A Deep Look Into Privacy and Security Of Vacuum Robot

Trong Nguyen
Department of Computer Science
Cal Poly Pomona University
trongnguyen@cpp.edu

ABSTRACT

IoT has become one of the fastest-growing industries along with Artificial Intelligence and Machine Learning. Nowadays, many people have smart devices in their homes. The more popular these devices become, the more effort we should invest in investigating their security.
Intelligent vacuum robot is one of the IoT devices that recently have gained public interest. Consumers have widely adopted vacuum robots, while their security has not been evaluated thoroughly. Researchers worldwide keep finding new flaws in vacuum robot systems, and many of the findings could lead to cyber-attacks.

In this paper, I would like to extend the topic by analyzing a popular vacuum robot device to exploit vulnerabilities and bring awareness to consumers. First, I used dynamic and static analysis to communicate between the mobile application and vacuum robot cloud server. After that, the consequences of the attacks were categorized using three aspects of CIA triads. I also used STRIDE threat modeling to construct attack scenarios based on the found vulnerabilities. Finally, I suggested the manufacturers' mitigations to secure their devices to protect users' data. [1]

1. INTRODUCTION

Many researchers warn users about the security risk of vacuum robots in the past. For instance, a group of researchers in the mobile network lab used reverse engineering on the robot's firmware. It figured out the symmetric key that could authenticate all neato vacuum robots [2]. Then this allowed the researchers to control all vacuum robots on their behalf as long as they knew the robot serial number. This raised a serious security concern because attackers may affect the availability of the vacuum robot system. Some attacks the researchers conducted was stealing attack, data leak attack, or discovering victim's IP address.

Another article tells us about a more serve attack by exploiting the vacuum robot's lidar sensor [3]. In detail, the researcher found a way to root the Xiaomi vacuum robot and gain root access to it. Xiaomi vacuum robots and many others in the market run Linux as their OS, so it's always possible to root them. After gaining root access, the researchers modified its software to have the lidar sensor catch sound vibrations.

Then they used machine learning to train a data set with sound vibration patterns and English words to generate an inference map between sound and English words. This model is then used to match sound vibration with 24 letters of the English word. Then using a LIDAR sensor on the robot, the research could catch the sound from the surrounding, send it over the internet, preprocess it, and use the machine learning model to predict the English word based on the lidar information. The attack allowed researchers to spy on victims to listen to sensitive information such as bank information and social security number.

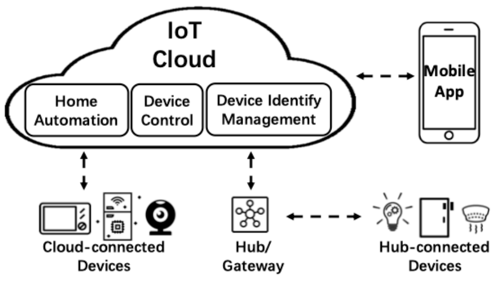
2. MOTIVATION

Recently, on E-Commerce websites such as Amazon, consumers have seen more and more vacuum robot models appear on the markets. There are so many unknown brands which offer good deals for their product. Unfortunately, sometimes a good deal does not come with best security practices. Normal users who purchase those products may not be aware of the cybersecurity risk they possess. As a graduate student interested in the Cyber Security field, I would like to research the vacuum robot ecosystem, sample one robot device model in the market, and conduct a security analysis experiment. By the end of this research paper, I hope I can bring awareness to the public about the hidden risks of those tiny helpful robots.

Some vacuum robots are designed a mini tesla car because it possesses hardwares such as LIDAR sensors, AI camera, collision sensor just like a tesla car does. I could say the vacuum robot is a mobile data collector that can be hacked, so the user's privacy will be affected. Moreover, when looking at cyber hacking, I believe many system breaches don't need to be performed using full-scale, high-performance computers. It could come from or tiny devices that we may not be aware of. For example, the Mirai attack in 2016 turned a hundred thousand IoT devices such as IP cameras, home routers into bots and used them to attack third-party servers.

3. EVALUATION

3.1 IoT Ecosystem



IoT Cloud Infrastructure

The IoT ecosystem consists of three main entities. The first entity is the IoT end device. It could be the devices connected directly to the cloud, such as some low-budget cameras and home assistants like Amazon Echo and Google Nest. These devices could connect through a hub; then, the hub would handle communication with the cloud server. The second entity is the mobile application. The mobile app users use to create accounts, manage their IoT device, and perform their IoT device through the mobile app. The mobile app would then establish a communication link with the IoT cloud, the third entity. IoT cloud contains home automation logic, device control, device identity management.

Vacuum Robot Ecosystem: Vacuum Robot Ecosystem also derives from the IoT ecosystem mentioned above.

