Bluetooth Vulnerability Research

Kelly-Ann Downer   
Cybersecurity Engineering  
George Mason University  
Fairfax, VA  
[kdowner@gmu.edu](mailto:kdowner@gmu.edu)

Kaitlyn Malone  
Cybersecurity Engineering  
George Mason University  
Fairfax, VA  
kmalone8@gmu.edu

Josh Minick  
Cybersecurity Engineering  
George Mason University  
Fairfax, VA  
jminick@gmu.edu

Vi Nguyen  
Cybersecurity Engineering  
George Mason University  
Fairfax, VA  
[vnguye67@gmu.edu](mailto:vnguye67@gmu.edu)

Jack Raymond  
Cybersecurity Engineering  
George Mason University  
Fairfax, VA  
[jraymon@gmu.edu](mailto:jraymon@gmu.edu)

*Abstract*—As Bluetooth technologies become more popular, the vulnerabilities within Bluetooth also increase significantly. These vulnerabilities make the systems and devices more exploitable and put confidential data at risk. There are two types of authentications when it comes to pairing, which are Legacy Secure Connections (LSC) and Secure Connections (SC). These processes allow the two Bluetooth devices to securely pair with one another and establish a secure connection between the two devices for future uses. However, an attacker can insert themselves into the system, pretending to be one of the pairing devices and manipulate the system connection once they get in. The objective of the senior design project is to exploit known vulnerabilities, create automated vulnerability remediation, and find mitigation plan to apply on future vulnerabilities.

Keywords—Bluetooth, Bluetooth Vulnerabilities, Exploits, Automation, Detection Systems, Remediation Systems, Python, Programming, Linux kernel, Remote Code Execution, Eavesdropping Attack.

# Overview

The sponsor that our team is working with is Lockheed Martin (LM), which is one of the biggest aerospace and security companies in the world and was created in 1995. Lockheed Martin works with private companies and organizations on defense and security projects and has contracts with the government to build aircraft and spacecraft for the military. Lockheed Martin is an international company that employs thousands of people who are working to ensure the company is operating properly through their use of extensive enterprise networks and information technology solutions. Lockheed Martin has tasked our group with researching known Bluetooth vulnerabilities and exploits and investigate ways to mitigate these vulnerabilities. From our research we found two prevalent and modern exploits: BleedingTooth and Crackle. BleedingTooth is a newer attack in which the attacker can also gain root user privileges, change any settings, and get any files on the user's system. This becomes more crucial based on the type of information being held on the device, whether it be a personal device or a CEO’s computer. Crackle is a different type of attack that uses packet sniffing of vulnerable devices. Some devices that are vulnerable are medical, meaning the lives of humans are at risk. If compromised an attacker can change the packets and display incorrect readings on the monitor causing a delay in medical assistance or causing a false alarm. These two exploits show the range of varied Bluetooth attacks and the potential for new vulnerabilities to be found. By researching and performing these two Bluetooth attacks, our team have learned where the system was vulnerable and created tools to mitigate the vulnerabilities that allowed the attacks from happening.

# Concept of Operations

## Phase I

This project is split into three primary phases: 1) Vulnerability Exploitation, 2) Automated Vulnerability Detection, and 3) Vulnerability Mitigation and Testing. In the first phase, vulnerability exploitation, the team was asked to exploit two Bluetooth vulnerabilities. First, two Common Vulnerabilities and Exposures (CVEs) were found and exploited successfully. Phase one included weeks of research about Bluetooth and potential vulnerabilities within the implementation of Bluetooth to Linux systems. Our team investigated devices that were both commonly owned as well as exploitable without too many external devices to help keep down operations costs. The two exploits we ended up choosing were the BleedingTooth vulnerability and the Crackle vulnerability. Both exploits can be dangerous if used by bad actors. Next, we reproduced the results and documented our findings. By reproducing the results and recording the exploitations, we can review our footage to help us easily identify mitigation strategies in the next phase. We also documented our findings in a report to both provide these findings to our Sponsor, but also to allow for further research and work to be done on this project in the future.

