Home IoT Devices Security Assessment

Team HackBerryFi
14-829: Mobile and IoT Security
Archit Agarwal, Dong Bao, Haran Sivaram, Samuel Sabogal, Xin Huang
Carnegie Mellon University
{architag, baod, hsivaram, ssabogal, xhuang2}@andrew.cmu.edu

Abstract

IoT devices have grown significantly each year, but their security issues have been largely ignored. In this paper we analyze several popular IoT devices in the market. Reverse engineering, static analysis, and dynamic analysis techniques are used to understand each of the components of the devices and expose several vulnerabilities in terms of bad authentication implementation, authorization, information leakage and insecure pairing protocol. Recommendations to fix these vulnerabilities are explained, as well as the tradeoffs in security and challenges in the assessment.

1. Introduction

In the HackBerryFi team we analyzed a set of home IoT devices currently sold in the market to assess their security and determine if the users of this kind of devices are being exposed to risks that they ignore.

Additionally, finding vulnerabilities on the devices can help to create security awareness on the manufacturers and alert their strategic managers about the products that are being sold.

The use of home IoT devices is increasing with each day that passes. Users acquire them without any knowledge of the security risk that the devices might bring to their sensitive information or privacy. Our hypothesis as the HackBerryFi team is that the security of such devices is not good enough. Manufacturers that do not take security seriously need to be exposed to protect people's privacy and sensitive data. A compelling reason for manufacturers to ignore security, is that from a business perspective it is not a feature that the average user wants and would pay more for it, it is possible that is a tradeoff in which the company takes the side of making more money by doing less effort. We believe that security awareness in users can be a factor that could push manufacturers to develop more secure products. In contrast, cybercriminals could be very aware of the vulnerabilities of the devices, which makes them an important adversary to consider. By leaking user data from vulnerable devices they can get sensitive information for fraud or identity theft.

The security assessment was done by understanding thoroughly the functionality of the devices by doing reverse engineering on each of their components. Additionally, dynamic and static analysis, both manual and automated, was done to find vulnerabilities. The vulnerabilities found by automated tools were

tested manually to eliminate the high number of false positives reported by automated means.

High level goals for the project can thus be summarized as below:

- Look at how IoT devices pair with smartphones and the network and look for potential flaws
- Look for both insiders as well as external attacks
- Firmware level vulnerabilities
- IoT device's web console application security vulnerabilities
- Static code analysis of the available firmwares and decompiled android applications

1.1 State of the Art and Motivation

As of today IoT device security has not matured much beyond the primitives that were put in place for traditional computing systems. This is troublesome in cases where the user interaction was an integral part of the security which is lost with the ubiquity of most IoT devices. Existing research illustrates multiple attacks on different surfaces of the IoT ecosystem. The attacks in [2] and [8] focus on network based attacks on the pairing phase for these devices, while [3] and [4] focus on firmware and application reversing to extract embedded keys.

Certificate pinning for SSL, DTLS for SSL over UDP, TPM storage, key distribution and mutual authentication are among the top fields being researched and developed as possible solutions to secure the next generation of devices. The motivation for our project is inspired from such imperfections in the IoT ecosystem. As a general rule, we believe that any authentication based protocol can be prevented from MiTM attacks till the time it incorporates proper authentication mechanisms, freshness as well as proper access controls. This hypothesis is proved in our analysis as all the 4 devices tested lacked proper authentication implementation leading to different types of internal and external attacks. The findings discussed in the later sections discusses all these aspects and we conclude by confirming our hypothesis.

2. Analyzed Device

A set of home devices from different manufacturers was analyzed. The following section illustrates the architecture and vulnerabilities found on each of them. Additionally, challenges and recommendations are explained.

2.1 Uniden Appcam 23

Uniden's AppCam is a system that allows a user to watch on real time video recorded by a camera. The user can watch it on an Android or IOS mobile application. The Camera needs to be connected to a Wifi network and the user's smartphone can have any internet connection. For this assessment it will be excluded the analysis of the IOS application.

2.1.2 Architecture Overview

After the analysis of the Uniden AppCam system, the following components were identified to be deployed and function in the following manner:

1. Camera: It communicates with the mobile apk using a UDP connection encrypted with AES 128. In this connection the video is sent directly to the Mobile App.
2. Mobile Android App: It communicates with the camera using a UDP encrypted

connection. Additionally, it has an encrypted UDP connection with three remote servers to send information when a configuration is changed. The encryption library in the APK is implemented on ARM native code.