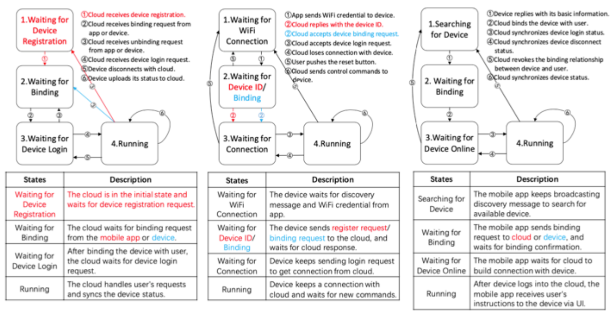
Vacuum Robot: There are four types of vacuum robots in the market now. First, the vacuum robot is equipped with an ultra-sonic sensor. This kind of robot uses a passive mechanism to navigate through the house. It would bump everywhere to finish house cleaning without having a planned map. The second type of vacuum robot is equipped with a Lidar Sensor. This kind of robot uses a laser beam to scan our house and construct a map; then, it would establish a route based on the map in memory. This kind of robot would have our house floor plan in its memory. The third type of vacuum robot uses a camera tilt of about 45 degrees to navigate our house. This kind of robot would try to scan the house's wall first then use another sensor to map our house. The last type is the most expensive and advanced one. It uses both a LIDAR and a camera. LIDAR sensor is used to map the home, and AI camera is used to detect an object to steer the robot always from objects

Vacuum Robot Mobile App: This is responsible for account registration, login, set up, and robot control. Users can use the app to connect with the robot directly or with a cloud server.

Vacuum robot cloud server: Used to handle account registration, device binding requests, and relay users command to the robot

3.2 State machine of IoT device or vacuum robot

All IoT devices in the market, including vacuum robots, follow state machines for their operational modes. These states machines can be illustrated through the following diagram. [4]


1.Waiting for Device Registration
2.Waiting for Binding
3.Waiting for Device Login
4.Running
①Cloud receives device registration.
②Cloud receives binding request from app or device.
③Cloud receives unbinding request from app or device.
④Cloud receives device login request.
⑤Device disconnects with cloud.
⑥Device uploads its status to cloud.
1.Waiting for WiFi Connection
2.Waiting for Device ID/ Binding
3.Waiting for Connection
4.Running
①App sends WiFi credential to device.
②Cloud replies with the device ID.
②Cloud accepts device binding request.
③Cloud accepts device login request.
④Cloud loses connection with device.
⑤User pushes the reset button.
⑥Cloud sends control commands to device.
1.Searching for Device
2. Waiting for Binding
3.Waiting for Device Online
4.Running
①Device replies with its basic information.
②Cloud binds the device with user.
③Cloud synchronizes device login status.
④Cloud synchronizes device disconnect status.
⑤Cloud revokes the binding relationship between device and user.
⑥Cloud synchronizes device status.

States	Description
Waiting for Device Registration	The cloud is in the initial state and waits for device registration request.
Waiting for Binding	The cloud waits for binding request from the mobile app or device.
Waiting for Device Login	After binding the device with user, the cloud waits for device login request.
Running	The cloud handles user's requests and syncs the device status.

(a) State Machine of an IoT Cloud

States	Description
Waiting for WiFi Connection	The device waits for discovery message and WiFi credential from app.
Waiting for Device ID/ Binding	The device sends register request/ binding request to the cloud, and waits for cloud response.
Waiting for Connection	Device keeps sending login request to get connection from cloud.
Running	Device keeps a connection with cloud and waits for new commands.

(b) State Machine of a Device

States	Description
Searching for Device	The mobile app keeps broadcasting discovery message to search for available device.
Waiting for Binding	The mobile app sends binding request to cloud or device, and waits for binding confirmation.
Waiting for Device Online	The mobile app waits for cloud to build connection with device.
Running	After device logs into the cloud, the mobile app receives user's instructions to the device via UI.

(c) State Machine of a Mobile App

Figure 2. IoT State Machine

At initial set up, the robot device waits for a Wi-Fi connection; the mobile application would search for the device. The cloud server would wait for the device registration. The robot device needs to send basic information such as the serial number to the mobile app to move to the second state. The mobile app would send the user id and robot serial number to the cloud server to initiate the binding process. Next state, the robot device would connect to the cloud server, the mobile app will synchronize device login status. In-state 4, the robot device would always try to keep the connection with the cloud server. Every time it loses contact with the cloud, it would try to reconnect. The link would be terminated until the users push the reset button on the robot device. The mobile application continuously synchronizes with the device connection status. The mobile application can also unbind the user account and robot device. The cloud receives binding and unbinding requests and gets a status update from the robot device, then relays those statuses to the mobile application.

However, most IoT devices lack a mechanism to enforce the consistency of those states. The state consistency means a state of all three entities must be synchronized, and none of these entities should be out of state. This mechanism should have existed to prevent such attacks as a binding attack, stealing attack.

Consequently, the IoT ecosystem suffers from flaws. One of the flaws is insufficient to state guard; for example, in-state 4, the cloud should only accept device control requests, not binding requests. However, in my next experiment, we would see the cloud also accepted that kind of request. The other flaws are unauthorized device login and unauthorized device unbinding. Ideally, the cloud should only allow requests issued from the bond with the owner's account. However, also in my experiment, we would see the cloud accept requests unconditionally from any users, even letting them unbind the device.

