Vulnerabilities in IoT

Krystle Bates, Rhea Shetty, and Kerin Shah

*Illinois Institute of Technology, Chicago, Illinois*

***Abstract*— With the emerging technology of the Internet of Things (IoT), more and more smaller devices are connected into the Internet for monitoring and control purposes. This paper introduces the vulnerabilities in Internet of Things (IoT) devices. Generally, manufacturers are not focussed about the security when introducing new product in the market. This can be proven by the current research using simple methods and devices to obtain sensitive information. By analysing the collected data, we will focus on major attacks on IoT devices and conclude the pros and cons of a different IoT protocols. Our main focus is on smart plugs, smart bands and alexa which we are planning to build on Raspberry Pi 3.**

1. Introduction

The "Internet of things" (IoT) is becoming an increasingly growing topic of conversation in the world. The future of technology is heading towards, "Anything that can be connected, will be connected." Statista forecasts that the total number of connected devices will reach nearly 27 billion by the end of this year, and 75 billion by 2025. In a 2015 report, Ericsson projected that there will be 29 billion connected devices and 18 billion IoT devices by the end of 2022. [2] With billions of devices being connected together, the IoT also opens up companies all over the world to more security threats.[1] The *OWASP Internet of Things (IoT) Project* lists the top 10 vulnerabilities in IOT as follows:

1. Weak Guessable, or Hard Coded Passwords
2. Insecure Network Services
3. Insecure Ecosystem Interfaces
4. Lack of Secure Update Mechanism
5. Use of Insecure or Outdated Components
6. Insufficient Privacy Protection
7. Insecure Data Transfer and Storage
8. Lack of Device Management
9. Insecure Default Settings
10. Lack of Physical Hardening

The Mirai botnet took the Internet by storm in late 2016 when it overwhelmed several high-profile targets with massive distributed denial-of-service (DDoS) attacks. The botnet was made mainly of embedded and IoT devices. It exploited the above mentioned vulnerabilities in IoT devices to carry out the attack. The attack changed IoT security altogether.

1. Problem Statement

With the vast never-ending supply of internet of things (IoT) devices, the thought of security is largely considered an afterthought. Rushing items to market opens up vulnerabilities within the devices due to the lack of penetration testing and evaluation. In addition to the security concerns issues, further issues such as designing, handling, and storing data are overlooked components.This project will explore the potential vulnerabilities in consumer IoT devices, exploit an IoT device and design possible solutions for them. We plan to use physical devices, simulation software and Raspberry Pi for our evaluation.

1. Prior Work

‘Data Security and Privacy for IoT Systems’ by Elisa Bertino talks about key challenges in data security and privacy[6]. Her talk regarding her work inspired us to conduct our own research in this domain.

In ‘A Study of Data Store-based Home Automation’, the authors have performed a systematic security evaluation of two popular smart home platforms, Google’s Nest platform and Philips Hue, that implement home automation “routines” (i.e., trigger-action programs involving apps and devices) via manipulation of state variables in a centralized data store. [3] A firewall based defense architecture for IoT devices has been proposed in ‘An Approach to Secure Smart Homes in Cyber-Physical Systems/ Internet of Things’ [4] Also, we referred to the paper, “Spying on the Smart Home: Privacy Attacks and Defenses on Encrypted IoT Traffic”. This piece described several strategies for mitigating the privacy risks associated with smart home device traffic, including blocking, tunneling, and rate-shaping with the help of Raspberry Pi 3 [5].

In our further research we discovered an author’s conducting a similar approach as to us. They analyzed the communication between a Google Home (device), Google Chrome Browser (paired web application) and the author

(user). The user would interact with the paired web application, and then verbally request a command for the device to perform.

