possibly due to the failure of attacker's control and command server. Other files may represent new malware (e.g., 0-day malware) that require further investigation. These results together demonstrate that HoneyIoT can emulate the vulnerabilities of different routers and smart plugs, and can collect various types of malware from attackers. In other words, HoneyIoT can be extended to effectively emulate other types of IoT devices.

We also deployed HoneyIoT-camera, HoneyIoT-router, and HoneyIoT -smartplug at three different locations (Virginia, Paris, Tokyo), and performed analysis on the collected malware based on the geographic locations. As shown in Figure 12, HoneyIoT placed in US collected most malware regardless of the device it emulates. In particular, HoneyIoT-camera placed in US, France, and Japan collected 467, 322 and 370 malware, HoneyIoT-router placed in the US, France, and Japan collected 780, 732 and 695 malware. HoneyIoT-smartplug placed in the US, France, and Japan collected 186, 147 and 162 malware. The performance difference among Honeypots deployed at



(a) HoneyIoT-router (b) HoneyIoT-smartplug

Figure 10: Vulnerability exploitation distribution



(a) HoneyIoT-router (b) HoneyIoT-smartplug

Figure 11: Malware collected

different locations is mainly due to the behavioral difference among different groups of attackers. The attackers can hand-craft their mass-scanners to restrict the scanning area with the goal of reducing the overall scanning time and targeting specific devices that are widely used in a given region.

In addition, we notice that the types of malware collected by these three types of honeypots have certain degree of overlaps. This phenomenon indicates that some malware is independent of the compromised IoT device type. In other words, some attackers essentially treat IoT devices as general light-weight linux machines. In this sense, for the purpose of malware collection, it is possible to obtain all types of popular malware over the public Internet just by emulating a potentially small subset of vulnerable IoT devices that are attractive to the attackers. In the future, we will further extend HoneyIoT to other types of IoT devices and try to locate this subset.

5.2.3 Covertness. In this subsection, we evaluate whether HoneyIoT is covert against widely used reconnaissance and honeypot detection tools. The Shodan honeyscore[12] is a well-known tool to check whether a remote host is a honeypot or not. Given an IP address, the Shodan honeyscore calculates the probability of the host to be a honeypot, in the range of 0.0 to 1.0, where 0.0 means that the host is definitively a real system, and 1.0 means that the host is a honeypot. The honeyscore is calculated by an undisclosed machine learning classification algorithm. According to Shodan, the honeyscore is affected by arguments such as the number of opened network ports, fingerprints of known honeypots, past records of the host IP address, and interactions with the host. Therefore, we are interested in finding out the capability of HoneyIoT in dealing with this state-of-the-art reconnaissance tool.

After the deployment of HoneyIoT on AWS, we wait for two weeks and then use Shodan API to acquire the honeyscores of HoneyIoT by providing their IP addresses. For comparison, we use T-pot [31], an all-in-one honeypot platform to deploy different open-source honeypots and collect their honeyscores. Specifically, Dionaea is a medium interaction honeypot that mimics multiple

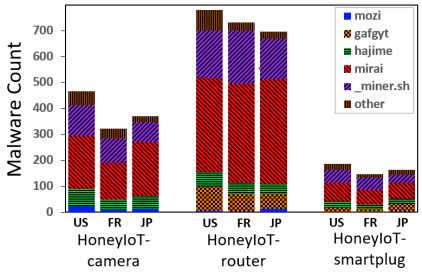


Figure 12: Malware collected based on geographic location

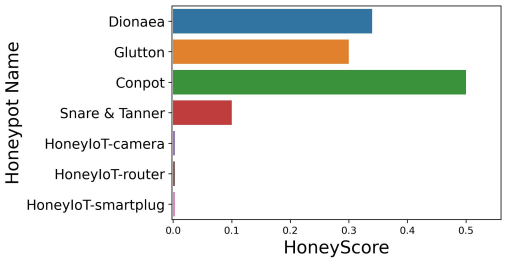


Figure 13: Honey Score for existing honeypots and HoneyIoT

service including ftp, upnp and mqtt, Glutton emulates vulnerable ssh servers, and Conpot is designed for emulating Industrial Control System (ICS). In particular, we configure T-pot so that only one specific open-source honeypot runs at one time. To reduce variation errors, we deployed multiple instances for each type of honeypot on different locations for two weeks before collecting the honeyscores of these honeypots via Shodan API.