3.3 Experimental Setup

In this experiment, I chose a popular vacuum robot to do a comprehensive cybersecurity analysis. This project aims to verify the IoT flaws mentioned above and measure the severity of damage that a malicious vacuum robot can cause to a user's privacy. The scope of these experiments would focus on the vacuum robot mobile application to investigate the data collection. In addition, I would observe the data communication between the mobile app and the cloud server to see if we can perform data leaking attacks and device stealing attacks.

3.3.1 Vacuum robot selection

The vacuum robot system that I used to experiment with is Proscenic M7 Pro on Amazon with its Android mobile application app Proscenic home. This type of vacuum robot has more than 2000 reviews on Amazon, and the mobile app has been downloaded more than 100,000 times from the app stores. Therefore, any vulnerability that we could find would be significant damage to the user's privacy. The chosen robot is equipped with LIDAR and ultrasonic sensors. The map data generated by this robot would be stored on Proscenic's cloud server. This robot also requires a WiFi connection, and the control commands are made by the mobile app then relayed through a cloud server.

3.3.2 Scope Of Empirical Vulnerability Analysis

The figure below shows the communication between the three entities. The red arrow and letters show what kind of analysis and testing I would perform on which entity. First, I would perform privacy analysis on the mobile application to investigate data collection on the vacuum mobile app. Next, I observed the data exchange between the mobile app and robot on the link between these two entities. Then I intercepted these requests and forged the requests to see if I could illegitimately retrieve data or control the device on behalf of the actual user.

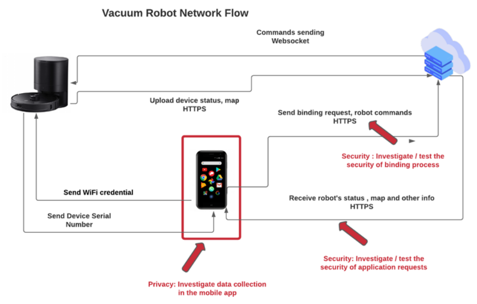

Vacuum Robot Network Flow
Commands sending Websocket
Upload device status, map HTTPS
Send binding request, robot commands HTTPS
Security : Investigate / test the security of binding process
Send WiFi credential
Receive robot's status , map and other info HTTPS
Send Device Serial Number
Security: Investigate / test the security of application requests
Privacy: Investigate data collection in the mobile app

Figure 3. Scope Of Vulnerability analysis

3.3.3 Experiment Methods:

To complete the experiment, I used two standard techniques in cybersecurity called static and dynamic analysis. The static analysis looks at applications at rest, such as searching through their source code to identify manicous code or library. For example, the reverse engineering process can be done using APK-Tool or JAD-GUI to convert Android installation file APK format to java source code [5]. On the other hand, the dynamic analysis looks at the data exchange when a mobile app communicates with a cloud server using Wireshark or mitmproxy. The researchers would then examine the packet captured by the tools to understand more about the data's protocols and business logic.

Apk-tool and Jad-Gui are the tools used to decode Android Apk files into original form, Java code. As a high-level programming language researcher, I could read and understand the application's behaviors with Java source code. App's behaviors contain business logic and are data-driven, so we could have a complete picture of the manufacturer's purposes. In addition, those tools also give us the ability to know about the 3rd libraries that the android app is using. In many cases, the Android apps use multiple 3rd extensions to speed up their software development life cycle. However, the cons are many 3rd extensions do not have their code public verified for security. It means that there is a big chance that an Android app is not malicious, but their 3rd library is, but the app is not aware of that issue. The reverse engineering tools are also crucial by extracting the developer's comments in the source code. There are cases that investigators get the developer's intention just by reading the code comments.

mitmproxy is an essential tool used in this research for dynamic analysis [6]. The software acts as an intermediate communication channel between the client and server. For example, we all know that an HTTPS connection is secured by an SSL certificate, which means data transfer between those two endpoints will be encrypted by public keys. mitmproxy would stand between the client and server. When the client sends the necessary information to the server to establish the TLS connection, mitmproxy will intercept and pause that connection. Then mitmproxy would use the client's information and send the SNI to the server on behalf of the client. The server would be tricked and send back the values needed to generate the interception certificate. The client is also be tricked into making it believe communicating directly with the server. The mitmproxy is the one that produces an interception certificate, so it holds the necessary private key to decrypt the traffic between client and server. Hence, using mitmproxy, I could see the decrypted data between the mobile app and the cloud server to better understand the app's behavior.

One of the most exciting features of mitmproxy is to allow users to intercept requests and modify the requests [7]. After doing that, the request could be replay to the server. Only the parts that user's changes are modified. The rest of the requests remains untouched, including the authentication token. Hence, mitmproxy users do not need to worry about authentication when experimenting. Instead, the researchers could solely focus on penetration testing of the systems. In the investigation of this paper, I would intensively use MITM proxy to confirm security flaws exists in the IoT device ecosystem.