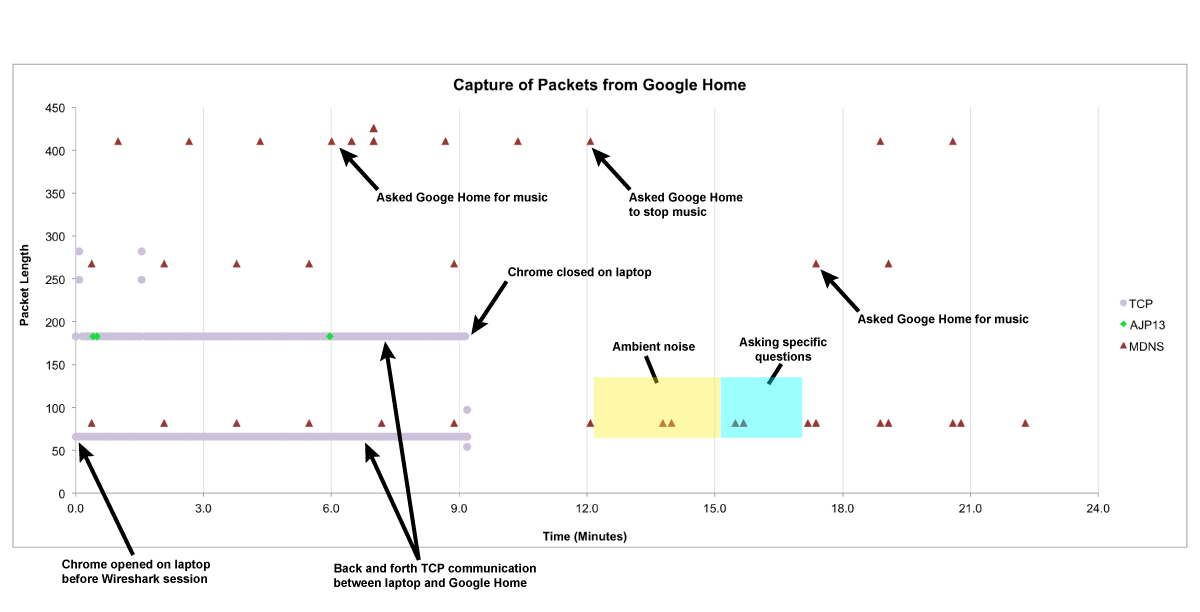


Figure: Diagram of packets being captured from a Google Home

Within the research, the author was able to see the packets being sent to and from their Google Home and browser. One observation that they made was the PSH or push flag. With the push flag notification, it tells the sender that the TCP data should be sent right away and it tells the receiving host to immediately push the data to the application. Without the PSH flag, the packet may sit in the buffer and cause a delay.[11]

Through this observation, we were taking it into account that our devices may potentially follow a similar format of communication.

1. Research Approach

Within our research, we wanted to be able to evaluate or approach multiple items in the IoT realm. This was due to the fact that IoT devices range in their scope of service and communication. We understand that one could argue that this would dilute the value of our research since it is broad. On the contrary, we wanted to be able to test a broad spectrum of items so that our scope or understanding of IoT devices would not be limited in just one device’s understanding. We decided to look at items that ranged including a smart TV, smart fitness tracker, and two different smart outlets.

At the moment of our research, we are still in the discovery phase. We were able to discover ways to initiate communication amongst devices and applications. At first glance, it was a challenge to find out how the have the devices send a signal on demand, since for instance, smart bands collect information in a passive way regarding the users actions and environment. However after further investigation, we were able to trigger communication between a phone application and the smart band and vice versa. For the first option, we were able to have the phone application “find the device”. When a device is found, a signal is sent to have the smartband vibrate. For the latter option, when the phone application is open on it’s camera mode, it listens for the smartband to send a signal to take a photo when the smartband is shaken. With these options, we were able to use these signal to track communication on demand.

For the smart TV, were not able to get much information about it but we were able to perform nmap scan for the open ports. Also we tried intercepting the traffic between the smart device and internet through a software, Burpsuite.

For the smart plugs, we were able to evaluate two different plugs. The plugs were manufactured to evaluate power of a 220v and a 110v device manufactured by and paired with different applications. We wanted to see if any of these small changes would matter in our research.

Although not mentioned in our results, we are also looking into simulating a doorbell on Raspberry Pi. Even though we were unable to report any findings at this moment, we thought of mentioning our additional research. We wanted to see if a simulation would also be helpful in discovering vulnerabilities in our research. As mentioned before, we are looking into a general and encompassed discovery phase to help evaluate our research and understanding of IoT devices.

1. Research Results

The first item that was evaluated was the smartband. This brandless item is charged by a USB and communicates via Bluetooth through a phone app, Yoho Sports. The device was evaluated by three different applications. The first application was WireShark. Within that application, a bluetooth communication was unable to be detected. Through research, this may have been due to the lack of a bluetooth dongle device. The computer that was being used, lacked bluetooth specific hardware to detect the communication.

The next application that was used to evaluate the device was Bluetooth LE Explorer. This software is a Windows specific service that is used to debug Bluetooth communication between a device and a Windows machine. In the beginning communication was detected. However, after working with the device and the application, it was later discovered that it was a one-to-one communication. This hindered the ability to be able to pair the device with the native application and learn about the communication between the device and the application.

Then we were able to use the phone application nRF Connect. Although designed to be paired with Bluetooth dongle, it was able to detect the communication between the smartband and the phone application. This allowed us to be able to observe the communication that is initiated between both the device and the phone application.

The first action that we observed was the manual trigger of the smartband being found.

