IoTCandyJar: Towards an Intelligent-Interaction Honeypot for IoT Devices

Tongbo Luo, Zhaoyan Xu, Xing Jin, Yanhui Jia, Xin Ouyang
Email: {tluo,zhxu,xijin,yjia,xouyang}@paloaltonetworks.com
Palo Alto Networks Inc.

ABSTRACT

In recent years, the emerging Internet-of-Things (IoT) has led to concerns about the security of networked embedded devices. There is a strong need to develop suitable and cost-efficient methods to find vulnerabilities in IoT devices - in order to address them before attackers take advantage of them. In traditional IT security, honeypots are commonly used to understand the dynamic threat landscape without exposing critical assets. In previous BlackHat conferences, conventional honeypot technology has been discussed multiple times. In this work, we focus on the adaptation of honeypots for improving the security of IoTs, and argue why we need to have a huge innovation to build honeypot for IoT devices.

Due to the heterogeneity of IoT devices, manually crafting the low-interaction honeypot is not affordable; on the other hand, purchasing all of physical IoT devices to build high-interaction honeypot is not affordable. This dilemma forced us to seek an innovative way to build honeypot for IoT devices. We propose using machine learning technology to automatically learn behavioral knowledge of IoT devices and build "intelligent-interaction" honeypot. We also leverage multiple machine learning techniques to improve the quality and quantity.

1. INTRODUCTION

In recent years, the emerging Internet-of-Things (IoT) has led to rising concerns about the security of network-connected devices. Different from conventional personal computer, such IoT devices usually open network ports to permit interaction between the physical and virtual worlds. In 2020, the number of interconnected devices will grow from 5 billion to 24 billion, attracting 6 trillion investment in various domains and applications like healthcare, transportation, public services and electronics [5]. The well-known IoT device exploration website, Shodan [22], has shown that millions IoT devices are exposed in Internet without proper protection. Therefore, finding vulnerabilities on IoT devices become the frontline of battle between white and black.

Honeypot is one of the common methods to discover 0-day vulnerabilities that widely used by security practitioner. In general, honeypot mimics interaction in real fashion and encourages unsolicited connections to perform attacks. Even though Honeypot is a passive approach, it can still efficiently find zero-day exploit attempt at the early stage of massive attack. There are many commercial honeypot products available, and more than 1000 honeypot projects on Github. However, we find out that the majority of the honeypot for IoT devices are low-interaction with fixed replying logic and limited level of interaction.

On the other hand, vulnerabilities on IoT device are usually highly depend on specific device brand or even firmware version. This leads to the fact that attackers tend to perform several checks on the remote host to gather more device information before launching the exploit-code. It turns out that such a limited level of interaction for existing honeypot projects is not enough to pass the check and fail to capture the real attack. Although malware for IoT device is relatively simpler than traditional ones, without properly handling the responses, the effectiveness of the IoT honeypot will be compromised.

In this paper, we present a method to build IoT honeypot in a automatic and intelligent way, named *intelligent-interaction*. Utilizing the public available IoT devices on the internet to gather the potential responses for the requests captured by our honeypot, we are able to obtain behaviors of different type of IoT devices. However, to pass attacker's checks, we also need to learn the best response which has a higher probability to be the expected one for attackers. We leverage multiple heuristics and machine learning mechanisms to customize the scanning procedure and improve the replying logic to extend the session with higher chance to capture the exploit code.

The rest of the paper is organized as follows: Section 2 gives a brief overview of honeypot and our motivation for building intelligent-interaction IoT honeypot. Section 3 explains how we customize the scanning module, *IoTScanner*, to collect raw behavior knowledge from the internet. Section 4 talks about our method to cluster IoT responses and generate *IoT-ID* to pinpoint IoT devices accuracy. Section 5 discusses how we leverage machine learning techniques to improve the reply logic. Evaluation and our interested findings from the captured traffic are presented in Section 6.

2. BACKGROUND AND MOTIVATION

In this section, we discuss a novel way to simulate the behavior of IoT devices to build an intelligent-interaction honeypot. The dilemma we are facing is that neither low nor high-interaction method can be used to build honeypot for IoT devices. Our honeypot can achieve the high coverage (the advantage of low-interaction honeypots), and behavioral fidelity (the advantage of high-interaction honeypots) at the same time. Since our honeypot only simulate the behaviors of the IoT devices, the requests and code sent

from the attackers to our honeypot would be processed as the real device. Therefore, unlike the high-interaction honeypots, there is no risk for our honeypot to be compromised.

2.1 Conventional Honeypot

In the honeypot research area, there are two categories of honeypots: *high* and *low* interaction. Low-interaction honeypots are nothing more than an emulated service and give the attacker a very limited level of interaction, such as a popular one called honeyd [18]; High interaction honeypots are fully fledged operating systems and use real systems for attackers to interact with. A good survey paper [4] revisited all of the honeypot research projects since 2005.

Both Low and High interaction honeypots have pro and cons. Low-interaction honeypots are limited and easily detectable; High-interaction honeypots are commonly more complex, furthermore deployment and maintenance often takes more time. In addition, more risks are involved when deploying high-interaction honeypots since an attacker can get complete control of the honeypot and abuse it, e.g., to attack other systems on the Internet. Thus it is necessary to introduce and implement data control mechanisms to prevent the abuse of honeypots. This is usually done using very risky and resource intensive techniques like full system emulators or rootkit-type software as in the GenIII honeynet [2].

Efforts on Automated Building High-Interactive Honeypot. Automatically building an interaction system for the honeypot have been studied by [7, 13, 14]. They investigate how to generate responses for certain request in a protocol-independent way, involving traffic clustering, building state machine and simplifying states. However, the major difference distinguish our work from the prior research is that all of prior projects rely on a large dataset for a specific protocol that was captured from live traffic. With this dataset as the ground truth, they extract the common structure from the traffic as the template, and generate the random data to fill the template. On the other hand, due to the various of customized protocol for IoT devices, it is hard to find such a clean and complete traffic dataset. Moreover, the live traffic contains only small portion of malicious traffic and it is hard to identify them from the dataset. Since the honeypot only need to simulate the behaviors that attackers are interested in, which can lead to the vulnerability. There is no need to learn all the behaviors, but it is important to learn the critical ones, which are highly possible to be missed from the live traffic.

Challenges for IoT Honeypot. Honeypot is not a new topic. Lots of honeypot frameworks are available (e.g honeyd [18], GenIII honeynet [2], and nepenthes [1]) either open-sourced or commercialized. Why do we want to talk about building a honeypot on IoT devices? Short answer is that IoT honeypot cannot be built on conventional honeypot technology. The `heterogeneity` feature of IoT devices makes the development of low-interaction IoT honeypot very time-consuming; and the price of real device as well as the `lack of emulators` makes it impossible to build high-interaction IoT honeypot.

