A Bad Side of Pi

Implementing a Covert-Channel Virus on Raspberry Pi

Ben Davis
bdavis41@calpoly.edu
California Polytechnic State University
San Luis Obispo, California

Ayan Patel
apatel60@calpoly.edu
California Polytechnic State University
San Luis Obispo, California

Kun-Pu Lee
klee225@calpoly.edu
California Polytechnic State University
San Luis Obispo, California

ABSTRACT

Viruses are a very common threat to modern systems. Many viruses rely on the use of the Internet to transmit data back to a malicious entity. However, The Internet is liable to expose the attacker source and the virus can be perceived easily. By using a covert channel attack, we are capable of transmitting data from a Raspberry Pi over radio frequencies(RF) by its specific GPIO pin. We create virus containing a key-logger and a network sniffer. It also includes a transmitter to transmit a bytestream of data using RF. To ensure that there is no errors over the communication, we use the Hamming Code for forward error correction. Moreover, we attempt to transmit our digital information through Amplitude Shift Keying(ASK) and Frequency Shift Keying(FSK). On another computer nearby, we setup a receiver with an SDR to capture data and convert it back into readable data so that we can examine the log files on your own computer.

1 INTRODUCTION

A virus is one of the most common threats posed to modern computer systems. Many common viruses such as key-loggers and data sniffers rely on using Ethernet or Wi-Fi in order to transmit their ill gotten data back to the malicious entity. However, using these methods exposes the virus to additional avenues of detection. In order to make the virus more survivable among computer systems, we will be using a side-channel attack against the GPIO pins of the device to broadcast the data over a variety of RF transmission approach. By using this method of transmission, it is harder for a user to identify the data leakage.

2 BACKGROUND

In this section we describe two of the underlying technologies we use.

2.1 Virus

Modern computer viruses perform many malicious tasks once they've gained access to user machines. Among these are leaking user files, slowing the machine, and monitoring user activities. For the purpose of developing a simple virus to demonstrate the ability of our covert channel we have opted to leak two types of information. The first is key-logging which requires a low data rate to leak. During this sort of attack the user's keyboard buffer is read to determine which keys are being pressed. This enables confidential IP and passwords to be leaked without accessing any particular file. The second set of information which is being leaked is the devices network input traffic. This is information requires a high data transfer rate in order to completely leak, so it will likely prove extremely taxing on the covert channels which will be used to leak the data. We have elected to leak these two types of information because they provide meaningful information about the system's users while providing two different bandwidth necessities to properly transmit.

2.2 Modulation Schemes

Wireless communication requires the transmitter and receiver to understand how the RF waves will be manipulated to send particular bits of data across the channel. There is a wide array of modulation schemes; however, in this section we will briefly discuss the two simplest schemes which are used in our design.

Amplitude Shift Keying (ASK) is a method of transmitting digital information via RF waves through modulating the amplitude (strength) of the signal. This can be done in multiple steps to represent multiple bits or it can be simply represent a single bit as in Binary Amplitude Shift Keying (BASK).

Frequency Shift Keying (FSK) is an alternative method which relies on modulating the RF frequency to represent a set of bits. Binary Frequency Shift Keying (BFSK) is done using two distinct RF frequencies to represent a single bit at a time.

2.3 Hamming Code

In order to maximize range in an RF application it is beneficial to be able to recover from errors within transmission process. Accomplishing this can be done thought a process called Forward Error Correction (FEC) coding. FEC coding can be thought of as adding additional information to the data being transferred in order to recover from random bit flips within the received stream. One of the oldest methods of FEC is called Hamming Code.

Hamming code adds additional bits which send information about the parity of original data[10]. This allows a single error to be corrected. For a byte this is done using an additional 4 bits. These parity bits are placed in power of 2 bit positions (1 indexed). Each of these parity bits is then set according to the parity of the bits who reside in indexes that contain the binary representation of the parity bits index. This means the parity bit in index 1 is the parity of all bits who's binary index has a '1' in the right most position and the parity bit in index 2 is the parity of all bits who's binary index has a '1' in the second position (Figure 1). In order to determine if any errors occurred it is simply a matter of checking each of the parity bits. If any of the parity bits are found to be in error then it is possible to determine which bit is in error and change it. The bit in error is determined by finding the bit index which corresponds to all of the incorrect parity bits. For example if the parity bits at 1 and 8 are incorrect then the bit which flipped was the bit at index 9. Resolving this is a simple matter of flipping the bit back.