As shown in Figure 13, HoneyIoT has honeyscore of 0 which indicates that Shodan treats HoneyIoT as a real IoT device. On the other hand, the existing open-source honeypots all have honeyscores higher than 0. Dionaea, Glastopf, Conpot and Snare & Tanner have honeyscores of 0.35, 0.3, 0.5 and 0.1. These results demonstrate that HoneyIoT is effective at maintaining covertness against state-of-the-art reconnaissance and honeypot detection tools.

5.2.4 Scalability. Compared to real-device based honeypot which requires physical IoT devices [32, 33] whenever the honeypot is active, HoneyIoT offers a more flexible approach. Physical devices are only required during the attack trace collection phase, after which HoneyIoT can model IoT devices using the collected attack trace and use the trained model to interact with the attacker. Training the reinforcement learning model is a time-consuming but one-time job. In our case, it took us approximately two hours to process the attack trace and train HoneyIoT-camera with our server, which has an AMD Ryzen 7 5800 CPU and an RTX 3090 GPU.

At run time, the scalability and deployment cost of HoneyIoT depends mainly on its resource consumption. To evaluate the resource consumption of HoneyIoT, we collect the average CPU and memory usage of HoneyIoT in our AWS instance. In our experiment, we use the AWS T2.small instance which has one vCPU core and 2G memory. According to AWS, each vCPU core of a T2 instance is essentially a thread of a 3.3 GHz Intel Xeon Scalable processor. For comparison, we also deploy several existing open-source honeypots including Dionaea, Conpot, and Snare & Tanner on AWS using T2.small instance and monitor the CPU and memory usage of each honeypot process.

Table 3: Scalability

Honeypot	Average CPU usage	Average Memory usage	AWS Instance
HoneyIoT-camera	4.2%	174 MB	T2.small
HoneyIoT-router	4.7%	187 MB	T2.small
HoneyIoT-smartplug	4.1%	157 MB	T2.small
Dionaea	23.6%	786 Mb	T2.small
Snare & Tanner	7.1%	282 Mb	T2.small
Conpot	17.7%	548 Mb	T2.small

As shown in Table 3, during the interactions with attackers, the average CPU usage of HoneyIoT-camera is 4.2% and the average memory usage is 174 MB. HoneyIoT-router and HoneyIoT-smartplug have similar average CPU and Memory usage. On the other hand, Snare & Tanner has an average CPU usage of 7.1% and an average memory usage of 282 MB. Dionaea has an average CPU usage of 23.6% and an average memory usage of 786 MB. Conpot has an average CPU usage of 17.7% and an average memory usage of 548 MB. Dionaea and Conpot have much higher computational overhead than HoneyIoT mainly because they are not specifically designed for IoT devices and they emulate functions and services that are not supported by IoT devices. These results show that HoneyIoT is lightweight and has good scalability.

6 RELATED WORK

There has been considerable research on using honeypots [34, 35] to deceive the attackers. Many open-source or commercial honeypots have been deployed, especially for computer network services such as honeyd [36] and nepenthes [37], etc. Recently, due to the wide adoption of IoT devices, IoT honeypots have also been developed to increase the overall security of IoT systems.