## Phase II

In the second phase, automated vulnerability detection, the team is tasked with creating and documenting an automated tool. The automated tool is expected to read a device and determine any vulnerabilities it might have, as well as provide mitigations or patches to help secure the device. To do this, our group looked at unique identifiers for both exploits. The automated tool will scan the system and check for these unique identifiers and then alert the user of the findings. For our first exploit, BleedingTooth, we decided to scan the system for any of the known vulnerable kernels. By scanning the system, we show the user that their version is vulnerable to the BleedingTooth exploit and we can provide our recommendations based on the findings from the scan. For our second exploit, Crackle, we use the same idea that Crackle is based on, but we turn it into a security tool. Crackle scans packets captured from a Bluetooth sniffer and determines if the program can find a Long-Term Key from the provided packet information. We used that logic and combined it with our own code to allow the user to run our program and determine if their device is vulnerable to the Crackle exploit. If the user has a device that uses Bluetooth Low Energy as well as Legacy Pairing, we inform the user that their device is susceptible and can provide mitigations from there.

## Phase III

Lastly, in the third and final phase, vulnerability mitigation and testing, the team will patch the previously exploited vulnerabilities and create a mitigation plan that will ensure more efficient remediation in case of a repeat event. The findings from the previous phase will aid the team in finding appropriate mitigations to help secure the system. The team will then have the automatic tool to ask the user whether they wish to apply any available patches, such as running a system update, or if they would like to apply the patches and remediations later.

# Design

## Phase I

### BleedingTooth: The first exploit is a Bluetooth-based, Zero-Click Remote Code Execution affecting Linux systems with a kernel version between 4.8 and 5.9. This exploit is a proof of concept originally developed by Google security researcher Andy Nguyen to attack two BleedingTooth vulnerabilities in the Linux Bluetooth subsystem. Andy Nguyen is also responsible for identifying and reporting all three existing BleedingTooth Vulnerabilities: BadVibes, BadChoice, and BadKarma. For more information about the exploit and BleedingTooth vulnerabilities, please read Nguyen's writeup [1].

### BleedingTooth consists of three vulnerabilities. The two vulnerabilities used in the attack are BadChoice and BadKarma. BadChoice is a stack-based information leak resulting from improper error handling of the A2MP\_GETINFO\_REQ command within the Linux kernel. This command is intended to request information about the AMP controller of a device using the HCI device id. However, if the device id is invalid or of the incorrect type, the device will respond with an error status. This sent error status is of type a2mp\_info\_rsp, which has many variables, but the error handling of the a2mp\_getinfo\_req() subroutine only sets the id and the status of the response. Therefore, the given response has several uninitiallized variables which can be manipulated into disclosing up to 16 bytes of the kernel stack. By leaking the memory layout, an attacker gains the address of a target memory block which they can attempt to control. BadKarma is a heap-based type confusion in the amp\_mgr\_create() function within the Linux kernel. The vulnerability occurs when the A2MP channel is configured as ERTM/streaming mode and an object of type “struct amp\_mgr” is passed to “sk\_filter()” where an object of type “struct sock” was expected. As a result, “struct sock” variables are falsely set to values of the “struct amp\_mgr” object, which could result in accessing values from invalid addresses causing the kernel to panic. Using this vulnerability, an attacker can control the memory address of the sk\_filter pointer.

### Combining these vulnerabilites, an attacker could leak the memory layout, store a payload within a known address, and then have sk\_filter point to the payload causing a remote code execution attack. In the case of the BleedingTooth proof of concept, the payload triggers a reverse shell which allows a remote attacker to execute commands on the victim’s machine as the root user.

The BleedingTooth attack requires that the victim device is using a Linux kernel version between 4.8 and 5.9, has Bluetooth on and has their Bluetooth controller set to discoverable. To protect the safety of our personal devices we opted to implement our target environment on a virtual machine. Our chosen target operating system is the same as the one used in the original proof of concept by Andy Nguyen: Ubuntu 20.04.1 LTS running kernel version 5.4.0-48-generic.

The attacking device can be any device with Bluetooth connectivity that can compile and run programs using the C programming language. We initially chose a raspberry pi due to the cheapness and the portability of the device, but eventually realized it was not capable of performing the attack fast enough before the Bluetooth connection closed. Therefore, we opted to use a secondary laptop for increased performance.

### Crackle: Our second exploit is Crackle, a Bluetooth Eavesdropping Attack affecting Bluetooth Low Energy (BLE) Devices using Link Layer encryption and Legacy Pairing. Unlike the BleedingTooth exploit, Crackle does not attack a particular implementation of a Bluetooth stack, but instead exploits a flaw in the Bluetooth specification itself. The Crackle exploit tool was developed by Mike Ryan. For more information about Crackle, please see Crackle's GitHub Repository [2].