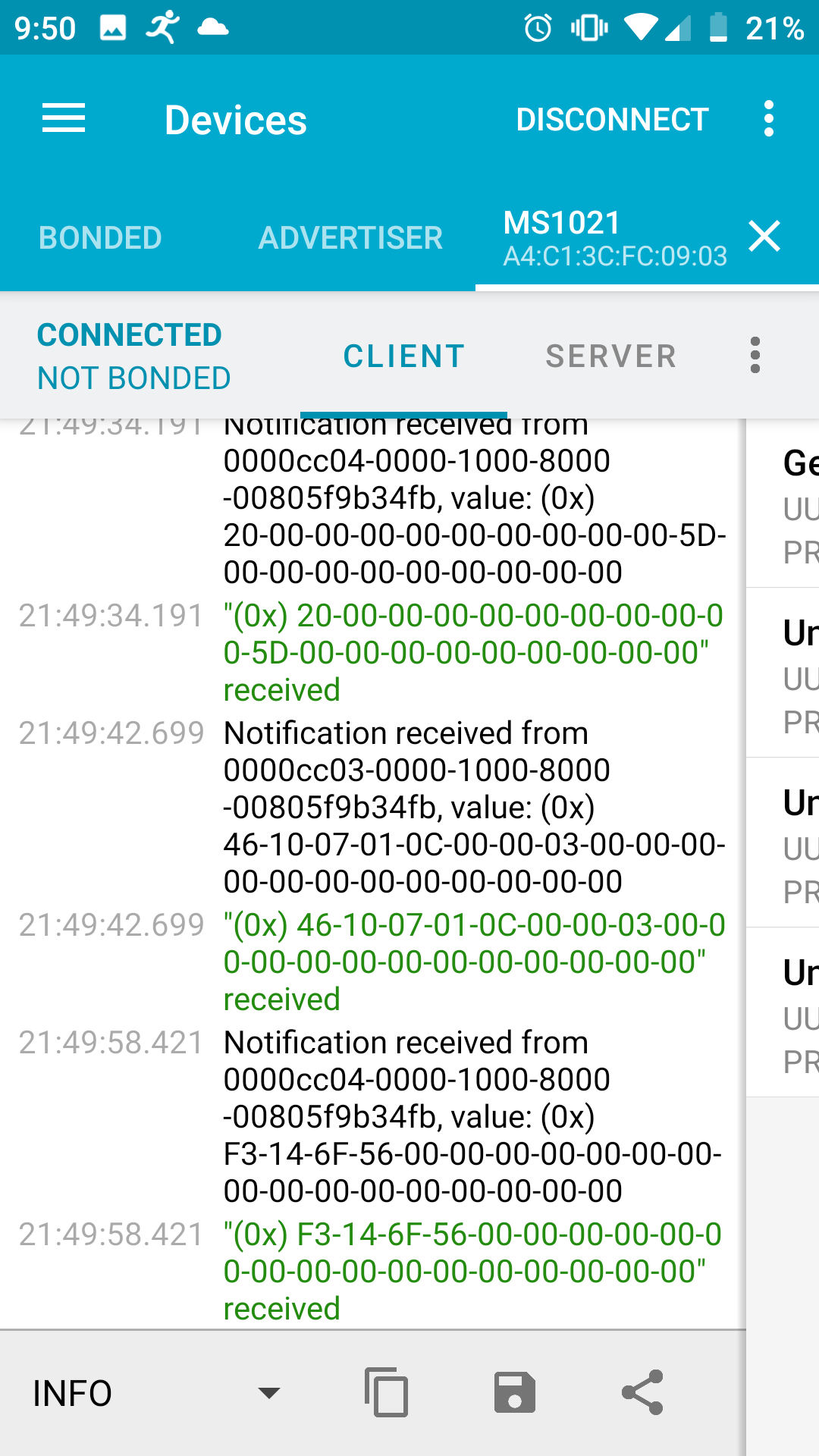


Figure: nRF Connect display showing the notification of the device being “found”

With the above display, there are multiple “notifications”. These notifications mark the phone application searching for the device with a signal sent to the device to vibrate indicating that it has been found. We were able to repeat this action to ensure that the notification was from the “find my band” option on the phone application. After that, we then tested the communication between the band and the phone application. We had the phone application open to the camera option that allowed the user to shake the smartband to initiate the camera to capture a selfie.