3 DESIGN

In this section we first describe our hardware configuration for this project. We then cover the design of our virus and convert-channel. Design decisions is also discussed.

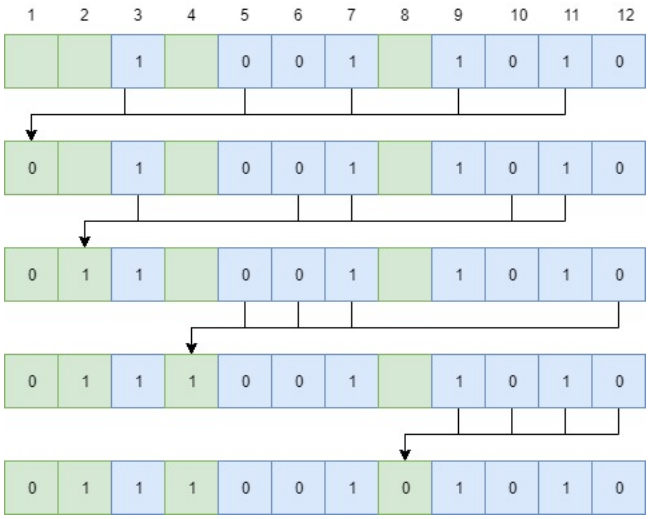

Figure 1: Hamming Code Computation

3.1 Hardware Configuration

We chose Raspberry Pi 3 as our victim computer due to its accessibility to its hardware configuration and operation system. We used the GPIO pin4 on the board as an antenna to transmit the key-logger file. In addition, plugging a hard wire to the GPIO pin4 can extend the transmission distance significantly.

For the receiver, We first chose a low cost($30 on Amazon) software-defined radio(SDR),RTL-2832U. We were trying to prove that an amateur level SDR can implement RF hacking easily. Unfortunately, we have found many issues such as an antenna resonance point which interfered our signals identification transmitted from the Raspberry Pi 3. Rather than using an amateur SDR, we eventually chose USRP B205mini-i as our receiver for this project. B205mini-i is commonly used in industry for research and development. It also costs more than the RTL-2832U as a $1000 market price. We also found that B204mini-i can receive signals clearly from a Raspberry Pi without plugging an hard wire. On the other hand, RTL-2832U cannot receive any signals if the we don't plug in a hard wire on the Raspberry Pi.

After the receiver captured the transmitted signals, the signals propagated to our PC for further signal processing. We then used GNURadio software for signal

processing. A GNU Radio is an open source software designed for software-defined radio users building their own module through the signal processing block[1]. With the aid of tutorials from some online tutorials[2] [3], we were able to build a basic FM receiver which its center frequency and RF gain are tunable for monitoring the FFT plot and the waterfall plot.

3.2 Virus Design

We create a virus to be put on a Raspberry Pi system that includes a key-logger and a network sniffer. This virus must operate in real-time and be able to capture outside of the user space when given root permission. In order to accomplish the real-time performance desired it is necessary to prevent the use of blocking functions and system calls. This virus then saves this data in a log file until it is ready to be transmitted over the covert channel.

3.2.1 Attack Vector. Because the complex nature of privilege execution is beyond the scope of this project we did not implement attack and propagation methods into our virus. Instead we will leave the privilege escalation to other papers. One method used to gain root privileges within Linux systems involves a row-hammer attack against the DRAM in which a memory row adjacent to the target row is repeatedly set causing a static charge to build on the target row changing the value. [13] Cozzi et al. discuss other methods which malware is able to perform privilege execution against Linux machines. [5]

The propagation method of this malware is also beyond the scope of the project. Several approached can be implemented including phishing and self-propagation. However, we leave a more thorough discussion of these for more focused papers. [6, 14]

Because the use of covert-channels is common on air-gapped networks it is possible that none of these traditional methods would be unsuccessful at infecting a target system. However, attacking these more secure systems is not impossible. One solution which would solve infection and, potentially, privilege escalation is the use of a disgruntled employee to perform the infection and attack. This sort of insertion vector is not uncommon, however it is usually relatively simple to defend against by properly limiting employee privileges before informing them they are fired. There are other methods which are able to bypass the air-gap. This including providing an unsuspecting employee with an infected USB drive such as was used in Stuxnet. [11]

3.2.2 Virus Flow. The virus has two main parts: a key-logger and a network packet sniffer. Once the virus is installed on the victim's Raspberry Pi, it runs in a loop lasting indefinitely starting at boot, checking for new key strokes and new packets each time. This will require root privileges which we assume are granted by the attacker. The data from both parts will be stored in a temporary file to then be transmitted over the GPIO pins through a radio frequency. Another Raspberry Pi will act as an attacker's receiver to receive the temporary files transmitted from victim's Raspberry Pi. Radio frequencies and maximum transmit distance will be determine in the future work.