We have to explain it with our journey on creating a prototype honeypot for IoT devices. The very straightforward way is to search for open-sourced IoT honeypot, and we did find a lot of them, such as IoTPot [17], SIPHON [9] and etc. Nevertheless, all of them are low-interaction honeypot like "Honeyd" [18], which is nothing more than a emulated service and give the attacker a very limited level of interaction. Obviously, those low-interaction honeypots can only get limited information for us. Due to the heterogeneity of IoT devices, it is challenging to mimic the interaction of different types of IoT devices from different vendors. Although not impossible, this requires a significant amount of technical work that cannot be easily reused. For example, consider the case of IP cameras, in order to visualize or simulate their behavior in a realistic way, one would need to not only broadcast some video to an attacker, but also react faithfully to commands such as tilting the camera.

Since low-interaction honeypot fails to satisfy our need, we turn into another direction, building high-interaction honeypot, which is impossible neither. There are two ways to build a high-interaction honeypot: physically or virtually. In the traditional definition, a physical honeypot is a real machine on the network with its own IP address. In IoT's context, it means we need to purchase real IoT devices for different brands and different types, and connect them to the internet. This solution is not practical due to the limited spaces and financial restrictions, not to mention the risks to introduce and implement data control mechanisms to prevent the abuse of honeypots. On the other hand, a virtual honeypot is a software that emulates a vulnerable system or network. But the fact is that, unlike operating systems (e.g. Android, Windows), the majority of IoT devices do not have any emulators available.

2.2 Intelligent-Interaction Honeypot

To conquer the challenges, we propose a generic framework toward building an *intelligent-interaction* honeypot for IoT devices. We will explain how and why do we need intelligent-interaction honeypot.

What is Intelligent-Interaction? The goal of intelligent-interaction is to learn the 'correct' behaviors to interact with clients from *zero-knowledge* about IoT devices. The correct responses to the clients should be able to extend the session with potential attackers, trick them to pass the check and send the exploit request. In order to achieve this goal, it requires our system to automatically collect the valid responses as candidates, By interacting with attackers, the learning process helps the honeypot to optimize the correct behaviors for each request.

How Intelligent-Interaction works? Figure 1 illustrates the overview of our system. There are 4 major components running separately but sharing the data to each other during the learning process. `IoT-Oracle` is a central

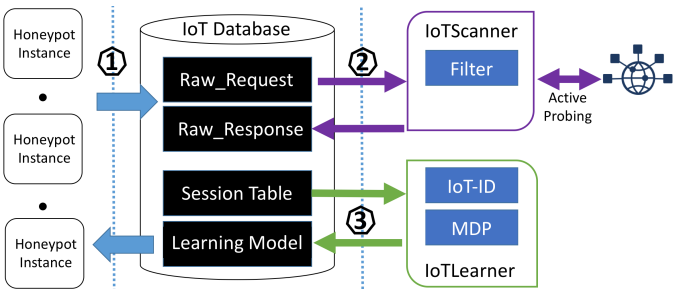

Figure 1 IoTCandyJar High-Level Architecture.

database that stored every information we obtained regarding to IoT devices. `Honeypot` module contains the honeypot instances we deployed on Amazon AWS and Digital Ocean. The purpose of them is to receive the traffic of attack and interact with attackers to allure them perform the real exploitation. They will periodically synchronize with the IoT-Oracle to push newly received raw request to the table *raw_request*, and retrieve the *iot_knowledge* table for up-to-date knowledges of IoT devices.

The module, `IoTScanner`, leverages captured attack's requests as the seed knowledge, and scans the internet for any IoT devices that can respond to these requests. The collected responses will be stored in the table *raw_response* for further analysis. The module, `IoTLearner`, utilizes machine learning algorithm to train a model based on the feedback from attackers with given response. After several round of learning iterations, our honeypot can optimize a model to reply to attackers.

At the very first moment, our system is behaved exactly like low-interactive honeypot since our system starts from zero-knowledge about IoT devices and their behaviors. We have evaluated that given a very short period of time, the honeypot can cover a lot of IoT devices.

Is Simulation Enough? For high-interaction honeypots, they usually deploy the real system or emulator in the virtual machine to react to attackers (e.g. responding to requests or executing uploaded/injected script). Our honeypot generates the response purely based on the learned knowledge, but not running it. Due to the attack surface of IoT devices, most of the attacks are launched using HTTP request and other IoT related protocol, and ultimately try to inject commands without authentication or get login credentials. Injected commands are very simple and concise, and usually are composed by a *wget* command to drop a shell (busybox) code from the malicious server to the device, assign permission to it and execute it. Detail of attacks and injected commands can be found in section 3.4. Our goal is to capture the injected script and extract the malicious shell code from it. Therefore, the interaction with attackers is not complicated and just reply a response to them. As a result, simulating the behavior of IoT devices is enough to build an effective honeypot.

Why Intelligent-interaction? As we just discussed in the previous paragraphs, contemporary attacks on IoT devices tend to be simple and straightforward and not hard to catch them using honeypot. However, for the most of attacks, attackers usually did some check on the target devices to know the device is vulnerable or not. Using the previous example (CVE-2016-6433) on Cisco Firepower, before sending the exploit request, attackers may check whether the device is Cisco Firepower and the version is 6.0.1 or not. This can be done by sending a request to *{ip}:443/img/favicon.png?v=6.0.1-1213* and checking the response status is 200 or not. And attackers may further try to login with the some credential. If any of these steps failed, attackers will stop attacks and our honeypot may not be able to capture the real exploits.

3. IOT-SCANNER: ACTIVE PROBING IOT BEHAVIORS

The first step to build an intelligent-interaction honeypot is to collect responses from all types of IoT devices. Fortunately, from the internet, we can find all of the IoT devices that are accessible. Therefore, we design and implement a module, *IoT-Scanner*, to actively probe the IoT devices on the internet and collect their responses to various of requests we have captured from the honeypot. Scanned result will be stored in the central database as the our 'raw' knowledge for further learning procedure.

Importantly, we want our probing to be polite and prevent unwanted traffic to Internet. The details of our probing can be found in our previous work [24]. To make the scanning process effective and not illegal, we have adopted a variety of *filtering* to narrow down the scope of remote host, make the traffic more IoT-related and eliminate the harmful requests.

3.1 IP Filtering.

Comparing to the 4.3 billion IPv4 address space, the number of IoT device is still tiny. Collecting a subset of IP address for IoT devices not only increase the quality of our scanning result but also speed up the scanning process. To our best knowledge, we cannot find a reliable and complete IP address set for IoT devices. Therefore, we build our own IoT-IP database from the scratch.

Device Type	Vender	Count
IP-Camera	Hikvision	8,785
	Avtech	4,391
	Dahua	4,002
	NetWave	3,713
	Kucam	1,302
	Tenvis	202
	Unknown	892
Router	TP-Link	4,560
	Linksys	3,604
	Netgear	2,461
	Sky	2,186
	BuffaloTech	235
	ZyXEL	1,232
Printer	HP	3,200
	Epson	2603
	Canon	1,989
	Brother	1,230
Smart Router	Linksys	1,581
	Unknown	330
Firewall	Huawei	783
	Fortinet	623
	Cisco	525
	SonicWall	553
	3com	197
	Juniper	30
Voip Gateway	D-Link	6,369
	Innovaphone	3,598
	AddPac	1,671
	Technicolor	959
	Edgewater	100
ONT	Alcatel Lucent	1,263

Table 1 IP collection of IoT devices.