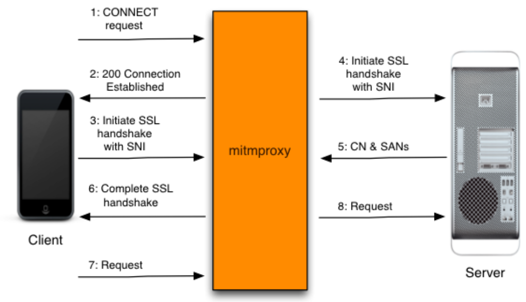

1: CONNECT request
2: 200 Connection Established
3: Initiate SSL handshake with SNI
6: Complete SSL handshake
7: Request
Client
mitmproxy
4: Initiate SSL handshake with SNI
5: CN & SANs
8: Request
Server

Figure 4. mitmproxy traffic flow

3.3.3 Scope of Empirical Vulnerability Analysis

However, Android OS has a mechanism to check whether the server possesses a well-trusted certificate or flag that connection and refuses to exchange the data. I used a Nexus phone; then, I rooted it to alter the system file. Next, I injected a mitmproxy certification into the system file; then, the Nexus 6 will unconditionally trust the mitmproxy server and accept exchanging data. The Proscenic application worked perfectly on the rooted Android phone and communicated with the cloud server without any issues. Consequently, all the network communication between the Proscenic app and the cloud server was exposed to the mitmproxy server and ready to be modified.

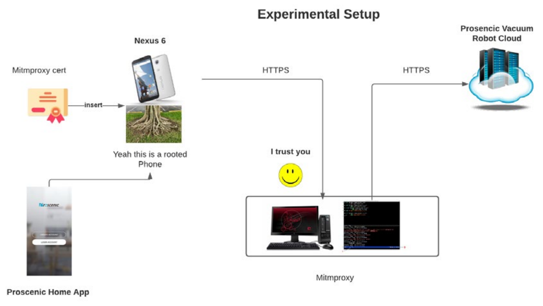


Figure 5. Technique to overcome the Ssl Issue on Mitmproxy

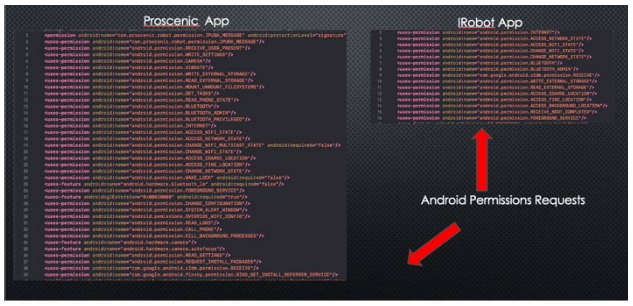
3.4 Proscenic Mobile App Data Privacy Investigation

Using mitmproxy, I detected some suspicious web requests made by the vacuum robot mobile app. These requests were periodically sent to two remote servers, which were Alibaba cloud and JD cloud. These cloud services have been known for collecting user data for analysis. I was curious about what kind of data they were collecting and decided to use more techniques to reveal it later. I noticed another suspicious thing that the device posted data even when the app was running in the background. Right after the device was unlocked, the application immediately made post requests to those endpoints.

One more thing, the request body was encrypted to obfuscate the content to prevent someone from seeing the data payload. We all know that internet communication is protected by public-private critical infrastructure known as SSL or HTTPS. It means that web request's payload data are encrypted. However, in this experiment, we see that the payload data in submissions was also encrypted again using an encryption algorithm. We believe the extra encryptions here played another purpose except protecting user data. The programmer who develops the app may want to hide the data payload to prevent normal users from knowing about collected data. Also, some research about the JPUSH library presents that this library is famous for collecting sensitive information from users [8]. For that reason, I decided to use reverse engineering techniques to reveal the myth.

One quick way to overview Android application usage is to look at the Android manifest.xml file. I used the APK tool and Jad-GUI to look at the permission manifest of the app. Every android application, when being developed, has the manifest file indicating which permissions they need and will request those permissions from the Android OS system. Hence, an android application can request as many as it desires. Then all responsibility of allowing these permissions would solely depend on users. If we use Android phones, we would be asked if we want to allow some permissions when launching an app. Please make sure to read that permission carefully and do not allow all permissions without reading them. Some manicous apps request more permission than they need.