3.2.3 Key-logger Implementation. The key-logger in this project utilizes the 'showkey' Linux command in a bash script loop that will send data to a C program which will parse the codes and print characters for each key stroke in a file. It also records dates and time so that attacker can examine the victim's activity precisely. In order to parse the codes, we have to load a keymap.txt into our C program first. keymap.txt acts like a look-up table to decode each keystroke so that the keycode can be converted into an alphabet or number. Afterward, keyed sentences, date and time will be record in a output.log file.

At first, we attempted to read key strokes from the keyboard buffer. This can be done but was slightly more difficult and does not scale to other systems. The keyboard buffer name stored in /dev/input/ has a different file name depending on the Linux operating system as well as the type of keyboard. To be able to deploy the virus on a larger variety of systems, we chose the path using the 'showkey' Linux command.

3.2.4 Network Sniffer Implementation. In order to implement network sniffing we used the functionality provided in the PCAP library. In order to make this non-blocking we used the dispatch function to handle any packet received on the device. Using this approach has the advantage that all input traffic it captured regardless of the port, protocol, or target application. Because of the limitations of the covert output channel it is also necessary to prioritize packets since it is unlikely that

all packets will be able to be sent. To this end the data-link header is stripped off of received packets and they protocol type is determined and used to set the priority of this packet. We write packet data to a file which will later be accessed in order to transmit the data. The functionality of this virus was verified to work across users when the virus was being run in root.

3.3 Covert-Channel Design

Within this section we describe the design of our RF protocol used in the covert channel. We also discuss how this is implemented on our Raspberry Pi within software.

3.3.1 Channel Modulation and Frequency. We seek to allow for the use of two forms of modulation so that which ever is determined to be the best fit for the application during testing can be used. Because of the extreme simplicity of transmission over short length GPIO pin we are limited to fairly simple modulation methods. We opted to use BASK and BFSK because of their natural resilience to channel noise and their ease of implementation.

We decided to use 200MHz as our center frequency. This decision was influenced primarily by the limitations of the device, legal ramifications, and likely-hood of detection. The Raspberry Pi's maximum theoretical broadcasting frequency is 650MHZ and as low as 2.55MHz. However operating at 650MHz presents the potential problem of inconsistent clock frequency transfer to the GPIO. Using the low end of the radio spectrum proves difficult using a GPIO pin because the small antenna length causes the relative transmission power to suffer and as such the RF wave produced has difficulty distinguishing itself from RF noise. The decision not to use a frequency near 88MHz to 108MHz is primarily due the increased likely-hood of detection from common FM radio transmitters. Other frequencies in the low 100MHz range tend to be legally reserved for satellite communications. The high 200MHz and 300MHz frequencies are allocated for military and "Fixed Mobile" usage.[4] Once we go above the 400MHz range we also would expect to see destructive interference between multiple pin broadcasting which would minimize the effectiveness of our ASK. The 174-216MHz radio frequency however is allocated in the FCC table as a low-power auxiliary broadcast channel which makes it ideal for our purposes.[4] In order to avoid excessive leakage outside this range caused by the square wave generated off the Raspberry Pi we opted for the center of this frequency.