### The Crackle tool exploits a flaw in the Bluetooth Low Energy pairing process that enables an attacker to guess or brute force a Temporary Key (TK) which can then be used to determine the Short and Long Term Keys (STK and LTK) from which all subsequent communications can be decrypted. To brute force the key, an attacker would need to use a Bluetooth sniffer to record and save the BLE pairing conversation between two devices. The attacker would then run Crackle with the captured packets as input. Crackle uses these packets to analyze the connections and determine the best strategy to crack the LTK. The speed with which Crackle cracks the key depends on the number of pairing packets that were captured.

### Crackle can only be used to eavsedrop on a BLE connection between devices using Link Layer encryption and Legacy Pairing. The authentication methods in Legacy Pairing are Just Works and a 6-digit PIN. The TK used in these authentication methods is a value in the range of 0 to 999,999 padded to 128 bits, making it easy for Crackle to brute force. If the TK is cracked, Crackle will unencrypt the initial packets that were encrypted using the TK and attempt to obtain the LTK. The LTK is usually immediately exhanged after first establishing initial encryption using the TK, so no additional brute forcing is needed. After obtaining the LTK, Crackle will output the key for the user and optionally decrypt the remaining packets that were encrypted with the LTK. If an attacker obtains the LTK, they may record future connections between the devices and later decrypt the packets using Crackle’s “Decrypt with LTK mode”.

### The attacking device can be any device with Bluetooth connectivity that can compile and run programs using the C programming language. For the victim devices, both devices need to support BLE and at least one device needs to use Link Layer encryption and Legacy Pairing.

## Phase II

### To create our automated detection tools, we chose to design two command line programs written in python.

### BleedingTooth: The first automated vulnerability detection tool is intended to run on a given Linux system and determine if the kernel version is between 4.8 and 5.9. If so, the given device is vulnerable to exploiataion by the BleedingTooth remote code execution attack and the user will be notified of the next steps for remediation. If not, the user is safe from exploitation and the user will be notified that no further actions are needed.

### Crackle: The second automated vulnerability detection tool is intended to to detect whether a given Bluetooth connection is vulnerable to having its temporary key cracked by detecting if Link Layer encryption and Legacy Pairing is used. It is possible to know whether a device uses Legacy pairing simply by analyzing its advertisement data when it is discoverbale. Unfortunately, the only means of determining if a device uses Link Layer encryption is by analyzing packets sent from the device during an active Bluetooth connection to another device. Therfore, our automated detection tool will analyze a given packet capture and inform the user whether each of the devices used in the connection are vulnerable to exploitation.

## Phase 3

### To create our automated remediation tools, we added vulnerability fixes and suggestions to our detection tools.

### BleedingTooth: The first automated remediation tool will detect whether a system is vulnerable by checking its kernel version. If it is vulnerable, it will notify the user that their system is unsafe and will ask them if they wish to update the device. If the user responds yes, their system will install a newer, stable kernel version (5.10 or above) and reboot to apply the changes.

### Crackle: The second automated remediation tool will detect whether either of two devices in a Bluetooth connection are vulnerable by analyzing a given capture file. As Crackle is a flaw in the Bluetooth specification, if a device is susceptable to exploitation, it can not be patched. Therefore, if either device is susceptible, the user will be notified and told to use the device with caution. If either of the susceptible devices are currently paired to the user’s device, the tool will ask the user whether they would like to remove them. Once removed, the user will be unable to use the susceptable device until they repair it in a secure location.

# Implementation

## Phase I

### BleedingTooth: To set up the exploit, we needed to create two devices, one for the victim, and the other for the attacker. Our victim device is a virtual machine running Ubuntu 20.04.1 LTS with the 5.4.0-48-generic Linux kernel. The following steps describe victim device’s setup process:

#### Using VirtualBox, create a virtual disk and boot the Ubuntu 20.04.1 LTS image

#### Follow Ubunutu’s installation instructions on the live environment

#### Enable USB 3.0 passthrough on VirtualBox to connect to a USB Bluetooth dongle to the virtual machine

#### Install the 5.4.0-48-generic kernel and it’s extra modules

#### Reboot the device into the vulnerable kernel

#### The attacking device can be any device with Bluetooth connectivity that can compile and run programs using the C programming language. We initially opted to use a Raspberry Pi, but found that we were unable to achieve a remote code execution. Instead, the attack timed out and caused a denial of service of the victim device’s Bluetooth capabilities. After troubleshooting the connection processes, analyzing the code for the exploit, and retesting many times, we finally discovered that the attacking device was the issue. The Raspberry Pi was not powerful enough to run the exploit, as the connection would drop before it could finish executing. We therefore needed to use a second laptop to perform the remote code execution. The following steps describe how to perform the BleedingTooth attack:

1. On the attacking device, use Netcat to listen for TCP/IP connections from a given port
2. In the same terminal, write the commands that will be executed on the victim’s device
3. In a new terminal, run the exploit using the victim’s Bluetooth address, the IP address of the attacking device, and the port used for Netcat in step 1
4. If successful, the victim’s device will run “/bin/bash -c /bin/bash</dev/tcp/Source\_ip/Source\_port” and the attacker can use the terminal with Netcat as a remote shell for the victim’s device

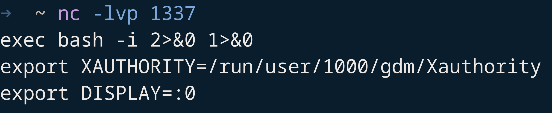


Fig. 1. Netcat listening for incoming connections from port 1337.

As seen in Fig. 1, our attacking machine uses Netcat to listen for an incoming connection from the victim. Once the connection is established, the first three lines are executed. The first command spawns a bash shell in interactive mode and redirects the standard error and standard output to standard input. The next two lines set environment variable that allow GUI applications to be run by the root user



Fig. 2. Running the exploit

In Fig. 2, we run the exploit in a separate terminal. This proof-of-concept code will connect with the victim’s device over Bluetooth, store a given payload in memory and then execute it, causing a remote code execution attack. The payload is the command “/bin/bash -c /bin/bash</dev/tcp/Source\_ip/Source\_port” which will execute a bash shell with root privileges on the victim's device and will take input from a remote connection. Thus, this command makes the victim’s machine establish a remote connection with the attacker on a specified sport. Since Netcat is already listening on our attacking device, the connection will be established. The first command that is run after the connection is established, “exec bash -i 2>&0 1>&0”, redirects the output of the shell to its input, which means the shell’s output is redirected to the remote connection. Without this, commands could be executed on the victim’s device, but the attacking device would be unable to see the results of a command output to their terminal.

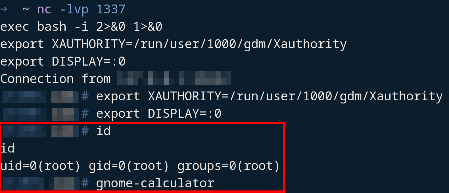


Fig. 3. Resulting remote shell

Finally, Fig. 3 shows that the attack was successful, as we have successfully created a reverse shell. Since the uid is zero, we are executing commands as the root user. As a demonstration, we open the calculator application on the victim's device.

2) *Crackle:* The Crackle exploit required a specific piece of hardware called a Bluetooth Sniffer and Wireshark. For this exploit our target device was once again a virtual machine running Ubuntu 20.04.1 LTS. To establish the victim machine, the following steps were taken:

#### Using VirtualBox, create a virtual disk and boot the Ubuntu 20.04.1 LTS image

#### Follow Ubunutu’s installation instructions on the live environment

#### Enable USB 3.0 passthrough on VirtualBox to connect to a USB Bluetooth dongle to the virtual machine

After ensuring Bluetooth was properly working on the victim device, the attacking device was configured:

1. Install Wireshark, Python, pySerial, and Crackle repository for packet analysis.
2. Ensure packet sniffer (AdaFruit Bluetooth Sniffer) is properly configured and connected to the attacking device.
3. Place sniffer in between victim machine and Bluetooth device during pairing process to capture communication packets using Wireshark.
4. Run pcap file through Crackle to analyze for necessary packets in the decryption process.

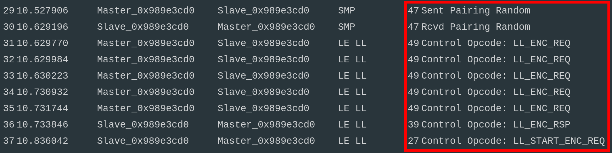


Fig. 4. .pcap file with critical packets captured

Given the required packets are captured, use Crackle to brute force the LTK and further decrypt communication packets.