We fetch the raw IP information from either online platforms, such as Censys [6], ZoomEye [25] and Shodan [22], or our own deployed port-scanning tool (e.g. MASSCAN [15]). We use the port-scanning tool to collect the basic information about a given IP address, like open ports in the remote machine, and the banner information of that open

port. Similar information is collected by these online platforms as well, but they provide query tools to search through their database more conveniently, which (1) Given an port number, what are IP addresses open the port? (2) Given a keyword, what are IP addresses serve content containing the keyword?

Currently, the widely adopted way is to use the different types of `banner` information to determine whether the machine behinds the given IP address is an IoT device. For example, the `Telnet` banner information can be used to identify the device type. It has been strongly recommended that computers display a banner before allowing users to log in since the publication of the "Computer Misuse Act 1990". Login banners are the best way to notify offenders before their unauthorized access.

Hence, we continuously conduct two searches on these platforms: (1) Search Port number (2) Search keywords, i.e, brand name, for IoT devices. We periodically store and update these information in our databases. First, we periodically search information through existing open platforms. Before using these information, we conduct another around of sync scanning to verify the port is indeed open. If so, we will treat these IPs as the higher priority and probe them to collect possible responses. So far, we have collected more than 40,000 IP address for IoT devices, as Table 1 shows. Apart from the feed from open platform, we also conduct Internet-wide probing if our target port is unique for IoT devices.

3.2 Port Filtering.

Among the 65535 ports, IoT devices may only listen a small portion of them for interaction. One of the most popular one is port 7547 for *TR-069* service, which is SOAP-based protocol for remote management of end-user devices. It is commonly used by IoT devices such as modems, gateways, routers, VoIP phones and set-top boxes. Another example is port 1900 for Universal Plug and Play (UPnP) protocol, and 67% of routers open this port to facilitate devices and programs to discovery routers and their configuration accordingly. For the IoT devices providing remote configuration through an embedded web server, they usually expose the port like 80, 8080, 81 and etc. We also monitor and scan the ports used by the protocols that are heavily utilized by IoT device, including port 5222 for XMPP, port 5683 for CoAP, and port 1883/8883 for MQTT. Table 2 highlights the port list we have identified from our analysis and prior survey [16], and we will prioritize to scan the traffic on these ports first.

Device Type	Open Ports
IP-Camera	81(35%), 554(20%), 82(10%), 37777(10%), 49152, 443, 83, 84, 143, 88
Routers	1900(67%), 21(16%), 80(1%), 8080, 1080, 9000, 8888, 8000,49152,81,8081,8443, 9090,8088,88,82,11,9999,22,23,7547
Printers	80(42%), 631(20%), 21(13%), 443(7%), 23,8080,137,445,25,10000
Firewall	8080, 80, 443, 81, 4433, 8888, 4443, 8443
ONT	8080, 8023, 4567
Misc	5222 (XMPP), 5683 (CoAP), 1883/8883 (MQTT),

Table 2 Ports used by IoT devices.

3.3 Seed Requests Filtering.

One of the critical input to our scanner is the requests captured by the honeypot. Our goal is to learn how IoT device reacts to each of them in order to simulate the behaviors. In our database, we stored around `18 million` raw requests in total in the past several months. It is not feasible and efficient to scan all of them, and we clean up the raw request database to eliminate the traffic that is clearly not IoT-related. For example, nearly half (53%) of the captured requests that not contain payload. Other none-IoT traffic including the ones of *BitTorrent* protocol (7%), MS-RDP protocol (5%) and SIP protocol (4%). It is important to notice two types of special traffic among the HTTP traffic: the HTTP proxy traffic (6%) that are redirected by proxy agents (usually looks like "*GET http://full-url HTTP/1.1*") and the scanning traffic (1%) for root path only. For the UDP traffics, we also identified the majority of them (6%) is some shell code such as the command using busybox. By applying the heuristics we explained above, we successfully reduce the total number of raw requests for scanning from 18 million to less than `1 million` (as Figure 2 shows).

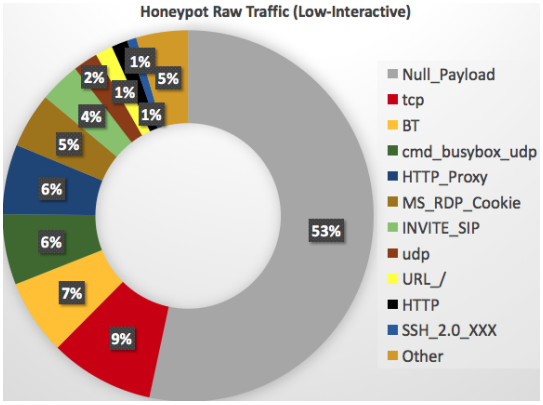

Figure 2 Raw Requests from Low-Interaction.

In addition, we group the traffic based on the port and further reduce the duplicated and similar requests within each group. Figure 3 shows the number of requests on each port. The most popular port attackers tend to scan the port 80, and more than 90 percent of the traffic is a meaningful HTTP request. Due to the recent botnet *Mirai*, scanning on port 7547 suddenly ramps up in the past several months.

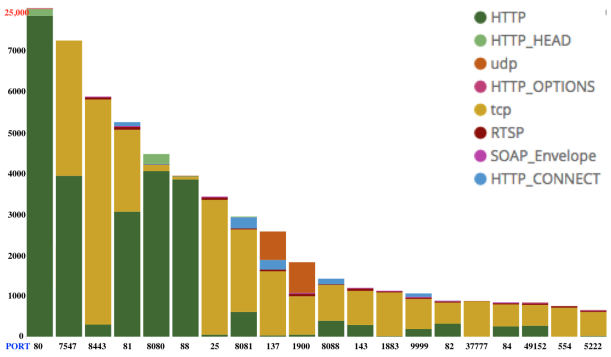

Figure 3 Traffic Type by Ports.

3.4 Exploit Traffic Filtering

When we use the captured traffic as the content to scan internet, it is important to filter out the dangerous ones, such as the requests containing exploit code. In our system, we leverage multiple heuristics and existing detecting tools (e.g. snort rules, our firewall) to detect the exploit code in the traffic. Once the exploit request is detected, we will mark it and won't use it to scan, which means our simulation will stop at this phase.

Remote Command Execution (RCE). Command injection is one of the most prevalence attacks on IoT devices. Attackers usually embed the malicious shell-code inside the request, and send it to the vulnerable device. Due to the poor implementation, the vulnerable IoT device will execute the injected command without authorization. Usually, the injected code can be executed with the privileges of the vulnerable program that handles the request (e.g. the web server).

