Pattern-of-life Analysis and Deviation Detection for Bluetooth and Bluetooth Low Energy Devices

MIDN 1/C Austin Andres
MIDN 1/C Edwin Zerwekh
Advisor: Dr. Dane Brown
United States Naval Academy
AY' 2022

Introduction:

Bluetooth and Bluetooth Low Energy (BLE) devices have become an incredibly common technology in our day to day lives. In recent years, the technology has extended its reach to fitness devices, audio pairing with mobile phones, and even controlling the many appliances in our homes. The connection medium between our devices poses a large security risk because of the open nature of the domain in which the signals are sent. A posed scenario includes a replay attack whereby an attacker, within range of your connected device, can resend a command to your device that has been intercepted. The attacker would reproduce this intercepted signal and resend it to the device you are connected to. This may not seem so detrimental until the device is a lock on your home's front door, or keyboard strokes to one's computer. The ability to recognize these sorts of invasive anomalies is a concern that has not been addressed by any available products. What is currently available on the market are Bluetooth/BLE 'sniffers' that can currently monitor traffic on the traditional frequencies. The sophisticated end of products that perform this function can cost anywhere from ten to seventy-five thousand dollars, depending on their capability, and with that price tag comes the need for a technical expert in order to operate the device. These sorts of devices are capable of monitoring bluetooth traffic but cannot categorize the normal patterns of life (POL) of a device much less generate alerts when a potential anolomoy has been witnessed by the device. It may sound like these devices do not amount to much, but in reality, they simply were not designed for the previously listed capabilities. The anomalies we're looking to identify are any behavior outside the devices' categorized POL, to include any malicious interactions by an aggressor.
For the sake of remaining succinct, this paper will assume that the reader understands the general fundamentals of Bluetooth and BLE communications so that we may explain in detail what our project has accomplished. This capstone has successfully utilized low-cost, off-the-shelf products to produce the capability of monitoring a device's normal bluetooth communications, creating a POL for the device, and generating alerts when an anomaly has been detected between the two communicating devices. The following work details how we developed this capability, to include the hardware/software utilized, the problems we faced while staying within our objective's limits, and follow-on work that would extend this project's capabilities.

Hardware Operations:

It will be best to explain what our culmination of software and hardware does, before we explain how we achieved that state, the issues we ran into that limited our progress, and where we would look to further our developments. Starting with the exterior hardware, we utilize three Ubertooth One devices operating on the 2.4 GHz frequency to intercept Bluetooth and BLE communications in the area. Each Ubertooth is operating on Firmware Version 2020-12-R1 (API: 1.07). We chose the Ubertooth One because it cost $150 each, but more importantly, came with user-friendly, open-source software that

consolidated the intercepted traffic in packet capture (PCAP) files. This open-source software made our initiative easier because we would have a streamlined method of analyzing the intercepted packets with Wireshark, also a user-friendly, open source program. The three Ubertooths were then connected, via USB, to our computer running on Ubuntu 20.04.3 LTS. This will be the machine that hosts the code and databases the entire entity operates on. We used a simple database called MySQL to house the pattern of life for an ELK Smart LED Light.

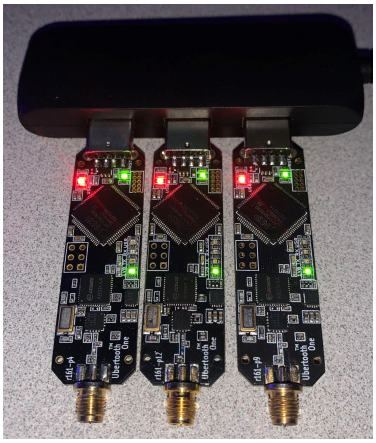
ELK Smart LED Light

We chose this device to examine due to the simplistic nature when proving this concept with low-cost products. In order to build the LED's POL, we monitored the device's traffic with our paired phone under every issuable command. Once we had sufficiently issued every command the LED could receive from the control device, the thousands of collected packets were used to identify specific attributes within the communications. This allowed us to achieve a database of known, trusted, behavior between the LED and the control device. Any device monitored would have to go through a similar analysis phase if the consumer sought to protect itself against a variety of attacks. The POL is essential to understanding when anomalies in communication have occurred because it provides the baseline for expected communication and allows for a baseline whereby all suspicious traffic may be compared to.