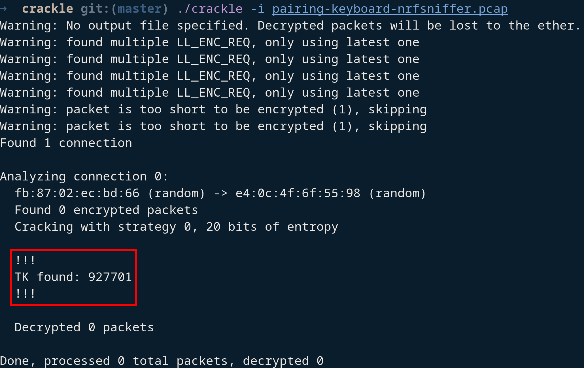


Fig. 5. Crackle successfully cracking the TK using brute force methods

In our experiment we opted to use packet captures from online sources to ensure the required encryption methods were used and captured. With these packets, Crackle was able to brute force the transfer key and decrypt future communications such as battery levels, device readings, and user actions.

*Phase II*

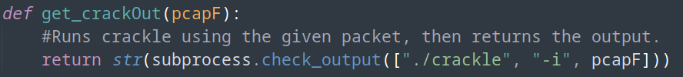
### BleedingTooth: Our automated vulnerability detection tool checks whether a given Linux system is using a kernel version between 4.8 and 5.9.

### 

Fig. 6. Kernel Vulnerability Detection

As seen in Fig. 6, the tool runs “uname -r” to get the device’s current kernel version. The isVulnerable() function is used to determine if a device’s kernel version is larger than 4.8, but smaller than 5.9. If it is, then the device is vulnerable, and it returns true. Otherwise, the device is safe, and the function returns false.

### Crackle: Our automated vulnerability detection tool determines whether a device is using Bluetooth Legacy Pairing and Bluetooth Low energy.



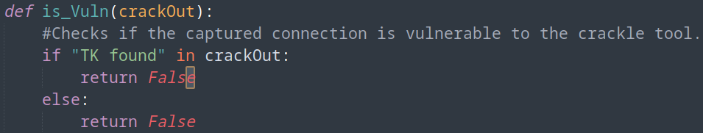


Fig. 7. Crackle TK search

Our Crackle automated tool runs Crackle using the command “./crackle -i .pcap” which then returns whether the TK is found to the command line. Second, the is\_Vuln() function searches the output from get\_crackOut() to see if Crackle found a TK. If a TK is found, then the device connection is considered vulnerable.

Once the tool has been run, it detects whether a device is vulnerable to the attack, whether the connection has been recorded, and some options for how to continue using the device. These options include accepting the risk and using the device as it is, disconnecting the device until it can be paired securely, and disconnecting the device and disabling Bluetooth services. All options have a risk factor associated as well as changing the ease of use of these devices.

*Phase III*

### BleedingTooth: If our tool detects that the system is susceptable, it will ask the user whether they would like to update to the latest stable kernel.

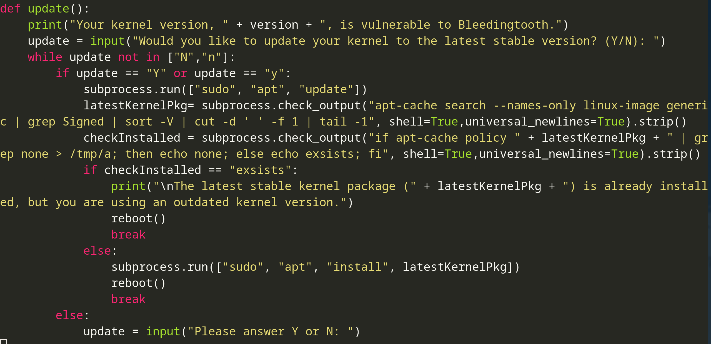


Fig. 8. BleedingTooth Update function

If the user replies yes, the tool will run “sudo apt update” to fetch the latest packages from the Ubuntu repositories. The tool then queries for the latest stable kernel package and checks whether the latest kernel is already installed. If it is already installed, the user will be informed that they already have the latest kernel, but that they need to reboot their device to use it. If the latest kernel is not installed, our tool will install it and will likewise ask the user if they want to reboot their device. If the user chooses to reboot their device, their machine will automatically boot using the latest kernel.