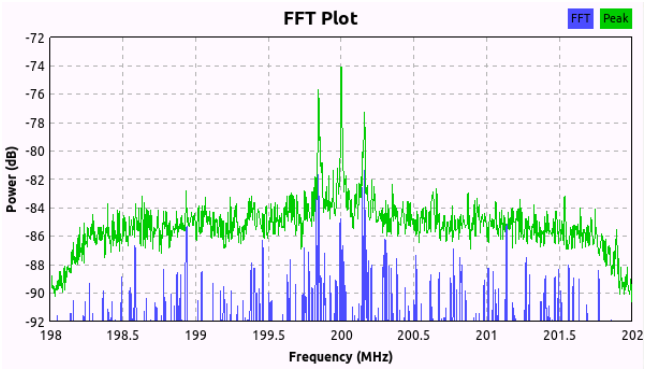

Figure 2: Peaks at 199.7 and 200.2 MHz Without Antenna

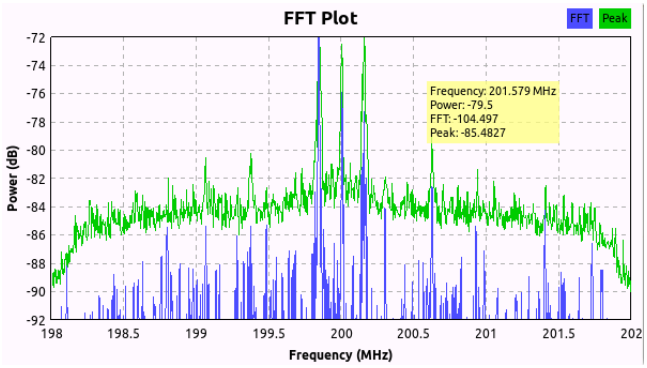

Figure 3: Peaks at 199.7 and 200.2 MHz With Antenna

3.3.2 Packet Format. We have opted to use a packetized format in order to send individual bytes of data at a time. We do this using an asynchronous packet format. Each transmission packet begins with "101" as its clock synchronization and start signal. This is then followed by 12 bits of an FEC coded data byte. The FEC we will be using is Hamming code for its relative simplicity (see Section 2.3). Then the packet is terminated with '0' trail bit. The default channel state is low to allow for easy detection of the start bit. Figure 4 shows the overall structure of our packets.

3.3.3 Implementation on Raspberry Pi GPIO. In this section we discuss the implementation of our cover-channel's transmission on the Raspberry Pi. This is a

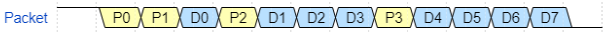

Figure 4: Data Transfer Packet

fairly simple task when looked at holistically, however it contains a variety of moving parts which must be considered to minimize transmission jitter.

4 TESTING SETUP

4.1 Virus Testing

Once we implemented the virus, we compiled it on a Raspberry Pi and began running it.Since we assume that a malicious disgruntled employee is installing the virus, we can assume they have or can get root privileges and install the virus through a download or a USB drive. Once the virus is on the system and running, it will stay running until the system is rebooted. If the user wants the virus to start again on every reboot, we assume they have the knowledge to write a bash script to add to the startup configuration.

We tested the virus by browsing sites and typing in information on the Raspberry Pi to ensure it matched the output log file. We had specific sites we visited as well as setup a specific webserver to visit to ensure packets actually matched what we were receiving. We typed into a text document and used different characters including backspace and special characters and then compared it to the log file we captured.

4.2 Covert-Channel Testing

4.2.1 Transmitter Testing. We tested the functionality of our transmission method by using a FFT on an SDR to monitor the broadcast channel. For the FSK these tests were done with an antenna attached to maximize the power of broadcast signal. For the ASK transmission our measurements were performed with the SDR antenna next to the GPIO pins. It is important to note the substantial harmonics which our square-wave signal generates.

4.2.2 Receiver Testing. We first built a basic FM receiver on GNURadio to ensure that the RTL-SDR we purchased online and the installed GNURadio software can work together for the future advanced receiver development. We then implemented an FSK using GNU-Radio. We were capable of tuning the desired FM frequency and RF gain for tracking the signals. Our GNU-Radio program outputs individual bits to a binary file which can then be decoded. For testing distance we broadcast 512 bytes of data and saved the received bits. We then developed a tool to take in these binary files and search for valid formed packets with the expected data values. Even in truly random noise this would still yield a non-zero number of valid packets. To mitigate this we used Bayesian statistics to determine the likelyhood that this packet was instead random data.

5 RESULTS

In this section we discuss the results of our tests. We begin by discussing the efficacy of the virus and then discuss the results of our transmitter and reciever pair.