The body in the HTTP `POST` request is the most common place to embed command. Other protocols can be used to inject command as well. For example, `Mirai` botnet compromises other devices using a documented SOAP exploit located in the implementation of a service that allows ISPs to configure and modify settings of specific modems using the TR-069 protocol (port 7547). One of those settings allows, by mistake, the execution of Busybox commands such as wget to download malware. For example, the embedded shell-code in the *NewNTPServer1* field will drop malicious code and execute it.

```
POST /UD/act?1 HTTP/1.1
Host: x.x.x.x:7547
SOAPAction: urn:dslforum-org:service:Time:1

<?xml version="1.0"?>
<SOAP-ENV:Body><NewNTPServer1>
cd /tmp;wget http://host/1;chmod 777 1;./1
</NewNTPServer1></SOAP-ENV:Body></SOAP-ENV:Envelope>
```

Other protocols can also be used to embed malicious shell-code. For example, multiple types of D-Link routers are vulnerable to UPnP remote code execution attack, allowing the shell-code embedded in the SSDP broadcast packet. The content of the M-SEARCH packet turns into shell arguments.

```
M-SEARCH * HTTP/1.1
Host:239.255.255.250:1900
ST:uuid:`;telnetd -p 9094;ls`
Man:"ssdp:discover"
MX:2
```

Obfuscation and decoding is a common way to evade the detection. For example, we have capture the traffic to the url */shell?%75%6E%61%6D%65%20%2D%61*, which is decoded from the original url */shell?uname+-a* for command injection. We also integrate the detection on the commonly used obfuscation mechanisms.

Info Disclosure. Due to lack of access control and authentication, a great number of IoT devices unintentionally leak information about their configuration and other sensitive information of the system. Often, this information can be leveraged to launch more powerful attacks. Although it is not harmful for the target remote device, we also sanitize such requests.

For example, D-Link personal Wi-Fi Hotspot, *DWR-932*, exposes CGI script */cgi-bin/dget.cgi* to handle most of the user side and server side requests. It replies the request from unauthorized users, so the attacker can view Administrative or wifi password in clear text by padding *DEVICE_web_passwd* as the value of *cmd* parameter in the url. Path traversal is another type of attack for gathering leaked information, such as passing the url like */../../../../etc/shadow*.

Data Tempering. Lots of vulnerabilities on IoT device allows attackers to temper data on the device. For example, IoT devices powered by the operating system *AirOS 6.x* allows unauthenticated users to upload and replace arbitrary files to *airMAX* devices via HTTP of the *airOS* web server, since the "php2" (maybe because of a patch) don't verify the "filename" value of a POST request. An attacker can exploit the device by overriding the file like */etc/passwd* or */tmp/system.cfg*. Similar to detecting the path traversal attack, we sanitize the path to sensitive files in requests.

3.5 Scanning Result

Due to the budget, we only deploy our IoT-Scanner on 3 machines in our lab. They fetch the seed request from a shared *redis* queue, and the newly captured request will be inserted into this queue as well. For each second, we send *300* different requests using separate threads, and set the timeout as 3 seconds. To speed up the scanning process, we tend to reuse the established session to the previous scanned IP addresses. Therefore, we send 10 requests to the same host machine at the same time. We also have 3 machines to periodically check the change of open port of existing marked IoT devices, and scan through the internet to find more available host machines.

For the existing seed requests, we successfully finish scanning around one week and collect 2 million responses in the database, although lots of open ports failed to reply any response or reset the connection. Since the majority of the scanning data is HTTP traffic, we extract the status code from them (Table 3) to quickly analyze them. For all of the ports, response code with 403 (Forbidden), 404 (Not Found) and 401 (Unauthorized) are the top 3 status codes from the IoT devices.

Rsp \ Port	8000	80	8080	88	7547
403	651,646	120,659	12,953	26,660	0
404	88,034	175,497	30,746	10,789	3,832
401	31,468	36,388	36,863	3,870	373
200	3,483	3,742	1,289	300	1267
501	481	1,898	6,337	3	6,080
307	40	0	0	0	0
unknown	52	1,693	10	2	2720
others	1,320	8,193	1,938	6	5140

Table 3 Scanning Result for HTTP protocol.

4. IOT-ID: PINPOINT IOT DEVICE

Currently, we determine whether the machine behind a given IP address is to check the existence of some predefined keywords or patterns in the response. For example, if we found one of the ports in the given IP address returns a header that contains pattern "*NETGEAR WNR1000v3*", we can tell that the machine behinds this IP address is a netgear

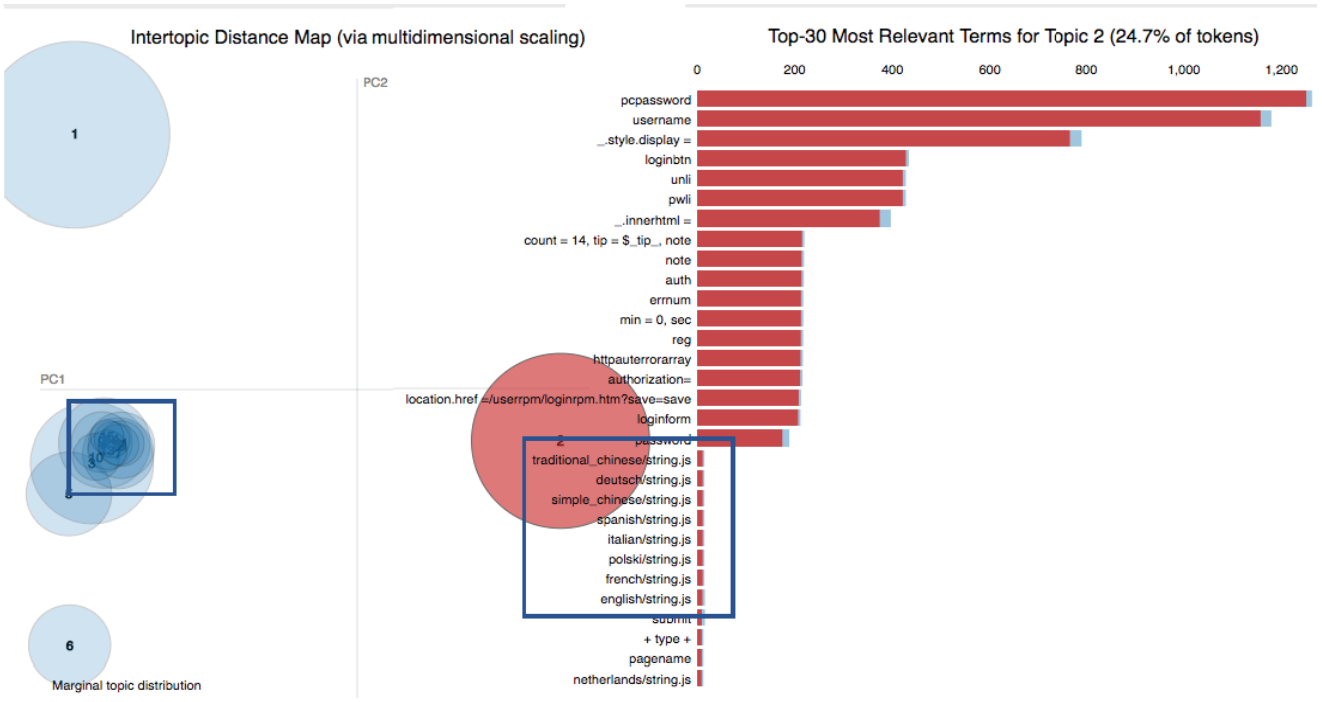

Figure 4 Visualization of Exemplary LDA Model

router and its version. However, many factors may lead to the ambiguous or even wrong information about the IoT device. For example, we observed that some IP addresses returns banner contents of multiple IoT devices in different ports. It returns a login portal of a camera at port 80, and returns management page of a switch at port 99. Another issue is that we found thousands of IoT devices changed their IP address frequently. As the result, we mark the IP address as an IoT device when we fetch the banner content from it, and it changed to a none-IoT device when we scan it to collect response to various of requests.