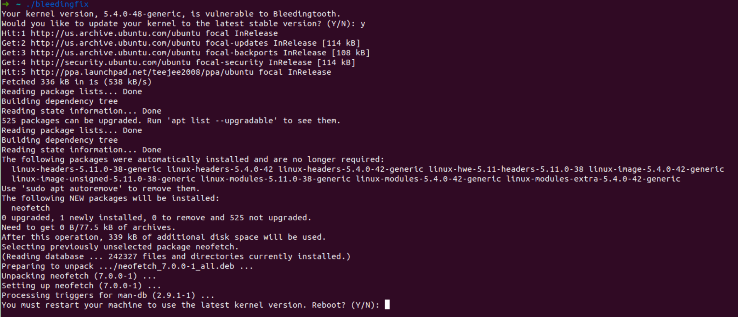


Fig. 9. Sample output of the BleedingTooth remediation tool

Fig. 9 shows an example execution of the BleedingTooth remediation tool. The kernel is updated and then the user is asked to reboot.

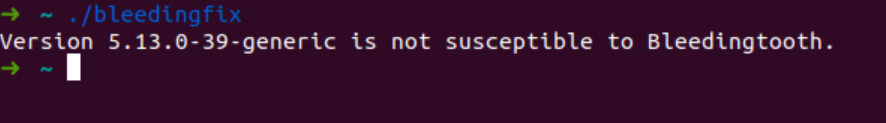


Fig. 10. Output of BleedingTooth Remediation tool after the update is applied

After rebooting, Fig. 10 shows that the device is now using kernel version 5.13.0-39-generic, so the device is no longer susceptible to BleedingTooth.