In the below figure, the image on the left shows permissions of the Proscenic robot app. The right image is the permission from the Roomba IRobot app. Clearly, with similar functionalities, the Irobot app only requests about 1/3 of the permission compared to the Proscenic app. By looking closely into the Proscenic Permission, you can see that there is RECEIVE_USER_PRESENT, which allows the app to send data when the user unlocks the phone even the app is running background. MOUNT SYSTEM permission allows the app to access the file system. BLUETOOTH PRIVILEGED allow the app to scan for nearby Bluetooth device. WRITE SETTING allows the app to change the system setting. READLOG allows the app to read OS system log of the phone [9]. More permissions appear in the manifest which the purposes are questionable.



```
<permission android:name="com.proscenic.robot.permission.JPUSH_MESSAGE" android:protectionLevel="signature"
<uses-permission android:name="com.proscenic.robot.permission.JPUSH_MESSAGE"/>
<uses-permission android:name="android.permission.RECEIVE_USER_PRESENT"/>
<uses-permission android:name="android.permission.WRITE_SETTINGS"/>
<uses-permission android:name="android.permission.CAMERA"/>
<uses-permission android:name="android.permission.VIBRATE"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS"/>
<uses-permission android:name="android.permission.GET_TASKS"/>
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
<uses-permission android:name="android.permission.BLUETOOTH"/>
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
<uses-permission android:name="android.permission.BLUETOOTH_PRIVILEGED"/>
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.CHANGE_WIFI_MULTICAST_STATE" android:required="false"/>
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name="android.permission.CHANGE_NETWORK_STATE"/>
<uses-permission android:name="android.permission.WAKE_LOCK" android:required="false"/>
<uses-feature android:name="android.hardware.bluetooth_le" android:required="false"/>
<uses-permission android:name="android.permission.FOREGROUND_SERVICE"/>
<uses-feature android:glEsVersion="0x00020000" android:required="true"/>
<uses-permission android:name="android.permission.CHANGE_CONFIGURATION"/>
<uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW"/>
<uses-permission android:name="android.permissions.OVERRIDE_WIFI_CONFIG"/>
<uses-permission android:name="android.permission.READ_LOGS"/>
<uses-permission android:name="android.permission.CALL_PHONE"/>
<uses-permission android:name="android.permission.KILL_BACKGROUND_PROCESSES"/>
<uses-feature android:name="android.hardware.camera"/>
<uses-feature android:name="android.hardware.camera.autofocus"/>
<uses-permission android:name="android.permission.READ_SETTINGS"/>
<uses-permission android:name="android.permission.REQUEST_INSTALL_PACKAGES"/>
<uses-permission android:name="com.google.android.c2dm.permission.RECEIVE"/>
<uses-permission android:name="com.google.android.finsky.permission.BIND_GET_INSTALL_REFERRER_SERVICE"/>
```

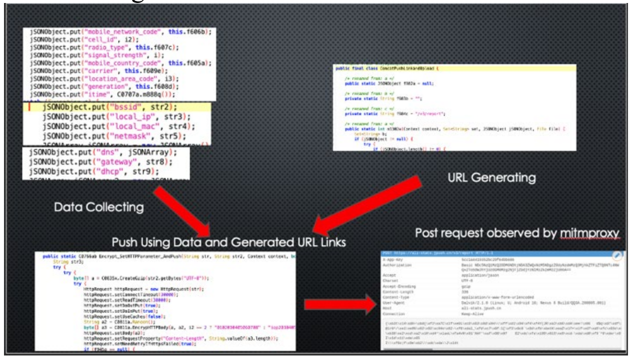
```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>
<uses-permission android:name="android.permission.CHANGE_NETWORK_STATE"/>
<uses-permission android:name="android.permission.BLUETOOTH"/>
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
<uses-permission android:name="com.google.android.c2dm.permission.RECEIVE"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_BACKGROUND_LOCATION"/>
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
<uses-permission android:name="android.permission.FOREGROUND_SERVICE"/>
```

Figure 6. Proscenic permission Manifest file

I took a deeper look by reverse engineer the android app into java source code. I found a code that collects sensitive user data such as our cellular network or data about the user's WiFi SSID , IP and MAC address, DNS. There is another piece of code that is collecting the phone IMEI number.

After collecting those data, the app would use another piece of code to construct the URL to post data to. Then data was ready to be posted to remote servers.

On the bottom left of the below figure, the app used a function to encrypt what they are collecting in the first part, then used a web request post to post the data packet to the URL it had generated.



```
jSONObject.put("mobile_network_code", this.f606b);
jSONObject.put("cell_id", i2);
jSONObject.put("radio_type", this.f607c);
jSONObject.put("signal_strength", i);
jSONObject.put("mobile_country_code", this.f605a);
jSONObject.put("carrier", this.f609e);
jSONObject.put("location_area_code", i3);
jSONObject.put("generation", this.f608d);
jSONObject.put("itime", C0707a.m888q());
```

```
    jSONObject.put("bssid", str2);
    jSONObject.put("local_ip", str3);
    jSONObject.put("local_mac", str4);
    jSONObject.put("netmask", str5);
```

```
jSONObject.put("dns", jSONArray);
jSONObject.put("gateway", str8);
jSONObject.put("dhcp", str9);
```

```
public static C0766ab Encrypt_SetHTTPParameter_AndPush(String str, String str2, Context context,
    String str3;
    try {
        try {
            byte[] a = C0835x.CreateGzip(str2.getBytes("UTF-8"));
            try {
                HttpRequest httpRequest = new HttpRequest(str);
                httpRequest.setConnectTimeout(30000);
                httpRequest.setReadTimeout(30000);
                httpRequest.setDoOutPut(true);
                httpRequest.setDoInPut(true);
                httpRequest.setUseCaches(false);
                String a2 = C0811a.Random();
                byte[] a3 = C0811a.EncryptHTTPBody(a, a2, i2 == 2 ? "0102030405060708" : "
                httpRequest.setBody(a3);
                httpRequest.setRequestProperty("Content-Length", String.valueOf(a3.length));
                httpRequest.setNeedRetryIfHttpsFailed(true);
```