IoTPOT [10] is the first honeypot specifically designed for IoT devices. It is a low-interaction honeypot focusing on emulating telnet service which is used by many IoT devices. Hakim *et al.* [38] introduced U-POT, an IoT honeypot framework specifically designed for the UPnP (Universal Plug and Play) protocol which is widely used in smart home such as surveillance cameras, smart bulbs, and smart switches. It uses device description files to automate honeypots and provide fake responses. Seamus *et al.* [18] builds an SSH honeypot by implementing an interactive shell. They leverage reinforcement learning to conceal the honeypot characteristics. In HoneyCam [39], to emulate the video streaming services of IoT cameras, the authors propose to prerecord 360° video and map the 360° video to different fields of view based on the attacker's camera control commands. These IoT honeypots mainly focused on emulating specific protocols or services that are widely used by the IoT devices. However, as discussed in section 2, the attackers nowadays will perform various pre-attack checks to gather information for follow up attacks, and can notice that certain services are missing and suspect that they are not interacting with a real device.

As another approach, researchers leverage firmware images or real devices to build honeypots for IoT devices. Vetterl *et al.* proposed Honware [40], a virtual honeypot framework that can emulate different IoT devices based on their firmware image. HoneyCloud [41] utilizes the firmware image of IoT devices to generate hardware and software IoT honeypots in order to collect attacks against Linux-based IoT devices. Guarnizo *et al.* [32] proposed a high-interaction IoT honeypot architecture called SIPHON, which sets up honeypots on the cloud and leverages traffic forwarding technique to redirect attacker's commands back to IoT devices in their lab. Similarly, Tambe *et al.* [42] uses VPN to integrate the off-the-shelf IoT devices into a general honeypot architecture. Luo *et al.* [43] proposed IoTCandyJar, an intelligent-interaction honeypot that emulates the request-response pattern of IoT devices. IoTCandyJar first acquires the attacker's request packets using low-interaction honeypots and probes the IoT devices on the public Internet to collect valid responses for the requests. IoTCMal [33] uses real IoT devices to engage the attackers. It is a hybrid IoT honeypot which consists of a low interactive and a high interactive component. The low interactive component handles the login process and forwards the attacker's commands to the high interactive component that uses real IoT devices. Then, the attackers are essentially interacting with real IoT devices after the login phase. However, most of the IoT device manufacturers do not publicize the firmware images of their products. In addition, always forwarding the attacker traffic to real IoT devices suffers from scalability issues and cannot be deployed at a large scale.

Different from the aforementioned existing research, HoneyIoT does not limit itself to emulating specific protocol or specific IoT device. Moreover, through reinforcement learning, HoneyIoT can adaptively interact with the attackers and mislead them to upload malware.

7 CONCLUSIONS

In this paper, we proposed an adaptive high-interaction honeypot for IoT devices, called HoneyIoT, which can better engage attackers leveraging reinforcement learning techniques. To learn how attackers interact with IoT devices, we first build a system for collecting attack traces from real devices. We model the attack behavior using a Markov decision process and use reinforcement learning to determine the best responses to engage attackers based on the collected traces. We also use differential analysis techniques to mutate response values in some fields to mitigate fingerprinting attacks. HoneyIoT has been deployed on the public Internet, and experimental results demonstrate its effectiveness in evading reconnaissance and honeypot detection tools, as well as its ability to mislead attackers into uploading malware.

In the future, we will enhance HoneyIoT's deception capabilities by developing more sophisticated reward functions and generating responses that are more consistent and realistic. Additionally, we will extend HoneyIoT to emulate a wide variety of IoT devices and deploy them across various public cloud platforms and enterprise networks to collect more malware samples and analyze attacker behaviors. Our ultimate goal is to leverage the collected malware samples to detect zero-day attacks before they can be widely launched.

ACKNOWLEDGEMENT

This research was partially sponsored by the U.S. Army Combat Capabilities Development Command Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-13-2-0045 (ARL Cyber Security CRA).