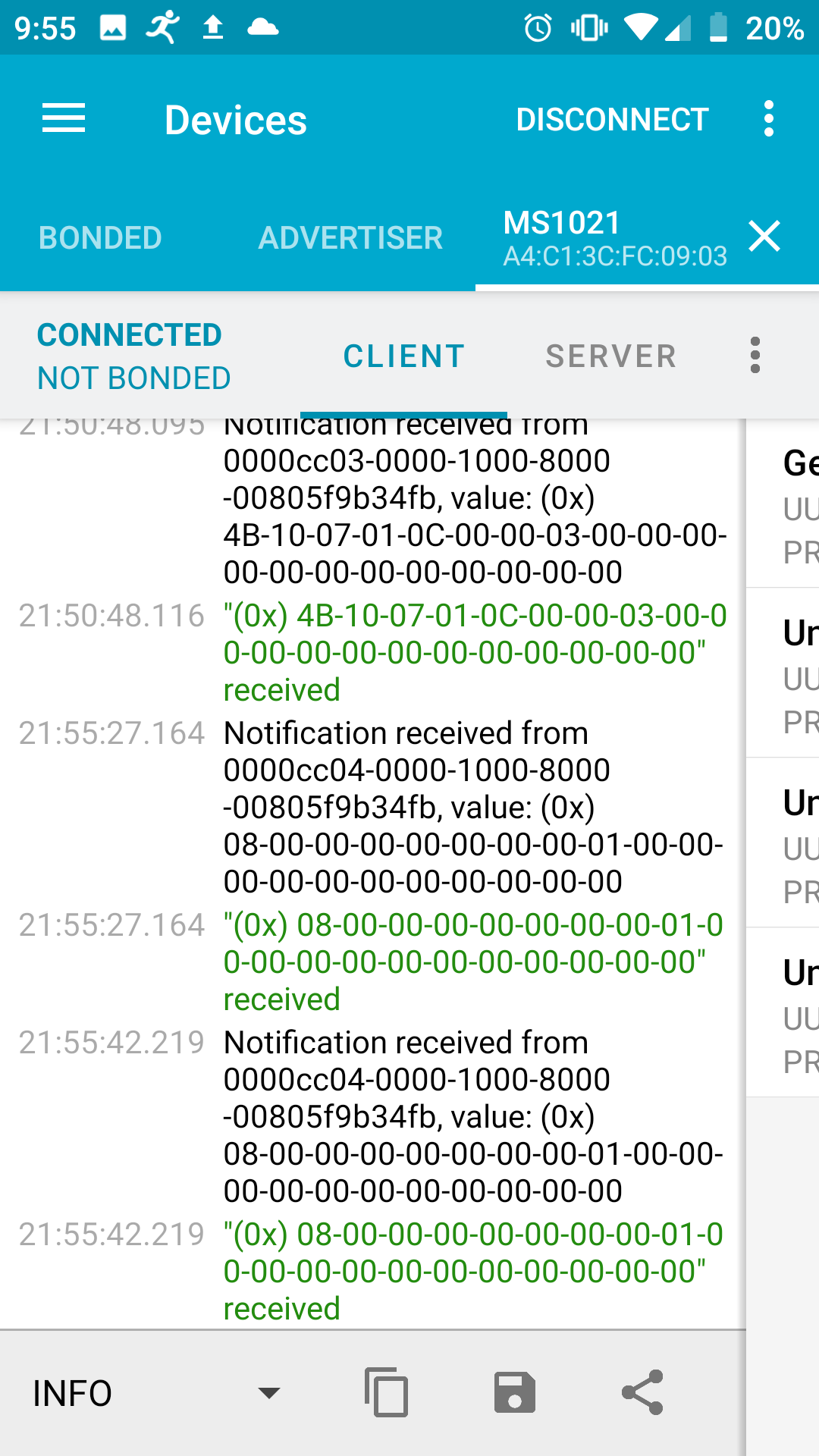


Figure: nRF Connect display showing the phone application receiving a notification or signal to indicate to the phone to capture a selfie

We were able to open the application, shake the band and have the phone application take a selfie. We are still trying to decipher what each of the signals or notifications were for. At this stage of our research, we are still understanding all of the communication that is occurring amongst the device and the paired phone.

The second pair of devices we evaluated were the two smart plugs of a different brands: Merkury and an unbranded plugs from the Chinese manufactures. For the smart plugs, the corresponding phone applications used by device were Geeni and Smart Life respectively. Nessus scanner was used in Kali Linux for the web application scanning, this which provided basic understanding of the application functionality and associated vulnerabilities. Next, we used OstorLab for mobile based application scanning. This provides a report that offers basic details about the application and the list of vulnerabilities related to it. The vulnerabilities in the application are divided into three levels: High, Medium and Low. The scanner presents an information section which includes basic details about its plugin, ping remote host, etc. This is shown in the below figure:

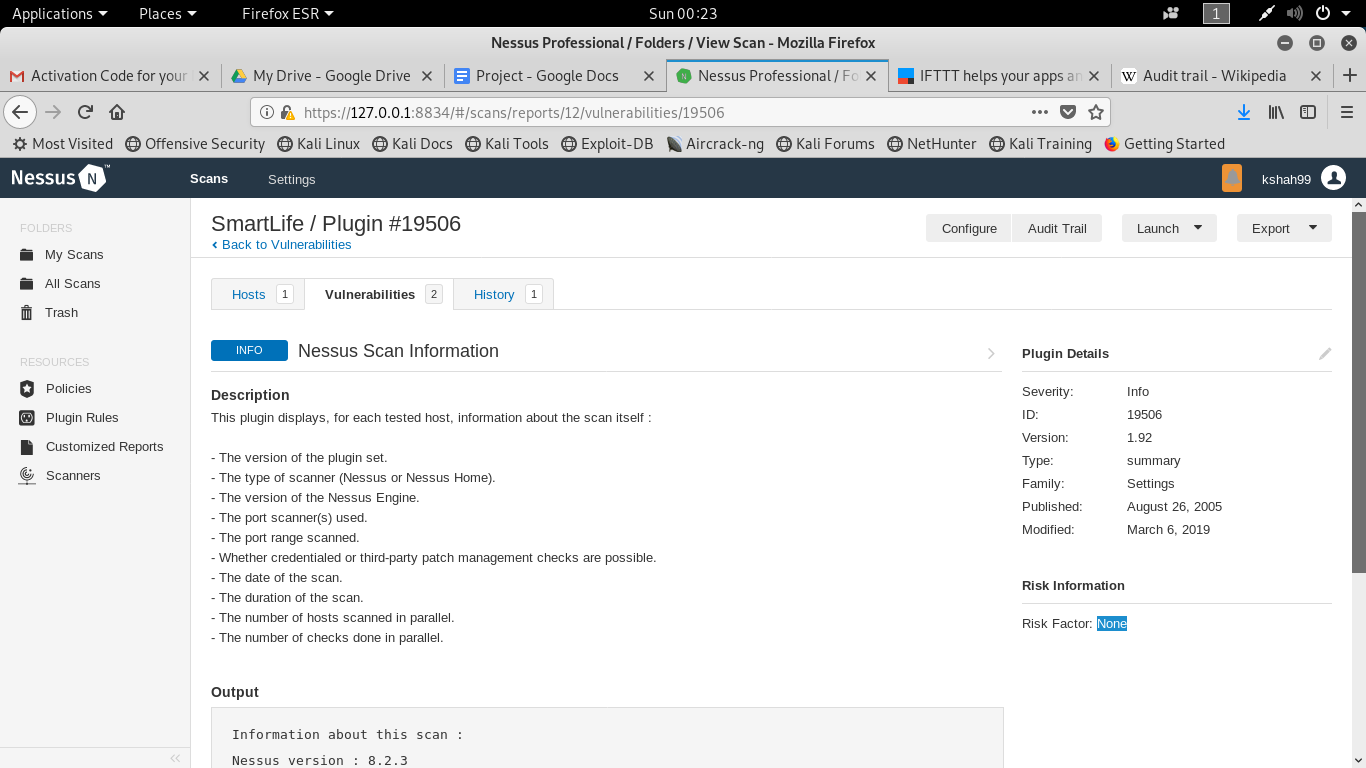


Figure: Information for SmartLife Web Application.