3. Remote servers: Three remote servers were identified:
    - 23.21.195.143, Ashburn, Virginia, Amazon
    - 122.248.232.207, Singapore, Singapore, Amazon
    - 176.32.104.236, Dublin Ireland, Amazon

2.1.3 Finding

Several vulnerabilities were found. They were not related to decisions in trade offs faced by the developers, but they could be interpreted as accidents in coding or lack of understanding in the technologies used. In the following subsections each vulnerability is explained

2.1.3.1 Logged Secrets (High)

The first type of vulnerability present in several parts of the code was the logging of secrets on the Android application. For debugging purposes, developers print information to the logcat of Android. An attacker, can decompile the APK to obtain the JAVA source code, and using search text functions or regular expressions can find all the places in which information is being printed to a log. Then, a malicious application can read all the sensitive information if it is installed in the user's smartphone. In this case, three pieces of sensitive information were being output to the log:

1. The Wifi password, which is required to be sent from the APK to the camera to allow the camera connecting directly to the WiFi network.
2. The password to authenticate to the camera
3. The configuration commands send to the camera. This would allow the malicious app to learn any configuration.

To validate these vulnerabilities, search for the following strings on the decompiled java source code:

- `System.out.println("connect wifi-apSSID="+str+"password="+str2+"success="`
- `System.out.println("Ex_IOCTRLPlayRecord, playCmd="`
- `System.out.println("IOCTRL_TYPE_SETWIFI_REQ, pwd=" + str2);`

2.1.3.2 Logged Secrets (Critical)

The second type of vulnerability was the use of HTTP protocol. It was found an HTTP API used by the application to register the camera and do remote operations such as sending an email. An attacker can change the email configuration using this API. The arguments to do it were found while reversing the firmware of the camera. On the other hand, the firmware could be downloaded from the official web site of the camera. Even though the website is using HTTPS, the link to the firmware is using HTTP. There is an HTTP connection inside of an HTTPS website, a fact that allows an attacker to tamper the firmware. To validate this vulnerability it can be searched on the decompiled source code for the following string that contain the first argument of the HTTP link:

- `apns.php?cmd=reg_client`

2.1.3.2 DoS (Medium)

The last type of vulnerability was DoS network attacks. With enough computing power, it is possible to DoS any network connection. However, if a DoS attack can be executed with very little computer power it becomes a vulnerability to consider. In this case, it was possible to use a single laptop to DoS the camera and preventing it from reporting real time video. To validate this vulnerability, run hping3 on kali linux and point to the UDP open port from the smartphone that receives data from the camera.

2.1.3 Recommendation and Trade Off

The recommendations to fix these problems are relatively straightforward. It is recommended to include in the code review process that automatically changes for printed strings to the logcat. This can be done with a simple script. For the HTTP connection, it should be installed digital certificate on the web server. Lastly, for the DoS attack, in this particular scenario in which the app is connected to only one camera, not more connections should be open if they come from different ports as the already connected port. It was verified on this device that launching the attack only from the same IP and port using a single laptop did not cause any effect on the device. Note that this is just a mitigation, and with more computing power DoS can be successful. In some cases the tradeoff between security and cost can make the managers decide to ignore security. However, in the particular vulnerabilities found on this device, the fixes are not expensive and easy to implement. They should be applied.

2.2 TP Link NC200

Yet another contender in the smart home camera space is the TP Link NC200. The camera features sleek design, WPS for easy installation, a local web interface as well as a mobile application that can connect to the camera remotely. The camera uses a clever reverse TCP connection with its server to bypass NATs on home networks without having the user to set up port forwarding. While a lot of care has been taken to secure remote connections, the camera fails to provide any measure of security within the connected Wi-Fi network. Though this might have been by design, these cameras are used as security cameras for coffee shops or offices where there are multiple users connected to the internal network that are not necessarily trusted by the owner of the camera. This accompanied by a badly designed authentication protocol gives rise to password leaking vulnerability during daily use of the camera.

2.2.1 Architecture

Initial connection set up is done over an open Wi-Fi AP opened by the camera that includes an HTTP FCGI interface to configure home wifi information. This period leaves the camera vulnerable to any eavesdropper who could potentially gain access to the user's home wifi. Upon connection the camera opens up 2 ports permanently that act at the web server (80) and web proxy (8080). It also opens port 1068 for both TCP as well as UDP connections.