REFERENCES

[1] N. Neshenko, E. Bou-Harb, J. Crichigno, G. Kaddoum, and N. Ghani. Demystifying IoT security: An Exhaustive Survey on IoT Vulnerabilities and a First Empirical Look on Internet-Scale IoT Exploitations. *IEEE Commun. Surveys & Tutorials*, April 2019.

[2] O. Alrawi, C. Lever, M. Antonakakis, and F. Monrose. SoK: Security Evaluation of Home-Based IoT Deployments. *IEEE Symp. on Security and Privacy*, May 2019.

[3] O. Alrawi, C. Lever, K. Valakuzhy, R. Court, K. Snow, F. Monrose, and M. Antonakakis. The Circle Of Life: A Large-Scale Study of The IoT Malware Lifecycle. *USENIX Security Symp.*, 2021.

[4] M. Ozmen, X. Li, A. Chu, Z. Celik, and X. Zhang B. Hoxha. Discovering IoT Physical Channel Vulnerabilities. *ACM CCS*, 2022.

[5] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, et al. Understanding the Mirai Botnet. *USENIX Security Symp.*, 2017.

[6] Mozi Botnet Accounts for Majority of IoT Traffic. https://threatpost.com/mozi-botnet-majority-iot-traffic/159337/.

[7] Nmap: Open-source network scanner. https://nmap.org/.

[8] Shodan Honeyscore. https://honeyscore.shodan.io/.

[9] Send-Safe Honeypot Hunter. http://www.send-safe.com/honeypot-hunter.html.

[10] Y. Pa, S. Suzuki, K. Yoshioka, T. Matsumoto, T. Kasama, and C. Rossow. IoTPOT: Analysing the Rise of IoT Compromises. *USENIX Workshop on Offensive Technol.*, 2015.

[11] Masscan. https://github.com/robertdavidgraham/masscan.

[12] Shodan: Search Engine for the Internet of Everything . https://www.shodan.io/.

[13] J. Franco, A. Aris, B. Canberk, and S. Uluagac. A Survey of Honeypots and Honeynets for Internet of Things, Industrial Internet of Things, and Cyber-Physical Systems. *IEEE Commun. Surveys & Tutorials*, August 2021.

[14] S. Morishita, T. Hoizumi, W. Ueno, R. Tanabe, C. Gañán, M. van Eeten, K. Yoshioka, and T. Matsumoto. Detect Me If You… Oh Wait. An Internet-Wide View of Self-Revealing Honeypots. *IFIP/IEEE Symp. on Integrated Network and Service Management*, 2019.

[15] A. Vetterl, and R. Clayton. Bitter Harvest: Systematically Fingerprinting Low- and Medium-interaction Honeypots at Internet Scale. *USENIX Workshop on Offensive Technol.*, 2018.

[16] M. Zamiri-Gourabi, Ali R. Qalaei, and B. Azad. Gas What? I Can See Your GasPots. Studying the Fingerprintability of ICS Honeypots in the Wild. *Annual Industrial Control System Security Workshop*, 2019.

[17] X. Feng, Q. Li, H. Wang, and L. Sun. Acquisitional Rule-based Engine for Discovering Internet-of-Things Devices. *USENIX Security Symp.*, 2018.

[18] D. Seamus, M. Schukat, and E. Barrett. Using Reinforcement Learning to Conceal Honeypot Functionality. *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, 2018.

[19] F. Araujo and T. Taylor. SysFlow: Scalable System Telemetry for Improved Security Analytics. *IEEE International Conference on Big Data*, 2020.

[20] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal Policy Optimization Algorithms. *CoRR*, abs/1707.06347, Mon 2017.

[21] V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, Feb 2015.

[22] V. Mnih, A. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous Methods for Deep Reinforcement Learning. *International Conference on Machine Learning (ICML)*, 2016.

[23] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[24] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann. Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research*, Nov 2021.

[25] S. Marchal, M. Miettinen, T. Nguyen, A. Sadeghi, and N. Asokan. AuDI: Toward Autonomous IoT Device-Type Identification Using Periodic Communication. *IEEE Journal on Selected Areas in Commun. (JSAC)*, March 2019.