As mentioned earlier, these scanners also include information about the different risk levels and gives details about the risk. These risk steps include how they can be exploited and the proper remediation steps that can be taken with related references. This can be seen in the figure below:

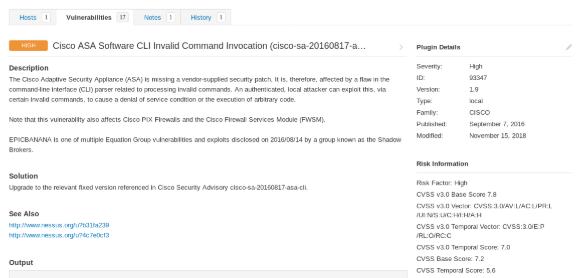


Figure: High level risk in Geeni web application.

From the results, an above high risk implies the Cisco Adaptive Security Appliance (ASA) is missing a vendor-supplied security patch. Therefore it can be attacked via certain invalid commands, to cause DOS attack.

Next, for OstorLab application we needed to perform the scan on the website itself. For the scan you need an .apk file of the mobile application. This can be easily available on the internet. Once you upload .apk file, OstorLab will give you a detailed report about the application as seen below

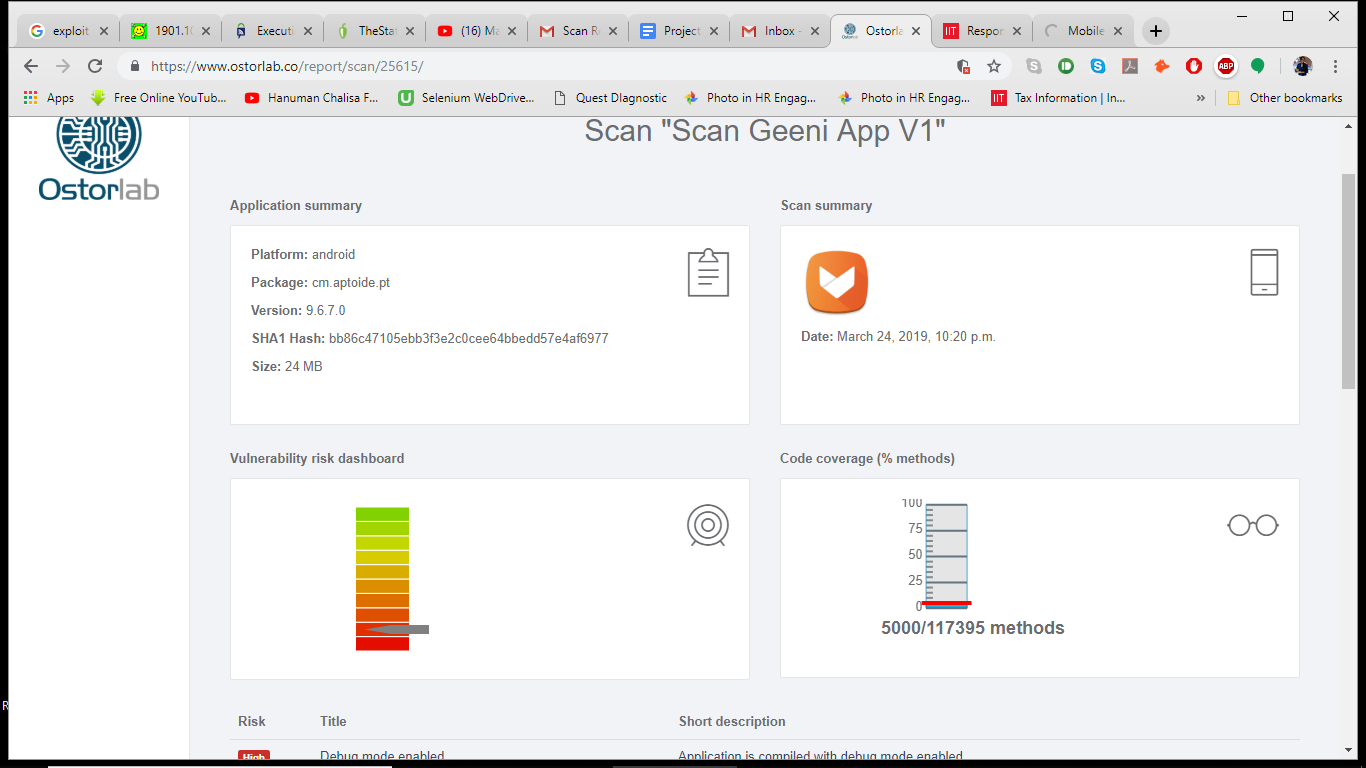


Figure: Detailed report about the Geeni mobile application

This report shows the risk of application, version, package and SHA1 hash. SHA1 hashing mechanism is insecure now-a-days as there have been proven collision attacks on it.