### 2) Crackle:

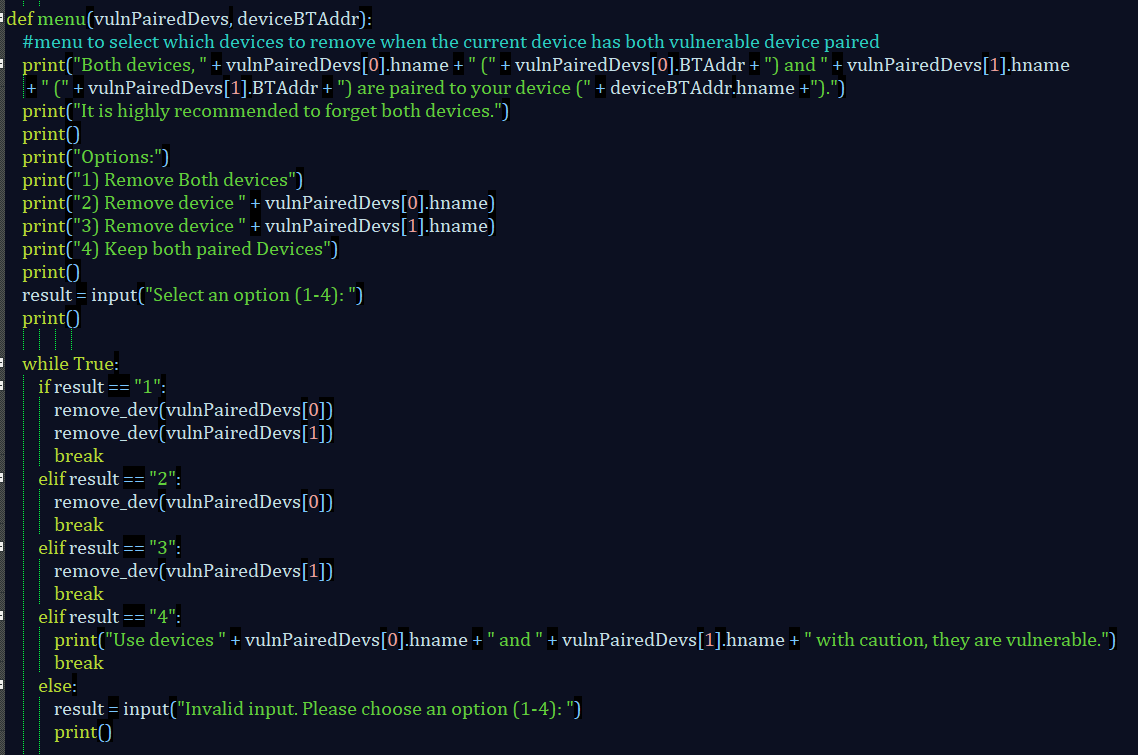


Fig 11. Bluetooth Pairing Vulnerability Detection

Fig. 11 above shows the main menu for our Crackle vulnerability detection tool. The code starts by stating which devices are paired to the user's device. The code then gives options labeled 1-4 which the user can choose to remove both connected devices, a single device specified by the Bluetooth address, or to keep both devices paired. Once the tool is run, the following Fig. 12 shows an example output.

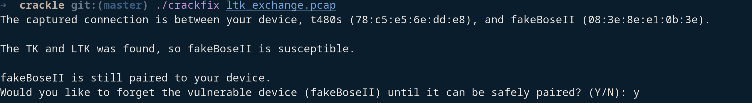


Fig 12. Crackle Automated Tool Command Line Output Case 1

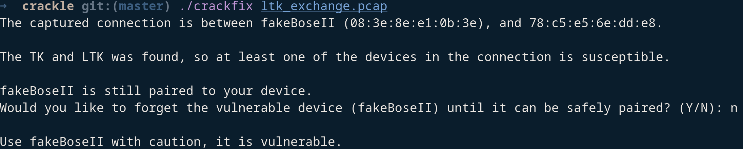


Fig 13. Crackle Automated Tool Command Line Output Case 2

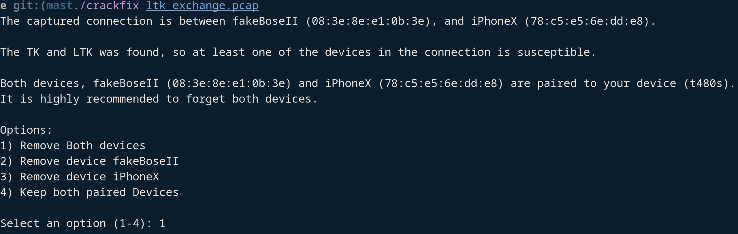


Fig 14. Crackle Automated Tool Command Line Output Case 3

As seen in Fig 10. and Fig 11. above, the program asks if the user would like to forget the vulnerable Bluetooth device and the user selects “yes” and “no” respectively to receive the following output. When the user selects to remove the vulnerable device, the program simply forgets the device until it can be paired again in private. This means that the Long-Term Key (LTK) would be different, and the connection would be secure. In Fig 12, there are two connected vulnerable devices so the program offers a set of four options as described above.

# Conclusion

This paper highlights the research of two vulnerabilities that affect the Linux operating system and introduces a detection tool to warn users. The team operated in three phases after the research was completed. In the first stage, the exploit was modeled, secondly, a tool was developed to detect the vulnerability and the last stage included testing of the tool. The first vulnerability researched is BleedingTooth. BleedingTooth has a family of three CVEs which together affect Linux Kernels 4.8 through 5.9. When exploited it spawns a remote shell on the attacker's machine where the attacker can exploit the device. The second vulnerability is Crackle, which affects the Bluetooth Low Energy using Link-layer encryption and legacy pairing. An attacker can modify packets being sent to the device causing incorrect information to be outputted. The dangers of vulnerabilities affect personal and medical devices and can become more serious based on the circumstance. The tool developed by the team allows for the detection and remediation of the victim's device to prevent them from harm if they choose to protect themselves.

##### Acknowledgments

We would like to acknowledge that this work, done in correspondence with Lockheed Martin, was done by George Mason University Cybersecurity Engineering Students. We also want to acknowledge our Sponsors, Christina Olk and Chase Whitt, and our mentor, Jamie Wheeler, for their guidance throughout this project.

##### References

[1] A. Nguyen, “BleedingTooth: Linux Bluetooth Zero-Click Remote Code Execution,” GitHub, Apr. 06, 2021. <https://github.com/google/security-> research/tree/master/pocs/linux/bleedingtooth (accessed Apr. 18, 2022).

[2] M. Ryan, “Crackle,” GitHub, Apr. 13, 2020. https://github.com/mikeryan/crackle (accessed Apr. 18, 2022).

‌