Software Operations:

After the hardware has been established and the device's database of intended traffic has been initialized, we can delve into our developed code that monitors and compares real-time traffic. It is best to explain this process starting with intercepted signals in the physical domain then describing the process we use to analyze the traffic. Starting with script_ELK.py, this program serves as the intermediate between our Ubertooths and the subsequent programs used to amend the POL, or detect anomalous traffic. The programs start by restarting all three Ubertooths, generating a random output file name for the three PCAP files, one text, and one CSV file. The purpose of the output file name generation is so that, if an Ubertooth is restarted, we do not overwrite previous files. The Ubertooths then start collecting local area traffic and generating PCAP files. Each Ubertooth's software has been altered such that it is either listening on channel 37, 38, or 39. The explanation for these specific channels and their significance will be explained later on. The generated PCAP files are then converted into a text file using a SEDs command. Essentially, all three Ubertooths will generate a PCAP file, but will all write

to a common text file that serves as input to our Main_ELK.py program. The Main_ELK.py program does the majority of the work in our process either in real time as the text file is being written, or retroactively with a finalized text file. This program takes the text file generated by the script_ELK.py program as input, which contains all the traffic collected by the three Ubertooths, then breaks the traffic down into its components for analysis.



Ubertooth Array

The specifics regarding how the components of the traffic are broken down are detailed in our GitHub repository for the readers' review. After the collected packets are parsed into its components there are two courses of action. The first course of action is writing to the MySQL database in order to update the device's POL. Before we can monitor a device, we need to observe how the device operates as it's intended use. This would be an area where we would implement machine learning, but for now our program manually inputs the POL to establish a baseline. The second course of action is a monitor mode where the program compares the collected traffic with the trusted traffic within the database. Before delving into monitor mode's capabilities, it is important to explain first the two states a BLE device can be operating in. The first state is called advertising mode where the device is constantly sending out packets in search of a control device to pair to. The analysis for this mode is quite simple as the device sends packets out at regular intervals and volume such that it can be recognized thereafter. We are able to look specifically for this magnitude/frequency of packets on a device-specific basis. Our program is tailored for the frequency and magnitude of the LED device's broadcast characteristics. There comes a potential for malicious activity when the broadcast characteristics begin to deviate from what is expected, which is where our program looks to identify a device posing as our LED light. This identified behavior brings us to our first anomaly detected being a man in the middle attack. This characterization of activity is logged and our program generates an alert that an anolomoly has been detected, which potentially prevents a host device from connecting to a device it thinks it knows/trusts. Another potential indicator of a man in the middle attack is large transients in the device's RSSI measurement. RSSI can be thought of as the signal strength of the BLE device. When a broadcasting/ connected device moves in relation to the host device it is connected to, the RSSI level naturally changes with the varying signal strength due to many factors. These normal transients are not worrisome, however when a large change in the RSSI level occurs within a near-instantaneous unit of time it can be an indicator that another device has

begun broadcasting or communicating with the host device. These characteristics are indicative of another attempt at a man in the middle attack and generates a logged report of the incident followed by an alert generated by the program. This sort of attack is not device specific as it is more a factor of the connection medium and the implied circumstances rather than some intrinsic quality of the device. That being said, this anomaly detection transfers well to any other device. The rest of the anomaly detection occurs in what was briefly mentioned earlier, connected mode. During this mode of operation, the host device is communicating back and forth with the paired device, sending all the commands needed for normal device operations. In this mode of operation, our ubertooths are capable of detecting replay attacks or any other suspicious activity that we have yet to specifically codify in our program. Any anomaly we have not specifically identified will trigger an alert and get logged such that an analyst can review the traffic at a later time to determine if the activity was abnormal but valid or abnormal and invalid. This is an important feature, from a security standpoint, because it is not feasible, nor prudent, to think we can anticipate every attack vector. For testing purposes, we acquired the NRF Connect (App) which is capable of picking up bluetooth communications in the area and sending signals to the nearby devices.

NRF Connect (App)

We configured the NRF to reproduce a command that changes the color of the LED bulb after seeing the same command be passed legitimately from our host device. Our program can distinguish the opt codes and a few other deviations in the command such that we can discern whether the sending device is our host device or the NRF. Once a replay attack or any other abnormal command has been detected, our program generates an alert and logs the questionable packet for later review.

Anomaly Notification

This program is concluded by creating a CSV file which contains all the logged events that the program deemed as abnormal based on the comparison with our database. It is important to note that most of our capability is tailored specifically for the LED light, but that the specific parameters can be altered for any other device.