[26] X. Jin, S. Manandhar, K. Kafle, Z. Lin, and A. Nadkarni. Understanding IoT Security from a Market-Scale Perspective. *ACM CCS*, 2022.

[27] A. Continella, Y. Fratantonio, M. Lindorfer, A. Puccetti, A. Zand, C. Kruegel, and G. Vigna. Obfuscation-Resilient Privacy Leak Detection for Mobile Apps Through Differential Analysis. *NDSS*, 2017.

[28] S. Needleman and C. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, Mar 1970.

[29] VirusTotal. https://www.virustotal.com/.

[30] Snare: Super Next generation Advanced reactive honeypot. https://github.com/mushorg/snare.

[31] T-Pot - The All In One Honeypot Platform. https://github.com/telekom-security/tpotce.

[32] J. Guarnizo, A. Tambe, S Bhunia, M. Ochoa, N. Tippenhauer, A. Shabtai, and Y. Elovici. Siphon: Towards Scalable High-Interaction Physical Honeypots. *ACM Workshop on Cyber-Physical System Security (CPSS)*, 2017.

[33] B. Wang, Y. Dou, Y. Sang, Y. Zhang, and J. Huang. IoTCMal: Towards a Hybrid IoT Honeypot for Capturing and Analyzing Malware. In *IEEE Int'l Conf. on Commun. (ICC)*, 2020.

[34] S. Kyung, W. Han, N. Tiwari, V. Dixit, L. Srinivas, Z. Zhao, A. Doupé, and G. Ahn. HoneyProxy: Design and implementation of next-generation honeynet via SDN. *IEEE Conf. on Commun. and Network Security (CNS)*, 2017.

[35] A. Zarca, J. Bernabe, A. Skarmeta, and J. Calero. Virtual IoT HoneyNets to mitigate cyberattacks in SDN/NFV-enabled IoT networks. *IEEE Journal on Selected Areas in Commun.*, April 2020.

[36] N. Provos. Honeyd: A virtual honeypot framework. *USENIX Security Symp.*, 2004.

[37] P. Baecher, M. Koetter, T. Holz, M. Dornseif, and F. Freiling. The nepenthes platform: An efficient approach to collect malware. *USENIX RAID*, 2006.

[38] M. Hakim, H. Aksu, A. Uluagac, and K. Akkaya. U-PoT: A Honeypot Framework for UPnP-Based IoT Devices. *IEEE Int'l Performance Computing and Commun. Conf. (IPCCC)*, 2018.

[39] C. Guan, X. Chen, G. Cao, S. Zhu, and T. La Porta. HoneyCam: Scalable High-Interaction Honeypot for IoT Cameras Based on 360-Degree Video. *IEEE Conf. on Commun. and Network Security (CNS)*, 2022.

[40] A. Vetterl, and R. Clayton. Honware: A Virtual Honeypot Framework for Capturing CPE and IoT Zero Days. *APWG Symp. on Electronic Crime Research*, 2019.

[41] F. Dang, Z. Li, Y. Liu, E. Zhai, Q. Chen, T. Xu, Y. Chen, and J. Yang. Understanding Fileless Attacks on Linux-Based IoT Devices with HoneyCloud. *ACM MobiSys*, 2019.

[42] A. Tambe, Y. Aung, R. Sridharan, M. Ochoa, N. Tippenhauer, A. Shabtai, and Y. Elovici. Detection of Threats to IoT Devices Using Scalable VPN-Forwarded Honeypots. *ACM Conference on Data and Application Security and Privacy (CODASPY)*, 2019.

[43] T. Luo, Z. Xu, X. Jin, Y. Jia, and X. Ouyang. IoTCandyJar: Towards an Intelligent-Interaction Honeypot for IoT Devices. *Black Hat*, 2017.