The UDP port is used to send and receive broadcast messages in the local network for the discovery phase of the pairing between an app that has just come online and the camera that is already connected. The UDP message sent by the camera contains the IP address of the camera which is then used by the app to connect to its

FCGI interface. The same port number for TCP is used to create an SSL connection to the TPLink cloud server.

The camera comes with a pinned certificate for the TP Link's certificate authority. This is effectively prevents any sort man in the middle attacks involving SSL stripping or certificate spoofing. The app also uses a pinned certificate in it's source code that prevents certificate spoofing even if trusted explicitly by the device owner. The camera periodically checks in with the server and whenever a user requires remote access, the two SSL connections are bridged.

However in the local Wi-Fi, the app and the web interface use the same HTTP based FCGI API. While the login API is essentially a plain text request that can be read by any internal sniffer, the response includes a session cookie for future authenticated requests, as well as a token that is used for more critical functions like changing passwords.

2.2.2 Finding

2.2.2.1 Password Leakage (High)

The developers decided that local streaming should require constant authentication as opposed to using the pre-established token from a previously successful login. Streaming uses the GET /stream/getVideo API on both the app as well as a web browser to access the camera feed over the local Wi-Fi. The request includes an authorization token that is generated by concatenating the username and password and encoding them in Base64. Any device connected to the internal network can decode the message and obtain administrator access to the camera. Since the password was leaked and not just a currently active session token, an attacker can lock the user out of his own device or use it to gain remote access after the attacker has left the internal network. The below expression shows the authorization header format for these requests.

```
Basic Authorization:(username:(password))
```

The parenthesis indicate Base64 encoding in the above expression.

2.2.2.2 SYN-Flood DDOS Attack (Medium)

The camera fails to provide basic protection against SYN flooding based resource exhaustion attacks. Using a single host sending out SYN packets to the cameras web interface overflows the camera's buffer and prevents any future connection using local port as well as the remote connection that runs on a seperate port. This also blocks the camera from communicating with its configured cloud storage for video recording. This vulnerability is a concern when these cameras are used for surveillance and security purposes. An insider could easily launch this DOS attack and render the security feature useless.

2.2.3 Recommendation and Trade Off

As a start the developers must include a SYN cookie for the TCP connections. SYN cookies prevent the need of local storage to maintain TCP connection state by having the clients resend a dynamically regeneratable token issued based on a private key on the web server. This will stop the easiest of DDOS attacks and will stop weaker devices from being able to affect this device.

The developers should be using the same pre-established tokens for streaming videos as well. While this would enable an insider to access the video feed, it would not give them

access to the camera's administrator settings nor will it allow the attacker to obtain remote access to the camera. An insider would always be able to access the video stream even without the tokens, since the data generated by the camera is sent over the clear and only the encoding algorithm used is required.

Better security guarantees can be achieved by having the camera encrypt local wifi traffic as well. The performance of bridging 2 SSL connections gives a fair indication that symmetric key based local encryption would not bottleneck performance. The existing certificate on the device can be used to set up a secure key distribution mechanism with the TP link server and have the mobile application only trust devices using keys distributed by their cloud server.

2.3 TP Link HS100 Smart Plug

TP link smart plug HS100 is an easily available smart plug that allows to control electronic devices connected to it using android and iOS mobile applications. This cloud enabled power plug has features like scheduling on/off, integrates with smart voice assistants like Amazon alexa and other IoT devices in a bundle as well as away mode.

Connecting this smart plug requires user to register and set up an account on TP link Kasa mobile application. Next, the smart plug pairs with the mobile application and the mobile device can then control the smart plug. The condition here is that both the devices need to be connected to the same WiFi network while pairing.

2.3.1 Architecture Overview

The smart plug HS100 architecture is an archetypal of cloud based IoT architecture involving the following components:

- Android and iOS application that connects with the smart plug connected to the same internal network.
- Smart Plug physical device - the device connects to the power socket directly and has reset button as well as LED that indicates the working of the plug.
- The cloud services - these services enable the plug to receive commands via mobile phone when its not connected to the internal WiFi network. There are multiple commands that can be operated from switch like firmware upgrade, authenticate to cloud account etc.