Challenges/Lessons Learned:

Reaching the capabilities outlined previously took months of research and extensive review of previous works to include reaching out to the creators of the Ubertooth. The following section will outline the trials we faced when working with low-cost equipment and how we either overcame certain challenges or limited the scope of our work in order to prove the concept. We began working with the Ubertooth devices in late September, 2021. Initially we spent our time understanding how to interface with the Ubertooth and began working on how to develop code in tandem with the Ubertooth's capabilities. We faced early successes by witnessing the devices in advertising mode and characterizing the normal POL in this state of operation. This early success was followed closely by our largest hurdle which was seeing bluetooth communications after a connection had been made with a host device. When we exhausted initial troubleshooting methods, we began diving into research mode in hopes that we could better understand Bluetooth/BLE communications and tap into the paired devices' communications. By late October, we were in a state of panic seeing as no previous research was proving useful to our pursuits. At this point, we were offered a commercial product by our subject matter experts (SMEs) at the National Security Agency, but we turned down the offer to stay true to our project's goal of accomplishing our goals solely using low-cost off the shelf products. The device so graciously offered by the NSA is a seventy-five thousand dollar piece of equipment, so we certainly did not want to utilize it, even if it was for the purposes of visualizing the data for educational purposes. While this device did not have the capabilities of creating a pattern of life or logging/generating alerts, it took a high degree of technical knowledge to use it. Thus, we had two strong reasons not to use the device: cost and difficulty of usage. It is noteworthy that at this time we were able to, for some unknown reason, pick up the connected communications between a Fitbit and an iPhone, but we were unable to reproduce this occurrence after the fact. This phenomenon will be better understood just a few months later. Early December we reached out to the Ubertooth creators on their github where we found an unanswered question trying to address our same problem. Within this question's comments brought to our attention the fact that the Ubertooth will randomly select from three data channels when a connection has been made, so if your devices are not communicating on those channels we would not be able to see the connected traffic. Our NSA contacts provided us with the opportunity to speak with Mr. William Pitchford whose team had been tackling a similar issue with the Ubertooth. We were beginning to realize during our tests that about one-third the time we were able to see the connected communications but that it was inconsistent due to the randomization of the data channel. One proposed solution was to restart the Ubertooth anytime a connection was dropped, but Mr. Pitchford's team attempted this solution with little success. Early January was when we began

to experiment with another low-cost device, the nRF52840.

nRF52840 Dongle

The nRF52840 fulfilled a similar role to the Ubertooth One, but it did not come with its own open-source software and it was only twenty dollars. The nRF was pivotal in our understanding of data channels because the nRF required the user to set the data channel which when done correctly allowed us to see the data transfer one-third of the time. We ended up not using the nRF for our final product because it did not fulfill the backend of packet capture by creating a PCAP file of the collected data like the Ubertooth. Nonetheless, the nRF was instrumental in our understanding of the data channels used for communication. Our solution to the data channel issue was to acquire two additional Ubertooths and set their data channels on startup such that one of them will be able to capture the data packets regardless of the channel the BLE device had picked. This may seem like a brute force solution that increases the cost of the project, but even at roughly five-hundred dollars, we are not even close to the commercial products in the tens of thousands. We completed the acquisition of the Ubertooths in late April and spent the next few months creating the programs explained before in order to get the Ubertooths working together and write to one output file, ensure the Ubertooths did not lose connection mid-capture, create a pattern of life for our LED light, determine the attributes that were indicative of specific and non-specific anomalies, and finally generate alerts of anomalies detected and create the log file containing each alert.

Future Work:

When we initially set out to prove this concept, we had far more ambitious goals and we heavily underestimated the challenges we would face with these low-cost devices. The rest of this work will list the capabilities we would further pursue with this proven concept, should we have more time. The first of many capabilities would be to monitor multiple devices. Our current state of operations is tailored to the LED light, but ideally, we would have multiple databases with the program wielding the capability to discern which device it is monitoring and correctly compare the associated database with the appropriate device. The next pursuit would be working towards an autonomous database formation whereby whenever a device was targeted or identified by our product, the program would be able to passively map its pattern of life without any sort of user input. This would require our product to always be running, at least in a passive collection mode, but would allow a user to immediately monitor its device without having to first establish a database. An additional feature we would hope to develop for our product is the ability to name certain attacks and not be limited to arbitrary alerts. This feature would allow us to issue a priority statement to the users' devices such that they could make an informed decision

concerning their next course of action. This action could be to turn the device off, or to simply be cognizant of certain malicious activity in the area. Lastly, we would look to incorporate a form of machine learning to create more complex POLs. The aim here is to take a complicated bluetooth device, such as a fitness monitor, and be able to detect not just technically wrong behavior, but anomalous, legitimate activity for that user. The full capability of this system has yet to be realized and we are certain that its full potential would offer a vitally needed capability for ensuring the security of Bluetooth devices.