Data Collecting
URL Generating
Push Using Data and Generated URL Links
Post request observed by mitmproxy

Figure 7. Proscenic Data Collection

3.5 Penetration Testing on Proscenic Vacuum Robot Cloud

3.5.1 Vacuum Robot Serial Number Generation

The following experiment is to exploit the vulnerabilities in the vacuum robot cloud server: Previously, I mentioned some IoT devices, including vacuum robot cloud servers, recklessly authorize web requests. In this robot model, I observed and saw if I supplied the correct robot device serial number,the cloud server allowed the request and let me check the online status of other robots, although I never owned it.

The web-API URL used to check the robot's online status is in the figure below. Then in the response body, we would see the online status corresponding to the robot's serial number. Valid means the robot is online, and false means the robot is currently offline. This web request is captured using mitmproxy as well.

The first step of this testing is to find other robot's serial numbers first. Then we can replay the robot checking request to check another robot's online status. For your information, every IoT devices on the market has its serial number generated in two ways:

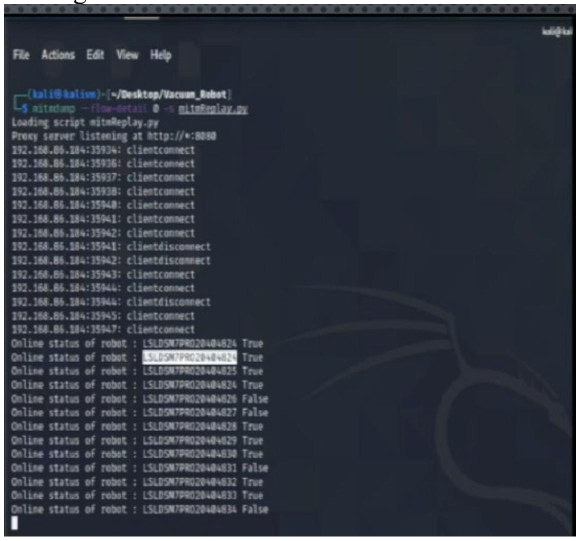
-Type 1: The device sends identity information to the IoT cloud. Then, the cloud generates the id and returns it to the device. This is also called dynamic device identity assigning.

- **Type 2**: Device id is generated by the device platform and hardcoded in the firmware. The robot in our experiment belongs to this type. This type of device is the most vulnerable because once the serial is leaked, it will be leaked forever. There is no way to change it since it's hardcoded. Which mean if cloud server they do not do proper authorization, these devices can be breached.

3.5.2 Vacuum Robot Online Status Penetration Testing Plan

I knew that Proscenic number is hardcoded and use arithmetic number. It means I can increase the last digit of my robot serial number by 1; then, I would get another correct serial number. When the mobile robot app is launched, it will send a robot status checking request. At that time, I captured that request and manipulated the serial number with the newly found serial number. Response from that request would contain the online status of another person's robot.

Similarly, I keep increasing the last digit of the robot serial number by one and replaying the request to check the online status of robots of the entire network. Instead of manually doing all the above steps, I wrote a script to automate the whole process from serial number generation to replay the web request. Then I was able to do a mass online checking on the vacuum robot server without restricting the server.



```
┌──(kali㉿kalim)-[~/Desktop/Vacuum_Robot]
└─$ mitmdump --flow-detail 0 -s mitmReplay.py
Loading script mitmReplay.py
Proxy server listening at http://*:8080
192.168.86.184:35934: clientconnect
192.168.86.184:35936: clientconnect
192.168.86.184:35937: clientconnect
192.168.86.184:35938: clientconnect
192.168.86.184:35940: clientconnect
192.168.86.184:35941: clientconnect
192.168.86.184:35942: clientconnect
192.168.86.184:35941: clientdisconnect
192.168.86.184:35942: clientdisconnect
192.168.86.184:35943: clientconnect
192.168.86.184:35944: clientconnect
192.168.86.184:35944: clientdisconnect
192.168.86.184:35945: clientconnect
192.168.86.184:35947: clientconnect
Online status of robot : LSLDSN7PR020404824 True
Online status of robot : LSLDSN7PR020404824 True
Online status of robot : LSLDSN7PR020404825 True
Online status of robot : LSLDSN7PR020404824 True
Online status of robot : LSLDSN7PR020404826 False
Online status of robot : LSLDSN7PR020404827 False
Online status of robot : LSLDSN7PR020404828 True
Online status of robot : LSLDSN7PR020404829 True
Online status of robot : LSLDSN7PR020404830 True
Online status of robot : LSLDSN7PR020404831 False
Online status of robot : LSLDSN7PR020404832 True
Online status of robot : LSLDSN7PR020404833 True
Online status of robot : LSLDSN7PR020404834 False
```