Same as Nessus, OstorLab also gives the details about the different levels of the risk which can be shown below:

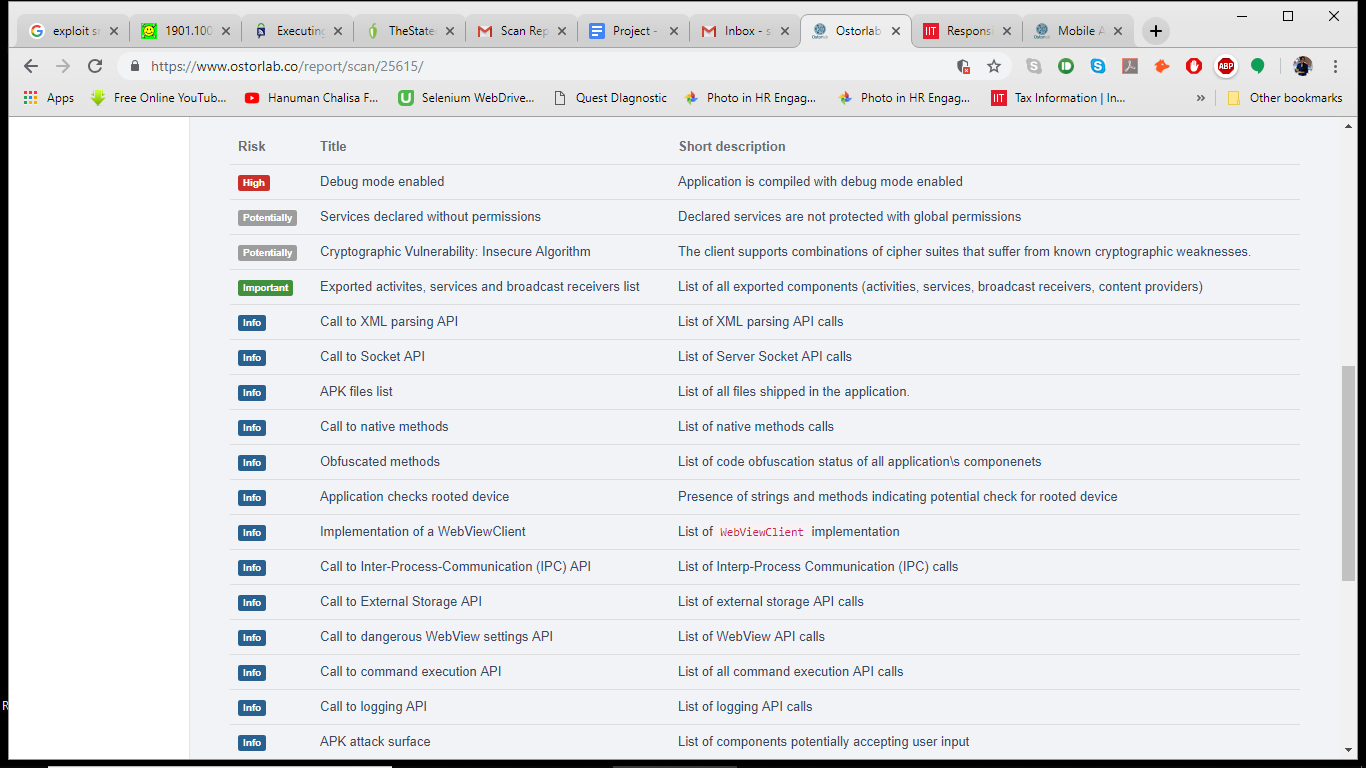


Figure: List of Vulnerabilities in Geeni Mobile Application

Now that we know the vulnerabilities and types of attacks, we can execute or perform them on the smart plugs. One of the medium level vulnerability we would like to discuss is the Insecure Network Configuration Settings in Smart Life mobile application. This is mostly due to the default network settings in the application. An attacker can perform Fake Wifi Access Points, Evil Twins and Man in the Middle Attack. A real life example for this type of attack is Mirai Botnet.

The next thing we work on was performing an actual attack on the web application. One of the vulnerability we saw was X-frame options settings malformed which can be exploited by clickjacking attack. This Clickjacking attack can be tested by by a simple test of trying to load the entire web application in a dummy HTML page iframe section.

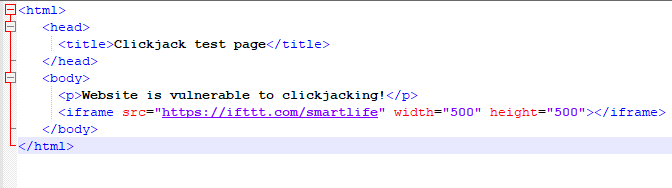


Figure: Code for ClickJacking Attack

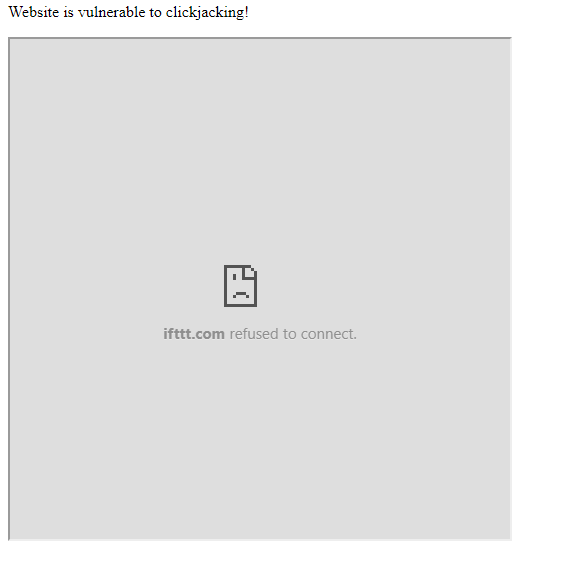


Figure: Result of ClickJacking Attack

The page fails to load, which proves that executing ClickJacking on the web application is not possible. If there are any complex approaches to performing the attack even though the test has failed, we are not aware of it yet.