Therefore, we propose the concept *IoT-ID* that enables us to distinguish different IoT devices and obtain accurate knowledge of them. Our idea is to leverage machine learning algorithm to cluster the scanning result and extract patterns from them as the signature of certain type of IoT device.

LDA-based Solution. To find the signatures, we take a closer look at the IoT traffic we collected. Our insights about the traffic is that traffic of similar devices should contain similar pattern. Based on this, we treat our problem as a natural language processing problem. Our goal is to find some word-combination, which is referred as *topic* in NLP , such that the combination can uniquely label common libraries, common expressions for same brand and firmware.

We propose to use a generative statistical model, Latent Dirichlet allocation (LDA) [11], that allows sets of observations to be explained by unobserved (*topic*). In detail, we treat each response as a document and the type of IoT device that reply the response as the topic of it. We split each document into a series of *words* by predefined delimiters. Then we calculate the statistical distribution of each word in the corpus and organize them into n categories. Each category is further formulated as topic and each topic is expressed as a generative statistic distribution.

Our implementation is based on one open-source implementation [12]. One of our generated models is presented by open-source LDA visualization tool, `pyLDAvis` [20] in Figure 4. This model contains HTTP traffic from 6 different router vendors and we summarize 15 different topics for them. As shown in the figure, the LDA model can successfully cluster words which are unique in each library and firmware. One library example is shown in the right side of the figure. This library provides multiple language support and LDA can group these language-specific words together as one topic. At the same time, we can find some topics share some common words which implies their traffic conform to common HTTP syntax.

The output of our LDA model is some topics, which is a series of mapping relations between word and its confidential probability. Based on the output, we can efficiently cluster collected similar traffic and extract the topic-to-word mapping as its IoT-ID.

5. IOTLEARNER: AN INTELLIGENT ENGINE TO LEARN IOT BEHAVIORS.

With the help of IoTScanner, our honeypot is enable to reply a valid response to client based on the received request instead of responding the fixed one. In this section, we discuss how to leverage markov decision process model to optimize the response selection with the maximal possibility to capture attacks.

5.1 IoTLearner Overview.

For for each individual request, the *IoTScanner* module could collect at least hundreds or even thousands of responses from the remote host. All of them are *valid* responses, but only few of them are the *correct* ones. This is because, for a given request, various of IoT devices can re-

spond to it under their own logic to process it and generate response. The most straightforward example is the request to access the root path of their web service: some devices may reply the login portal page, others may redirect it to another resource, and the rest may respond with an error page. Therefore, all of the scanning results are potential candidates as the response to the client, but the challenge is to find the one which is expected by the attacker.

Our approach. The idea behind our approach is to first randomly select the response from the candidate pool and record the next move from client side. We assume if we happen to select the correct one, attackers will believe our honeypot is the vulnerable target IoT device and they continue to send the malicious payload (e.g. injected command). Therefore, we need to store each transaction in a *session table*, and leverage machine learning techniques to extract the correct behaviors form the dataset.

System architecture. The architecture of *IoTLeaner* module is depicted in figure 5 to fetch raw response from the database and record each transaction to the database. Every incoming request to the honeypot will be forward to this module, and the selected response will be returned to the client. The core part of the module is *selection engine*, which normalizes the request and fetches the potential responses list from the scanning result. In random selection mode, it just randomly pick one from the candidate list and return it. In MDP selection mode, it first locates the state in the graph from the the normalized request, and followed by the model to select the best one.

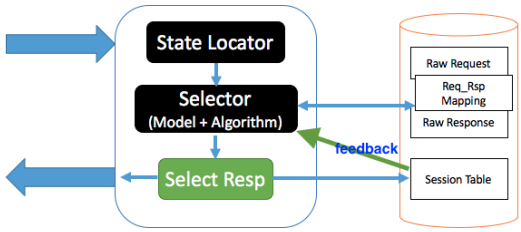

Figure 5 IoTLearner Overview.

Each decision that is made by the selection engine will create a new transaction to extend current session. All of the session information will be stored in the *session table*. Each row in the table represents a transaction like the tuple *<req1, rsp1, conn_info>*, and *conn_info* is the connection information as source ip, source port, destination ip, destination port, and protocol. For example, if the engine selects the response with id *rsp1* to reply the incoming request with id *req1*, such row will be inserted into the session table.

Challenges in learning best response. There are many factors that makes the learning process hard. The first one is that not all the clients who talk to our honeypot are attackers. It leads to the problem that not all the session are malicious to us. Only the ones can reach the exploitation is important to us.

5.2 Model Formulation.

We discuss how we formulate the response selection problem into the markov decision processes model. We assume whether the client continues the session or perform the attack simply determined by the response of the previous request. This is a reasonable assumption based on our best knowledge on existing malware samples we have analyzed. Therefore, we can approximate the statistical structure of session activities using a simple mathematical model known as an *order-1 Markov property*.

Markov decision process (MDP). Markov decision processes [19], also known as stochastic control problem, is an extension of the standard (unhidden) Markov model. MDP is a model for sequential decision making when outcomes are uncertain, such as computing a policy of actions that maximize some utility with respect to expected rewards. A collection of actions can be performed in that particular state, which actions serve to move the system into a new state. At each *decision epoch*, the next state will be determined based on the chosen action through a transition probability function. It can be treated as a *markov chain* in which state transition is determined solely by the transition function and the action taken during the previous step. The consequence of actions (i.e., rewards) and the effect of policies is not always known immediately. Therefore, we need some mechanisms to control and adjust policy when the reward of the current state space is uncertain. The mechanism is collectively referred as *reinforcement learning*.

Problem Formulation. In the standard reinforcement learning model an agent interacts with its environment. This interaction takes the form of the agent sensing the environment, and based on input choosing an action to perform in the environment. Every reinforcement learning model learns a mapping from situations to actions by trial-and-error interactions with a dynamic environment. The model consists of multiple variables, including decision epochs(t), states(x, s), transitions probabilities(T), rewards(r), actions(a), value function(V), discount (γ) and estimation error(e).

The basic rule of reinforcement learning task is the Bellman Equation [3] as expressed as:

$$V^*(x_t) = r(x_t) + \gamma V^*(x_{t+1})$$

It can be explained as the value of state x_t for the optimal policy is the sum of the reinforements when starting from state x_t and performing optimal actions until a terminal state is researched. The discount factor γ is used to exponentially decrease the weight of reinforcements received in the future. From the definition, the problem of RL is essentially to solve a dynamic programming problem. So the standard solution of RL is to use Value Iteration, which represents values V as a lookup table. Then the algorithms can find the optimal value function V^* by performing sweeps through state space, updating the value of each state by update policy until there is no change to state values (the state value have converged). The general update policy can be expressed as:

$$\Delta w_t = max_a(r(x_t, a) + \gamma V(x_{t+1})) - V(x_t)$$

However, to apply RL in our problem, there is one limitation. Our problem is essentially a *non-deterministic* Markov Decision Process, which means at each state, there exists a transition probability function T to determine the next state. In other words, our learning policy is a probabilistic trade-off between exploration, *reply with responses which have not been used before*, and exploitation *reply with the responses which have known high rewards*. To apply general

valuation iteration is impossible to calculate the necessary integrals without added knowledge or some decision modification. Therefore, we apply Q-learning [21] to solve the problem of having to take the max over a set of integrals.