2.3.2 Finding

Decompilation and analysis of the Kasa android app displayed the use of Autokey cipher as shown in figure 1. The messages are encrypted using a fixed IV value and iterative xor operations. The cipher messages were analysed and collected using sniffing packets in monitor mode through wireshark. It was also noted that the TCP port used for communication is 9999.

```
public static byte[] m7377b(byte[] bArr) {
    if (bArr != null && bArr.length > 0) {
        int i = -85;
        for (int i2 = 0; i2 < bArr.length; i2++) {
            byte b = (byte) (i ^ bArr[i2]);
            i = bArr[i2];
            bArr[i2] = b;
        }
    }
    return bArr;
}
```

Fig 1: Autokey cipher

A simple python script as shown in the below figure led us to decrypt the messages sent from the app to the smart plug.

```
def decrypt(string):
        key = 171
        result = ""
        for i in string:
                a = key ^ ord(i)
                key = ord(i)
                result += chr(a)
        return result
```

Fig 2: Script to decrypt cipher

The list of few of the commands decrypted are as follows:

- **On:** {"system":{"set_relay_state":{"state":1}}}
- **Off:** {"system":{"set_relay_state":{"state":0}}}
- **Reboot:** {"system":{"reboot":{"delay":1}}}
- **System Info:** {"system":{"get_sysinfo":null}}
- **Turn Off LED:** {"system":{"set_led_off":{"off":1}}}
- **Check config:** {"system":{"check_new_config":null}}

Now these decrypted messages can be leveraged to perform a **replay attack**. Anyone connected to the same WiFi network (hence an internal attack) just needs to identify the IP address of the live smart plug. This is an optimistic assumption since an attacker can port scan the internal network to identify the running services and identify the connected smart plug by looking at TCP port 9999. The attacker then has to replay the captured messages in encrypted form and the smart plug should behave as per the demands of the attacker. The attack map is shown in the below figure 3.

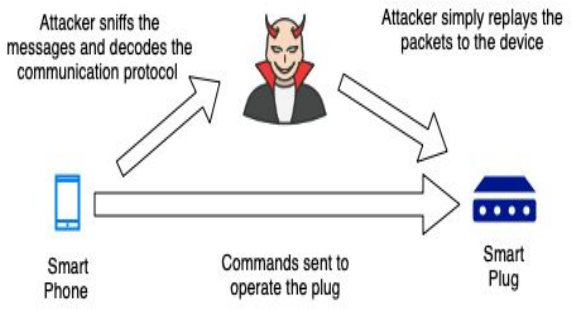

Fig 3: Replay attack

The TP Link HS100 smart plug communication protocol has been reversed by SoftCheck [3]. Here are the reasons why the TP Link HS100 smart plug is vulnerable to replay attacks:

1.There is no form of authentication token used in the messages, thus an attacker can directly replay them without knowing any legitimate user credentials.

2.There is no form of random token or nonce part of the communication protocol, thus there is no freshness property in the protocol making it vulnerable to replay attacks

3.Weak encryption ciphers like Autokey is in use which is easy to reverse engineer. The decryption of the protocol messages gives the command sent to the smart switch in clear-text which can then be replayed as well as manipulated to achieve malicious results.

2.3.3 Impact of the Vulnerabilities

The replay attack as discussed in the previous section can be categorised as a **"High"** severity vulnerability. The severity is determined by considering the likelihood and impact associated with each issue.

In this scenario, the impact is *critical* since the attacker can control the device remotely, thus defeating the core purpose of the device. An attacker can turn On/Off, reboot as well as gain

information about the connected network as well as the device leading to privacy loss.

Likelihood can be considered as *Medium* since it is assumed here that an attacker has access to the WiFi to which smart plug is connected. Hence, it can be termed as an insider attack. However, external WiFI attacks like KRACK makes the device vulnerable to external attacks as well.

Successful execution of the attack can lead to:

- Loss in availability
- Unauthorized operations
- Unauthenticated operations
- Privacy loss (Confidentiality)

2.3.4 Recommendation and Trade Off

It is recommended that the communication protocol used between the mobile application and the IoT device must ensure to have:

- Authentication token - to prevent from unauthenticated users
- Access controls - to prevent non-admin users to perform unauthorized operations i.e. vertical privilege escalation.
- The protocol must incorporate some form of random token to prevent replay attacks. These random tokens or nonce should be validated at the device end or by leveraging the cloud APIs.

Implementing any form of authentication tokens would require server to do more processing. This may be a bottleneck since IoT devices are generally low powered and less processing capability devices. However, the impact of the attacks possible due to these weaknesses outweigh the effort or business cost associated in implementing these changes. As specified in the last recommendation, the tokens can be verified using already incorporated cloud API's to outsource the processing and validation of users and incoming commands.