Figure 8. Proscenic cloud online status hacking

3.5.3 Vacuum Robot Map Sniffing Attack

As mentioned earlier, a vacuum robot with a LIDAR sensor would scan your house and generate a map; then, they would use that map to navigate our house. Most vacuum robots also store your map data in a cloud server. However, as our experiment shows, The Proscenic cloud server did not check robot ownership against the username – robot serial number mapping before accepting incoming requests. This experiment will use the same technique to forge the URL request and retrieve other people's map data. All I needed to do was intercept the web request used for map retrieval, then I manipulated the serial number and replayed the request. The server, without a doubt, responded to me with a house map of another user. At this point, an attacker who knows about this security hole could quickly write a script to pull thousands of map data from the Proscenic Vacuum Robot Owner.

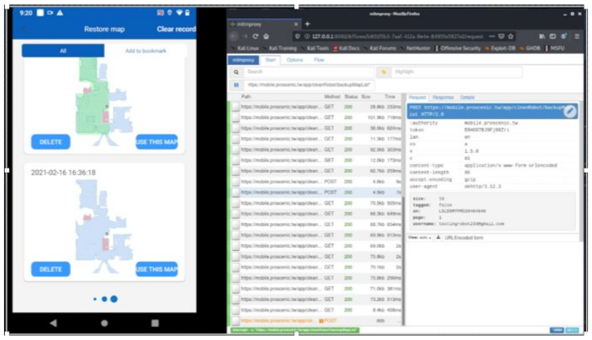


Figure 9. Proscenic cloud map data hacking

3.5.4 Vacuum Robot Stealing Attack

This experiment was about a vacuum robot stealing attack. In the previous experiments, I could check online status or retrieve map data of another robot because the server lacks user-base authorization. In this attack, I used two Proscenic accounts and one vacuum robot device. In the beginning, the vacuum robot belonged to the first account, the real owner's account. I would take the attacker's role and use the URL to illegally bind the robot to the attacker's account and unbind that robot from the victim’s account. The process was to reuse the binding request generated when the user set up the robot first with a different email address. Instead of sending a serial number and the real user's email address to the cloud for binding processing, we replaced the user’s email address with the attacker's email. As predicted, the cloud server just approved that malicious binding attack. The result is the vacuum robot serial number was bound to the attacker's account, then the attacker's phone screen showed that vacuum robot device. When we refreshed the victim's Proscenic app, the robot device disappeared because the victim had already lost ownership of that vacuum robot.

3.4 Evaluation using CIA triads, STRIDE threat model

3.4.1 Violation of CIA triads

From the impact points that I just found, I would like to put those them under CIA triads categories to evaluate the to get the complete picture of possible threats

-Violation of Confidentiality: By successfully guessing the robot’s serial numbers, an attacker can damage the confidentiality principle by gathering information about the online robot status, map data, victims’ home Wi-Fi SSID, local IP address
- Violation of Integrity: Using the same way, I acted like an attacker and was able to change the robot’s name without having ownership of that robot. I can possibly alter the map’s data of a robot to affect its navigation capability. For example, adding coordination to the map JSON to make the robots think an area is a wall and refuse to go there.
- Violation of Availability principle: Because the webserver is not one hundred percent secure, which is proven in this research, an attacker may perform a DDOS attack to bring the service down. Or at least, someone can impersonate a user’s robot, steal the ownership of the robot by binding it to another account or delete the robot’s map to alter its performance. All those kinds of actions would affect the availability of the system by preventing users from using the devices smoothly

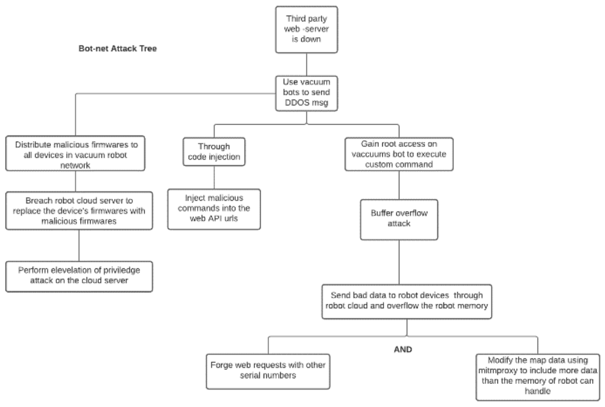
3.4.2 Apply STRIDE threat modeling to suggest mitigations

Microsoft STRIDE models view all threats under six categories: Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privileges. From the vacuum robot system analysis, I applied this threat model to cover all the threats that I found, including other possible threats.