Rather than finding a mapping from states to state value, Q-learning finds a mapping from state/action pairs to values(called Q-values) [10]. Instead of having an associated value function, Q-learning makes use of the Q-function. In each state, there is Q-value associated with each action. The definition of a Q-value is the sum of the reinforments received when performing the associated action and then following the given policy thereafter. Likewise, the definition of an optimal Q-value is the sum of reinforcements received when performing the associated action and then following the optimal policy thereafter.

Therefore, in our problem of using Q-learning, the equivalent of Bellman equation is formalized as:

$$Q(x_t, a_t) = r(x_t, a_t) + \gamma max_{a_t+1} Q(x_{t+1}, a_{t+1})$$

And the update rule of direct Q-learning is formalized as and α is learning rate:

$$\Delta w_t = \alpha[(r(x_t, a_t) + \gamma max_{a_{t+1}} Q(x_{t+1}, a_{t+1}, w_t)) - Q(x_t, a_t, w_t)] \frac{\partial Q(x_t, a_t, w_t)}{\partial w_t}$$

Reward Function. Reward function $r : (x_t, a_t) \rightarrow r$ assigns some value r to being in the state and action pair (x_t, a_t). The goal of reward is to define the preference of each pair and maximize the final rewards (optimal policy).

In our context, the immediate reward $r(x_t, a_t)$ reflects the progress we have made during the interaction process when we choose response a_t to request x_t and we move to the next state x_{t+1}. Since the progress can be either negative or positive, the reward function can be negative or positive as well. The heuristics of defining reward is that if the response a is the target device type expected by the attacker and the attack launch the attack by sending the exploit-code in the next request, the reward must be positive and huge. On the contrary, if the response is not an expected one (e.g. reflects a not vulnerable device version), the attacker may stop the attack and end the session. It leads to the dead-end state, and causes the negative reward. In other words, we reward the responses the could lead us to the final attack packet, and punish the ones that lead to the dead-end session.

One of our designs is to assign reward as a value equals to the length of the final sessions, since we believe the longer request sent by the attackers, the higher chance the malicious payload is contained. The standard session is 2 which means after we send our response, there are at least another incoming request from the same IP at the same port. If no further transition is observed, we assign a negative reward for that response. Other alternative reward assignment could be based on whether we receive some known exploits packets or not.

5.3 MDP Model Build

We explain how to initiate parameters that are required by the model to perform calculation from our existing result.

State and Action. Building the state of the Markov model without any notion of protocol semantics could lead to the lack of generality and sparse state space. Therefore, it is not able to handle anything that has not already been seen. The solution is to simplify and generalize the states by grouping the similar ones to a single state. In our case, we would like to classify the similar requests and similar responses to a same group. Due to the huge number of communication protocols used by IoT device, we have to do it in the protocol-independent way. Previous research [7, 13, 14] have studied how to simplify the state without understanding the protocol, on HTTP, SDP, NSS, NTS and etc. The details implementation can be found in these papers. In general, they all rely on the alignment algorithm to identify the similar portion of multiple strings as the structure of the protocol.

State Transition Probabilities. State transition probabilities can be described by the transition function $T(s, a, s')$, where a is an action moving performable during the current state s, and s' is some new state. More formally, the function transition function $T(s, a, s')$ can be described by the formula:

$$P(S_t = s' | S_{t-1} = s,\ a_t = a) = T(s, a, s')$$

where a is an action moving performable during the current state s, and s' is some new state.

In our context, transition function $T(s, a, s')$ refers to the probability of receiving request s' as the next one within the same session if we reply response a to the client as the reply of its current request s. To measure the probability of each combination of (s, a, s'), we deployed a naive algorithm by randomly return a response from the candidate set and saved the session information to the *session table*. After running a period of time, we are able to collect lots of sessions, and we parse each of them to count the occurrence of each combination (s, a, s'), which is denoted as $C(s, a, s')$. The value of the transition function $T(s, a, s')$ are defined as follows:

$$T(s, a, s') = C(s, a, s') / \sum_{x}^{x \in S} (s, a, x)$$

Online Q-learning Algorithms for Response Selection. Based on the Q-learning model, our learning process starts from receiving a request at the t_0 decision epoch. Given the request, we apply our matching algorithms to select a set of *identical* responses. We adopt ϵ-greedy [23] policy for action selection. In particular, we assign uniform probability for each available response as the initial transaction functions. Using this policy either we can select random action with ϵ probability and we can select an action with $1 - \epsilon$ probability that gives maximum reward in given state.

Then we start our Q-learning iteration and update our Q-learning table. When we learn reinforce for one state and action pair, $r(x_t, a)$, we first back propagate and update the Q lookup table. According, we can do the adjustment by removing the responses which ends with negative rewards and updating the ϵ value. The iteration ends until the model converges.

In practice, our model is running online and update in real time. Therefore, it may not converge and reach the global optimal. However, we think the model is still valuable because it only allows us to discard these undesired responses but also keeps sessions going as best as we can.

5.4 MDP for IoT Honeypot.

We will use the real world example to explain how we build MDP and calculate the probabilities for each response. For the demonstration purpose, we simplified the number of state and action in the graph and represent the model as a `state-space graph`. In the graph, each rectangle (black) represents a single state and each circle (orange) represents an action can be taken at certain state. The arrow in the graph refers to the transition from one state to the next state after certain action is taken. In our context, each state is a unique request abstraction, and each action for a request is the unique response candidate to reply.

Build MDP from Session Table for HNAP protocol. HNAP protocol is quite simple, especially for the attacking scenario. For example, before attacking our honeypot, attackers usually send a request to the url */HNAP1/* to get a SOAP response (an XML-formatted document with the resulting data) which contains the information of the host machine and its supported SOAP actions. Some devices do not support HNAP protocol, such as *SonicWall* firewall which returns *404 Not Found* page and *TRENDnet* router which returns *401 Unauthorized* page. Others may reply valid SOAP response but with different informations in the response, such as the model name as *WRT110* from response of *LinkSys* router and the model name *DIR-615* from response of *D-Link* router. Since we do not have any prior knowledge on the HNAP protocol and the expected behavior, we may try to send each of them to the attacker at the first stage and record the session information.

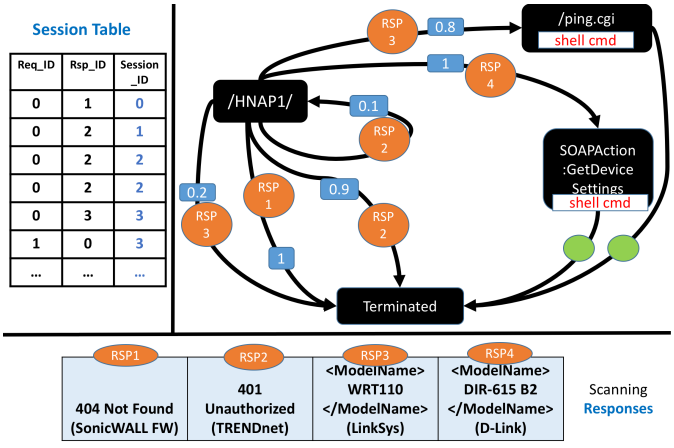

Figure 6 Build MDP State Graph From Session Table.