2.4 Wemo Smart Plug

The WeMo smart plug is an IoT device manufactured by Belkin and branded as WeMo. It has to be operated by a mobile application to remote control, allowing controlling the switch either from LAN or from WAN. Users set up the device by connecting it to their local WiFi hotspot, which allows remote controlling from LAN. By enabling remote control from WAN, users can access their device even if their mobile phone isn't in the same LAN.

In this project, we are mainly focusing on the pairing protocol to give the device access to user's local WiFi hotspot.

2.4.1 Architecture Overview

The pairing protocol of the WeMo smart plug consists of the following steps:

1.Turn on the pairing mode on the device and the device will start broadcasting an unencrypted pairing hotspot.
2.The user connects to the pairing WiFi using her smartphone. Then the user opens the WeMo application, select the home WiFi and enter the credential.
3.The WiFi SSID and password will be sent to the WeMo device. Then the device can connect to the user's home WiFi and stop the device pairing hotspot.

Figure 2.4.1 shows the process of the pairing protocol.

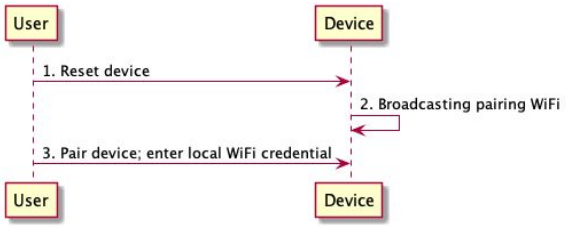

Fig 4: Pairing protocol of WeMo smart plug

2.4.2 Finding

We find that this pairing protocol is vulnerable that given the following attacking scenario. First the user is unaware of the existence of the attacker. Second the user is going to reset the device if she finds the device stops working.

The attack can be carried out in the following steps:

1.The attacker identifies the WeMo device and launches deauthentication attack.

2.Because the deauthentication attack, the device appears to be not working to the user. Therefore the user reset the device and repair it to home WiFi.

3.When the user is repairing the device, the device broadcasts the pairing WiFi. The attacker stops the deauthentication attack after the pairing WiFi is observed and start sniffing data transmitted over the pairing WiFi.

4.When the user enters home WiFi credentials, it will be stolen by the attacker.

Figure 2.4.2.1 shows the steps of the attack.

For the attacker, she has to answer the following questions.

1.How to find the victim device? To find the victim device, the attacker can sniffing wireless traffic and translate client MAC address into their manufacturer's identifier using the Organization Unique Identifier (OUI) database. The WeMo devices' MAC addresses always start with "belkin". And Belkin's manufactured devices include smart plugs, routers and wireless dongles. As all the Belkin smart plugs use the same pairing protocol, they are all vulnerable to this kind of attack. The routers can be distinguished by the fact that they usually broadcast encrypted WiFi hotspot rather than unencrypted ones. The wireless dongles can be distinguished from that their states usually changes more often than the smart plugs because they are often move around. The wireless dongles can also be distinguished by the amount of data transmitted as smart plugs are usually more silent.

2.How to know when to stop the deauthentication attack? The attacker should keep monitoring for unencrypted wireless hotspot of which the MAC address can be translated into OUI starts with "belkin". From our observation, we also find that the device hotspot MAC address is the device WiFi client MAC address plus one. This can also be used to tell if the device enters pairing state when the attacker sees such an unencrypted WiFi hotspot.

3.How to extract WiFi credentials from the intercepted traffic? According to the work from Liu et al., the WeMo devices encrypts the password using AES, as shown in Figure 2.4.2.2 [1]. The encryption/decryption key is derived from the serial number and MAC address of the device. While the hotspot SSID, serial number and MAC address are transmitted during pairing in plaintext, the attacker can easily decrypt the WiFi password and gain access to the user's WiFi.

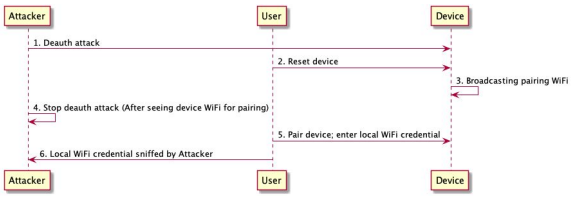

Fig 5: Attacker steals the WiFi credential

The following figure shows the pseudocode of WiFi password encryption logic.

```
function encrypt(WiFiPassphrase):
  Password <- MACAddress
      [0:6] : SerialNumber : MACAddress[6:]
  Salt <- Password[0:8]
  IV <- Password[0:16]
  SymmetricKey <- MD5(Password, Salt)
  CiphertextBits <- AES_CBC_PKCS5Padding
      (WiFiPassphrase, SymmetricKey, IV)
  return Base64_Encode(CiphertextBits)
```

Fig 6: WiFi password encryption logic

We also checked all the devices listed for sale on Belkin's website. It turns out all the wireless enabled IoT devices, mainly smart plugs, are using the same pairing protocol and therefore are prone to such an attack. Besides, there are other IoT devices that uses unencrypted WiFi for device pairing, such as the TPLink camera we analyzed.