- Spoofing is gaining access to a system by signing a false identity. The attack scenario is an attacker can perform a guess robot serial number to send the command to the robot on behalf of the owner. Mitigation that a manufacturer can do is use a server to randomly generate a serial number and assign it to the robot. In addition, the cloud server should check against the database to see if an owner owns that robot, then authorize the incoming requests.
- Tampering is an unauthorized modification of data. In an attack scenario, an attacker can change the victim’s robot name. Delete, modify map data and activity records. Mitigation for this threat is like what we should have in spoofing. Manufacturers should have a stronger access control
- Repudiation is the ability of users to deny that they performed specific actions or transactions. An attack scenario would be that an attacker may deny using an account with a tool to intercept and forge requests to perform unauthorized requests. Mitigation for that threat is the robot vendor should implement logging features that log username, IP addresses along with corresponding activities
- Information disclosure is unwanted disclosure of private data. An attack scenario would be an attacker can perform an injection, buffer overflow attack to execute privileged commands. Mitigation is protecting API endpoints by validating input, performing security in-dept, least privilege principle.

3.4.2 Attack tree for a bot-net attack scenario

As being said, IoT devices, including vacuum robots, can be impersonated then weaponized to perform an DDOS attack on a third-party server. The following attack tree may give us an overall picture of possible approach that an attackers may take. Although this is not a completed tree, and some approaches have not been tested for usability yet, it is still helpful to be aware of those attack vectors. Furthermore, the graph is well-explained, so I believe there is no need for further explanation.


Bot-net Attack Tree
Third party web -server is down
Use vacuum bots to send DDOS msg
Distribute malicious firmwares to all devices in vacuum robot network
Through code injection
Gain root access on vaccuums bot to execute custom command
Breach robot cloud server to replace the device's firmwares with malicious firmwares
Inject malicious commands into the web API urls
Buffer overflow attack
Perform elevelation of priviledge attack on the cloud server
Send bad data to robot devices through robot cloud and overflow the robot memory
AND
Forge web requests with other serial numbers
Modify the map data using mitmproxy to include more data than the memory of robot can handle

4. Conclusion

After the experiment, we understand that the IoT cloud ecosystem, including the vacuum robot ecosystem, could contain serious vulnerabilities that affect users' privacy and can be weaponized for bot-net attacks. The Proscenia cloud server's access control is weak and incapable of protecting users' data, so anyone who discovered this flaw can take advantage and retrieve users' data massively. The manufacturer could improve the security of their cloud server by enforcing state machines, performing defense in depth, least privilege mechanism. Consumers can protect their privacy by choosing IoT devices carefully. It is crucial to analyze the security of these devices before using them.

My future research would focus more on the robot device entity to learn what other data the robot sends back to the cloud server besides the navigation map data. First, I may research a robot model with a camera to learn how they process the images and where the robot sends them. After that, I will extend the attack trees and verify which approaches an attacker could implement in real life over a robot model.

5. References

[1] A. Bhardwaj, V. Avasthi, and S. Goundar, "Cyber security attacks on robotic platforms," *Netw. secur.*, vol. 2019, no. 10, pp. 13–19, 2019.

[2] F. Ullrich, J. Classen, J. Eger, and M. Hollick, "Vacuums in the cloud: Analyzing security in a hardened IoT ecosystem," in *13th {USENIX} Workshop on Offensive Technologies ({WOOT} 19)*, 2019.

[3] S. Sami, Y. Dai, S. R. X. Tan, N. Roy, and J. Han, "Spying with your robot vacuum cleaner: Eavesdropping via lidar sensors," in *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, 2020.

[4] Zhou, Wei, et al. "Discovering and Understanding the Security Hazards in the Interactions between IoT Devices, Mobile Apps, and Clouds on Smart Home Platforms." ArXiv Preprint ArXiv:1811.03241, 2018.

[5] "Apktool," *Github.io*. [Online]. Available: https://ibotpeaches.github.io/Apktool/. [Accessed: 04-May-2021].

[6] "How mitmproxy works," *Mitmproxy.org*. [Online]. Available: https://docs.mitmproxy.org/stable/concepts-howmitmproxyworks/. [Accessed: 04-May-2021].

[7] "Intercepting Android app traffic - ciko," *Ciko.io*. [Online]. Available: https://ciko.io/posts/intercepting_android_traffic/. [Accessed: 04-May-2021].

[8] J. Reardon, 3. Nathan Good, 3. Robert Richter Vallina-Rodriguez, 5. Serge Egelman, 6. Quentin Palfrey, and 8. Tr-20-, "JPush away your privacy: A case study of jiguang's android SDK," *Berkeley.edu*. [Online]. Available: https://www.icsi.berkeley.edu/pubs/privacy/TR-20-001.pdf. [Accessed: 04-May-2021].

[9] A. Bhatia, *android-security-awesome*. .