CGI-script MDP graph. The graph build by the complete session data we have captured is more complex. We select the request to the URL of *CGI* script and generate the graph of them. *CGI* script is used by many types of IoT devices, including camera, router and etc. Lots of preliminary checks and vulnerabilities have been performed on *CGI* script. As figure 7 shows, URLs such as *get_status.cgi*, *check_user.cgi* and *get_camera_params.cgi* are frequently scanned by attackers. Since they are accessible without any privileges, attackers tend to gather device information from them. After few requests, the session goes to the privileged *CGI* script and vulnerable ones.

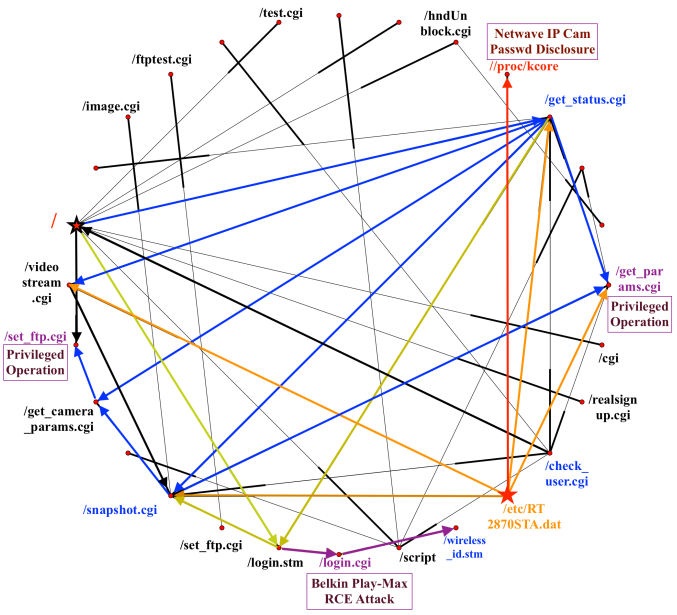

Figure 7 CGI-script MDP Graph.

6. EVALUATION

In this section, we evaluate the effectiveness of our intelligent-interaction honeypot and share some interesting findings from the captured requests. We have deployed it in 5 virtual machine from *Digital Ocean*. Due to the budget, we only choose the smallest one with configuration as: 512MB memory, 1 CPU, 20G SSD disk, and standard network bandwidth. For the preparation, we utilized the *1 million* requests that captured by the low interaction honeypot as seed to scan the internet and collected millions of responses using the scanner we explained above.

6.1 Session Improvement

The length of each session is one of the critical an easily measured indicator to demonstrate the effectiveness of our learning process: the more requests we can allure the attacker to send, the higher chance we can capture the exploitation. For the low-interaction honeypot, the majority of the session ends up within 2 transactions, so the average is below 2.

In our setup, we select 30k unique responses from scanning results and download a copy to each honeypot instance. The mapping between the response and the corresponding seed request is downloaded as well. We configure the honeypot as random reply mode at the first two weeks, in order to collect the reaction from clients. With the random selection mechanism, we managed to receive more requests from client. However, as figure 8 depicts, the majority of the session is still very short. This result is reasonable considering that we can get hundreds or even thousand unique responses from the scanner for an individual request.

Each honeypot instance contains 300k responses we have selected and its mapping to certain request. 2 of the honeypot instances use random selection algorithm, which will randomly choose a response from the pool that collected using the current received request; and the rest of them deployed the MDP algorithm as we explained in the previous

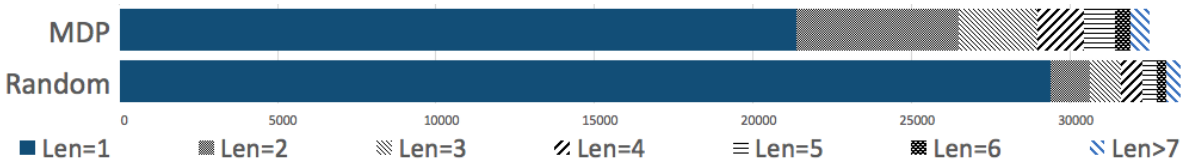

Figure 8 Session Length Distribution.

section. With 1 month of running, we have collected *1 million* valid requests.

6.2 Captured Pre-attack Check and Exploitation.

Utilizing intelligent-interaction honeypots, we have captured more crafted malicious requests from attackers. Using the session information, we also identified 50 pre-attack checks in various protocols for different types of IoT devices (e.g. IPCam, Router, Projector) from the MDP state graph. We highlights several cases in this part.

6.2.1 In HTTP Protocol

HTTP protocol is widely used by IoT devices for management. Device information can be leaked from all types of HTTP responses directly or indirectly. We discuss our observation based on the status code of the response.

200 OK. This is the most standard response for successful HTTP requests. Several versions of Netgear routers are vulnerable to leak model version, firmware and other information in the response to the request on */currentsetting.htm*. If attacker cannot parse or obtain valid token from the response content, they won't perform the attack action. For *Netwave* IP camera, we observe quite large number of request on the URL *get_status.cgi*, */etc/RT2870STA.dat* and *login.stm*. Authentication is not required to access these pages which reveal firmware versions (ui and system), timestamp, serial number, p2p port number, or wifi SSID.

401 Unauthorized. HTTP status code, *401 Unauthorized*, means the resource cannot be loaded due to the incorrect authenticated method or not authenticated at all. It seems like the 401 response is barely contains any other meaningful information. However, it still may embed information that helps client to perform next action (e.g. re-authentication, redirection).

For example, *WWW-Authenticate* field in the response header is used to describe the authentication schema [8]. Normally, this response generally returned from the web server to the IoT device, not from the web app[1]. All types of IoT device with the same brand share the same mechanism. Therefore, attackers could utilize the value of this field to determine current IoT device is vulnerable or not, especially for *Netgear* modems or routers. Our honeypot observed the check for pattern *NETGEAR R7000* and *NETGEAR R6400* to launch the remote code injection attack (CVE-2016-6277) on this specific router version.

Sometimes the content of 401-unauthorized response may leak sensitive information as well. For example, netgear wireless router (*N150 WNR1000v3*) contains credentials (token) when the request failed on the basic login attempt, as the following snippet shows:

[1]The 403 Forbidden response is tied to the web app's logic.

```
HTTP/1.0 401 Unauthorized
WWW-Authenticate: Basic realm="NETGEAR WNR1000v3"

<html><head><title>401 Unauthorized</title></head>...
<form method="post" action="unauth.cgi?id=2143918018"
     name="aForm">
</form></body></html>
```

We find out that attackers need to successfully find the pattern and extract the token from it before crafting malicious payload to further exploit the router.