Analysis of the off brand smart plug was performed mainly using nmap port scanner and wireshark to capture packets. The key findings were as follows:

1. TCP Port 6668 was found open for IRC Service on the device
2. UDP Filtered Port 49154 was found open for an Unknown Service

Nmap returns the MAC address for devices as well. The information associated with the MAC address revealed chipset used in the device, without having to touch the device let alone disassemble it. With minimal efforts, we found the entire guide of the device provided by the chipset manufacturer.[8] As a rudimentary step, we tried to telnet into the device through the open ports. Telnet to port 6668 was successful. Access to the device in character by character mode was given for a small window of time. Moving forward, we are going look at how this can be exploited.

To capture packets using Wireshark, the entire process of connecting to the device using the mobile app was performed. Broadcasted UDP packets were captured while the app was connecting to the device. The device also periodically broadcasts UDP packets. The packets broadcasted by the device contained the product key of the device in plain text. Hence, potentially sensitive information was communicated in an insecure manner. These packets were IPv4 packets. UDP as a protocol and IPv4 are known to be vulnerable. We were unable to capture any direct packets between the device and he app, which leads us to believe that the communication is carried out over cloud.

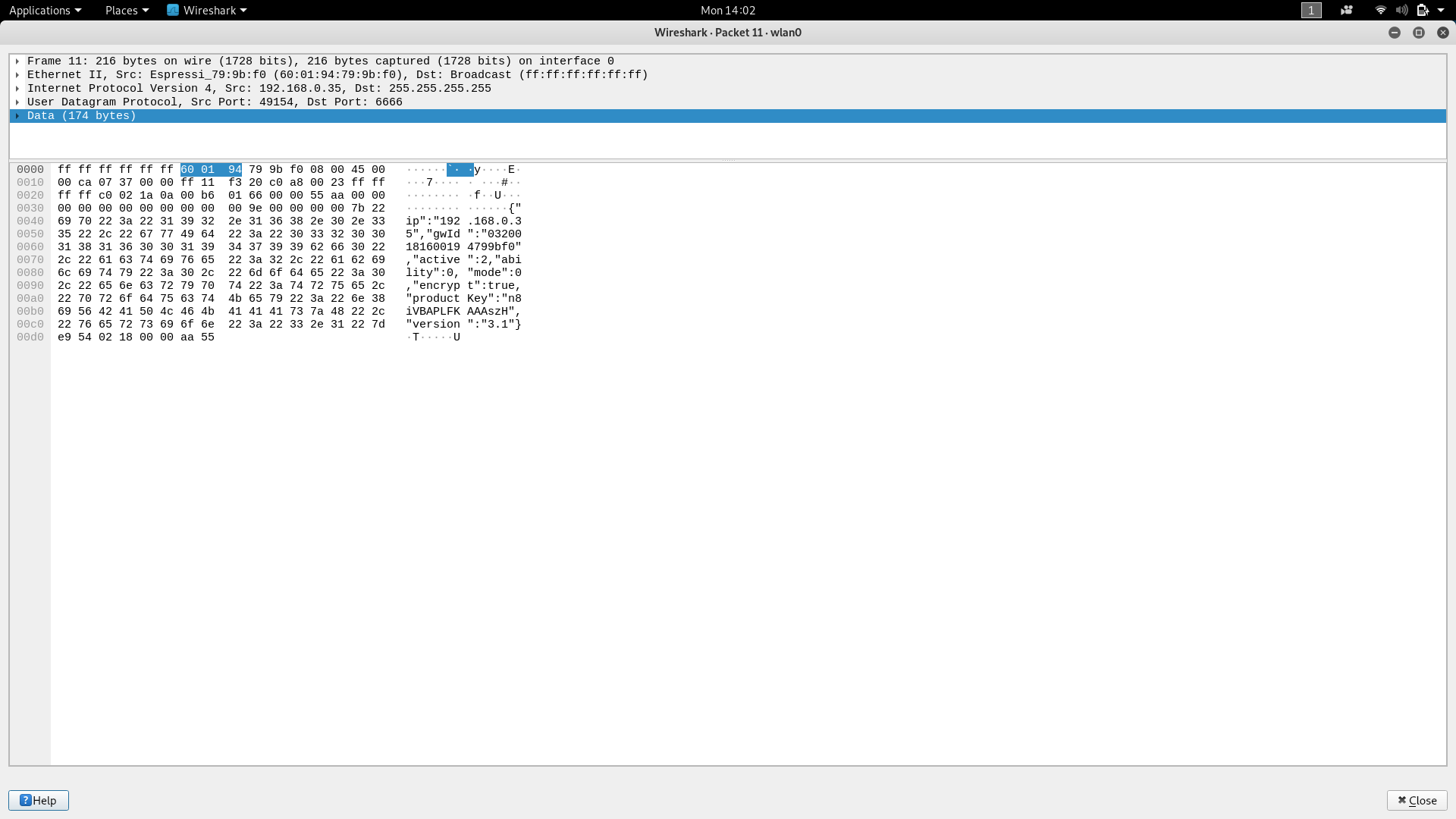


Figure: Wireshark packet capture of broadcasted UDP packet revealing product key of device

The device can be set in AP configuration mode as well. In this mode, the device turns into a hotspot and the mobile app connects to the hotspot of the device instead of connecting to the common WiFi network. The device hotspot uses open WiFi, which is highly insecure. Through our research, we came across possible attacks on the device set in AP mode. [9]

Port scanning resulted in the following UDP ports being open:

1. 67 DHCPS
2. 776 WPAGES
3. 49153 Unknown
4. 49226 Unknown

Apart from the vulnerability scan of the mobile app, we also looked at other features within the app. On trying to brute force the login credentials of the app, we found that the app employs account lockout mechanism. Once logged in, to connect to the device the only requirement is to be in the same WiFi network and know the Wifi password. For reasons unknown, the app only has support for 2.4 Ghz WiFi band. An issue we realised was that even if the attacker is in the same network and knows the WiFi password, connecting to the device is not possible unless the device is manually set in pairing mode. To be able to do so, the attacker would need access to the physical device.