2.4.3 Recommendation and Trade Off

Our recommendation to fix the issue in using unencrypted WiFi as part of pairing protocol is to use an encrypted WiFi instead. The password should be generated randomly for each device. Asking the user to enter a random generated password isn't very user-friendly, but it provides better security and prevent the user's WiFi credentials being stolen in this attack. Besides, this setup should happen very frequently. The password can also be generated from commonly used English words, which makes entering the password easier. But this method doesn't provide enough randomness in passwords, and can still be attacked by brute-force decryption of the intercepted password.

3. Conclusion

We explored weaknesses in authentication protocols, resilience to insider and outsider MiTM attacks, and implementation flaws in the firmware and APKs for the four IoT devices. We observed that all the devices incorrectly implemented authentication protocols. We can also conclude that these devices trust the internal network, which makes them vulnerable to insider attack. For example, simple DoS attack coming from a simple laptop can make IoT devices lose availability. Additionally, it was shown enough evidence to claim the home IoT ecosystem is insecure and users should be concerned about their privacy and sensitive information that cybercriminals can use for identity theft or fraud.

4. Challenges

There were multiple challenges faced by our team while evaluating the 4 IoT devices for potential MITM attacks and other vulnerabilities. Certificate pinning, though is a good security step, restricted us to simulate rogue access point and related MiTM attacks. Another challenge was while performing static and dynamic analysis of the different firmwares - firstly all the firmwares were updated and secondly, reversing the code and native libraries turned out to be requiring larger time and effort.

5. Future Work

The future work involves analysing in detail the device's firmware for static and dynamic analysis. Additionally, some of the native drivers were not reversed completely. Developers sometimes use obfuscated native

code to protect communications, so this is an interesting part to reverse in more depth. Our project timelines also restricted us from looking at the physical penetration testing of the devices. Using UART, SPI, or JTAG protocols to access the devices physically might be a promising path to take.

In this project we specifically looked at insider and external threats leading to MiTM attacks. The future work would also involve designing the threat model and analysing Android and iOS components of the IoT ecosystem along with trying firmware downgrade attacks.

6. References

[1] Liu, H., Spink, T., & Patras, P. (2019, March). Uncovering Security Vulnerabilities in the Belkin WeMo Home Automation Ecosystem. In 2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops) (pp. 894-899). IEEE.

[2] Mohit Sethi, Aleksi Peltonen, Tuomas Aura (2019, May) Misbinding Attacks on Secure Device Pairing and Bootstrapping, Cornell University Library, arXiv.org

[3] Reverse Engineering the TP-Link HS110. (n.d.). Retrieved December 11, 2019, from https://www.softscheck.com/en/reverse-engineering-tp-link-hs110

[4] The risks of IoT: Internet-connected LED light bulb vulnerable to hacking. (2014). *Networks Asia,* Retrieved from https://search-proquest-com.proxy.library.cmu.edu/docview/1543514176?accountid=9902

[5] Matheus-Garbelini. (2019, September 8). Matheus-Garbelini/esp32_esp8266_attacks. Retrieved from https://github.com/Matheus-Garbelini/esp32_esp8266_attacks

[6] Y. Khamayseh, B. Y. Muneer and M. Abu-Jazoh, "Intelligent black hole detection in mobile AdHoc networks," International Journal of Electrical and Computer Engineering, vol. 9, (3), pp. 1968-1977, 2019. Available: https://search-proquest-com.proxy.library.cm.u.edu/docview/2201017555?accountid=9902

[8] Khalid M., Mujahid U., Najam-ul-Islam M., Tran B. (2020) Probabilistic Full Disclosure Attack on IoT Network Authentication Protocol. In: Arai K., Bhatia R. (eds) Advances in Information and Communication. FICC 2019. Lecture Notes in Networks and Systems, vol 70. Springer, Cham