404 Not Found. When the requested resource could not be found, the server will return the 404-not-found response. 404 page can also be used to identify an IoT device by attackers. For example, we have observed the attacks on ZyXEL's modem *Eir D1000*. Attackers send a legitimate request to the URL */globe*, and expect a 404-not-found page with pattern *home_wan.htm* in it. This special 404 page tells attackers that the host supports the SOAP-based protocol TR-069 and they can inject command to it by embedding it in the 'NewNTPServer' field of the request.

6.2.2 In Customized Protocol

Besides HTTP protocol, preliminary check happens on IoT protocols and even customized ones. Home Network Administration Protocol (HNAP) is one of the example. This SOAP-based protocol first invented by Cisco at 2007 for network device management, which allows network devices (e.g. routers, NAS devices, network cameras, etc.) to be silently managed. Due to the long history of its buggy implementation, lots of attacks have been discovered, such as utilizing *GetDeviceInfo* action to bypass authentication and inject shell command to the *SOAPAction* field to launch the RCE attack. As we discussed in the section 5, attackers commonly check the response to the URL */HNAP1/* to get service list supported by the device. Another example is the customized protocol used by *Netcore* and *Netis* routers, which open an UDP port 53413 as the backdoor for remote configuration. Attackers send a payload with 8 bytes of value '*\x00*' and expect pattern like '*\xD0\xA5Login:*" in it to confirm the device is vulnerable.

6.2.3 Using Echo Command.

Remote command execution is one of the common attacks on IoT devices, and some of the vulnerabilities allow attackers to view the output of commands. For example, the recent found vulnerability on router NETGEAR DGN2200 for all of its firmware versions (v1-v4) which does not require admin access to execute shell commands on the router. The vulnerable script is ping.cgi which is designed for users to submit diagnostic information to the router. However, due to the implementation flaw, if attackers send a POST http request to this url with the command as the value of parameter *ping_IPAddr* in the payload, they are able to execute

the command with *nobody* permission. The returning page contains the result of the injected command.

We have captured malicious requests that tries to take advantage of this vulnerability, but the command is very simple such as `echo` command. We observed that the majority of the malicious sessions are terminated at the requests with `echo` command in it. It is because the attackers usually generate a random string after the `echo` command, and the response content should contain the exact same string if the command is executed. However, since the random string changed in every request, it is highly possible that the string in the scanned result match the current received request.

```
POST /ping.cgi HTTP/1.1
referer:http://x.x.x.x/DIAG_diag.htm

IPAddr1=1&IPAddr2=2&IPAddr3=3&IPAddr4=4&ping=Ping&ping_
IPAddr=12.12.12.12; echo "zP8ZDXwQCC";
```

As the request example shows above, the attacker check whether the random string *zP8ZDXwQCC* is in the response before sending the real exploit shell code. We handle this type of check by inserting the string in the echo command to the right place in the response page.

7. CONCLUSION

Building honeypot for IoT devices is challenging using the tradition methods due to the special characteristic of IoT. However, attacks on IoT devices perform preliminary check on the device information before launching the attack. Without the proper interaction mechanism with the attacker, it is extremely hard to capture the complete exploit payload. We propose an automatic and intelligent way to collect potential responses using scanner and leverage machine learning techniques to learn the correct behaviors during the interaction with attackers. Our evaluation indicates that our system can improve the session with attackers and capture more attacks.

8. REFERENCES

[1] Paul Baecher, Markus Koetter, Thorsten Holz, Maximillian Dornseif, and Felix Freiling. The nepenthes platform: An efficient approach to collect malware. In *Recent Advances in Intrusion Detection*, pages 165–184. Springer, 2006.

[2] Edward Balas and Camilo Viecco. Towards a third generation data capture architecture for honeynets. In *Information Assurance Workshop, 2005. IAW'05. Proceedings from the Sixth Annual IEEE SMC*, pages 21–28. IEEE, 2005.

[3] Bellman equation. `https://en.wikipedia.org/wiki/Bellman_equation`.

[4] Matthew L Bringer, Christopher A Chelmecki, and Hiroshi Fujinoki. A survey: Recent advances and future trends in honeypot research. *International Journal of Computer Network and Information Security*, 4(10):63, 2012.

[5] Here's how the internet of things will explode by 2020. `http://www.businessinsider.com/iot-ecosystem-internet-of-things-forecasts-and-business-opportunities-2016-2`.

[6] censys.io website. `https://censys.io/`.

[7] Weidong Cui, Vern Paxson, Nicholas Weaver, and Randy H Katz. Protocol-independent adaptive replay of application dialog. In *NDSS*, 2006.

[8] John Franks, Phillip Hallam-Baker, Jeffrey Hostetler, Scott Lawrence, Paul Leach, Ari Luotonen, and Lawrence Stewart. Http authentication: Basic and digest access authentication. Technical report, 1999.

[9] Juan David Guarnizo, Amit Tambe, Suman Sankar Bhunia, Martín Ochoa, Nils Ole Tippenhauer, Asaf Shabtai, and Yuval Elovici. Siphon: Towards scalable high-interaction physical honeypots. In *Proceedings of the 3rd ACM Workshop on Cyber-Physical System Security*, pages 57–68. ACM, 2017.

[10] Mance E Harmon and Stepanie S Harmon. Reinforcement learning: A tutorial. In *Technical Report*, 1997.

[11] Latent dirichlet allocation. `https://en.wikipedia.org/wiki/Latent\_Dirichlet\_allocation`.

[12] Open source lda implementation. `https://radimrehurek.com/gensim/models/ldamodel.html`.

[13] Corrado Leita, Marc Dacier, and Frederic Massicotte. Automatic handling of protocol dependencies and reaction to 0-day attacks with scriptgen based honeypots. In *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2006.

[14] Corrado Leita, Ken Mermoud, and Marc Dacier. Scriptgen: an automated script generation tool for honeyd. In *Computer Security Applications Conference, 21st Annual*. IEEE, 2005.

[15] masscan github. `https://github.com/robertdavidgraham/masscan`.

[16] NSFOCUS. Analysis on exposed iot assets in china. http://blog.nsfocus.net/wp-content/uploads/2017/05/Analysis-on-Exposed-IoT-Assets-in-China-0521.pdf, 2017.

[17] Yin Minn Pa Pa, Shogo Suzuki, Katsunari Yoshioka, Tsutomu Matsumoto, Takahiro Kasama, and Christian Rossow. Iotpot: analysing the rise of iot compromises. *EMU*, 9:1, 2015.

[18] Niels Provos et al. A virtual honeypot framework. In *USENIX Security Symposium*, 2004.

[19] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[20] Pyldavis. `https://pyldavis.readthedocs.io/en/latest/`.

[21] Q-learning. `https://en.wikipedia.org/wiki/Q-learning`.

[22] shodan.io website. `https://shodan.io/`.

[23] Michel Tokic. Adaptive ?-greedy exploration in reinforcement learning based on value differences. In *Advances in Artificial Intelligence*, 2010.

[24] Zhaoyan Xu, Antonio Nappa, Robert Baykov, Guangliang Yang, Juan Caballero, and Guofei Gu. Autoprobe: Towards automatic active malicious server probing using dynamic binary analysis. In *Proceedings of the 21st ACM Conference on Computer and Communications Security(CCS 14)*, November 2014.

[25] zoomeye.org website. `https://zoomeye.org/`.