Lastly we evaluated Samsung smart TV, in which we used nmap and burpsuite[8]. While performing nmap, we can see there are many open ports available for smart TV. When individually intercepted, all ports were secure enough for us to be unable to get into the smart TV[7]. This is because samsung uses its own encoding and decoding system.

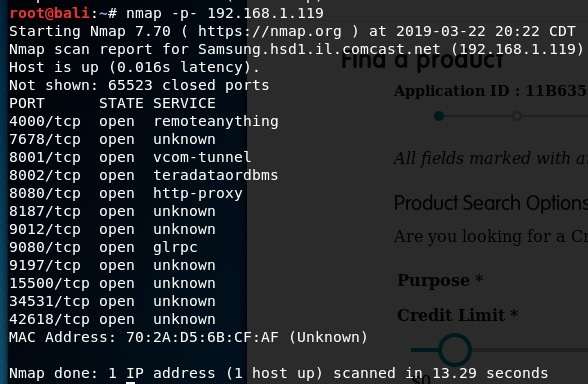


Figure: List of Open Ports in Smart TV

Post our broad spectrum analysis, we have concluded that it would be feasible for us to explore our options in attacking the smart TV given the time constraint.

1. Lessons Learned

Within our research, we soon learned that you have to either do one of two things. First, you have to either think creatively to understand vulnerabilities as the standard or assumed communication is secured in some form of sense. Second, you need to have in your possession a hardware device that assists with detection of IoT devices, such as a bluetooth dongle. Such items are not readily available at most electronic stores, so the average individual would have to make an additional effort in purchasing the item from the store online.

We learned that we need to improve our understanding of bluetooth communication. One of our potential end goals is to be able to take the communication of the smartband and mimic the call of the smartband being shaken to take a photo without the user of the band shaking the band themselves. Also, for smart plug from the information available we can infer that using IPV6 can increase the security for devices. We also came to the conclusion that the device itself does not have any kind of authentication mechanism, which could be a potential threat. Finally, we learned that it is highly insecure for the device to use open WiFi network in the AP mode.

Conclusion

There are always two sides of a coin; applying IoT technology to smart devices yields both opportunities and security risks.In our reconnaissance phase, we were able to get the raw data through Nessus, Ostor Lab, Wireshark, Burpsuite and nmap. Now we need to process the raw data to get the leaked information which then we will use for attack phase. Therefore, appropriate measures have to be taken by vendors to make smart devices more secure and suitable to live with. Our future work will be to attempt attacking the devices and documenting how easy or difficult it is to carry out an attack and whether or not it was successful.

References

[1] Dark 3, Inc (2018) The State of IoT Security, Retrieved from http://pepper.me/static/media/TheStateofIoTSecurity.171becc4.pdf

[2] The SSL Store, (Nov. 29, 2018) Executing a Man-in-the-Middle Attack in just 15 Minutes Retrieve from: https://www.thesslstore.com/blog/man-in-the-middle-attack-2/

[3] Beware of the App! On the Vulnerability Surface of Smart Devices through their Companion Apps, D.M. Junior, L. Melo, H. Lu, M. d’Amorim, A. Prakash, 2018, Retrieved from: https://arxiv.org/pdf/1901.10062.pdf

[4] https://www.ostorlab.co/report/scan/25615/

[5] Embedded Hardware Hacking 101 – The Belkin WeMo Link, G. Johnson, Aug. 22, 2016, Retrieved from: https://www.fireeye.com/blog/threat-research/2016/08/embedded\_hardwareha.html

https://github.com/tudorthe1ntruder/evil-plug-weaponizer

[6] Wireshark Dissector for Samsung remote protocol, Wiki,, https://wiki.samygo.tv/index.php?title=Wireshark\_Dissector\_for\_Samsung\_remote\_protocol

[7] How to easily hack your Smart TV : Samsung and LG, July 7, 2015, Retrieved from: https://iicybersecurity.wordpress.com/2015/07/07/how-to-easily-hack-your-smart-tv-samsung-and-lg/

[8]Smart Plug User Guide, Espressif IOT Team, 2016, Retrieved from: https://www.espressif.com/sites/default/files/40c-esp\_smart\_plug\_user\_guide\_en\_v1.2\_20160413\_0.pdf

[9]Ling, Zhen & Luo, Junzhou & Xu, Yiling & Gao, Chao & Wu, Kui & Fu, Xinwen. (2017). Security Vulnerabilities of Internet of Things: A Case Study of the Smart Plug System. IEEE Internet of Things Journal. PP. 1-1. 10.1109/JIOT.2017.2707465.

[10]h[ttps://wiki.samygo.tv/index.php?title=Wireshark\_Dissector\_for\_Samsung\_remote\_protocol](https://wiki.samygo.tv/index.php?title=Wireshark_Dissector_for_Samsung_remote_protocol)

[11] <https://itp.beverlychou.com/packet-sniffing-a-google-home/>