5.1 Virus Results

We ran into some issues with accessing the keyboard buffer, so that was why we had to use the 'showkey' command instead. We were successfully able to capture both keyboard data and network packet data and store them into a file. After running the tests on the Pi with the virus running, we compared the resulting log files with the actual or expected data. For both files we found that we had 99 percent similarity and we got our expected results. In terms of special characters, there were some keys that we had not expected and therefore were ignored in our log file. For the network sniffer, we verified that what we received matched what we were sending from our web server.

5.2 Covert-Channel Results

We were able to successfully transmit over the covert using FSK. Figures 5 and 6 demonstrate best case transmission waves. We were able to attain transmission BAUD rates as high as 40000 symbols/sec. With our packets structure this translates to 2.666 kbit/sec throughput. We also verified the success of our FEC through recovering data with a single bit flipped.

The distance of the antenna receiving the data has to be fairly close to the Raspberry Pi, especially if the Pi does not have an antenna wire connected to the pin. With a wire plugged into GPIO pin4 on Raspberry Pi, the transmission distance then can be extended to

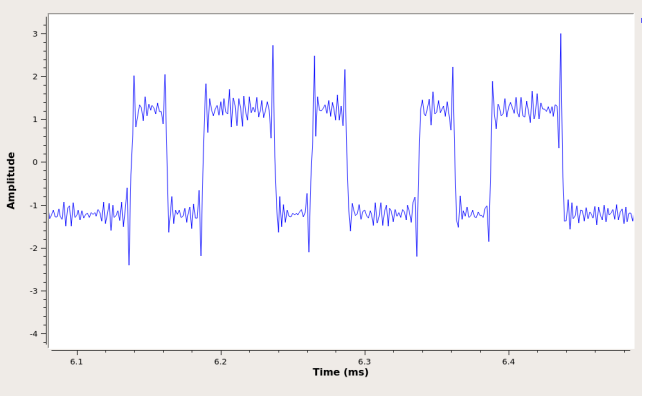

Figure 5: 0xAA Transmission Quadrature Demodulation

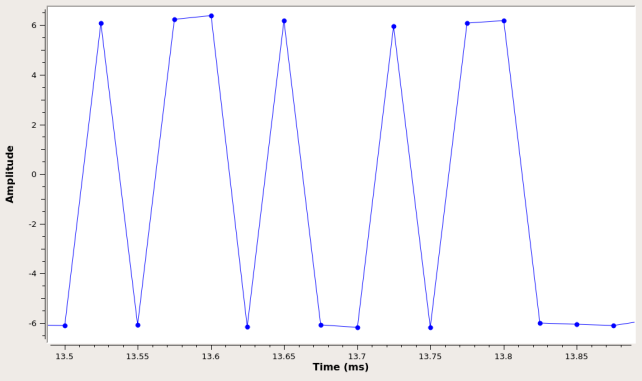

Figure 6: 0xAA Transmission Bit Points

around 2 meters with 75% packet loss. To achieve the same packet loss without a wire antenna we needed to be within 22 cm of the Raspberry Pi.

6 RELATED WORKS

In this section we discuss other current works in fields related to our virus design. We first discuss covert-channel works and then discuss privilege escalation attacks.

6.1 Covert-Channels

The use of covert-channels to circumvent air-gapped systems is one of particular interest to covert agencies as it allows the siphoning of data off of a computer without any direct connection to the internet. In this section we will describe some notable recent covert-channels.

All modern computers have a shared set of components which they require to operate. One of these components is a memory bus. Kachlon et al. developed and tested GSMem as a method exploiting this bus in order to covertly leak data off of the computer system[7]. Under normal operating conditions a memory-bus's side channel emissions tend to the be low; however, by sending specific values over the bus as specific frequencies it is possible force the bus to emit RF waves. These RF waves were used to create a BASK signal at the same frequency as is used by GSM cellular devices. Once the data is received by a nearby cellular device it is a relatively simple process to further propagate this data.

When attacking an Air-Gapped computer it can be difficult to find an infection method. One such infection method is using a bad actor and a USB drive. In order to build on this Guri et al. developed a covert channel through a USB device[8]. USB devices use a non-return-to-zero inverted coding scheme to send data over its adjacent D+ and D- connections. By driving these lines to they invert at a specific frequency it is possible to generate a target frequency emission off of these two bus signals. They showed this could be used to generate a BFSK to leak data.

In some cases RF covert-channels may not be a feasible way to leak data from an air-gapped system. In order to find another covert-channel which is sufficient in this case it it possible to leak data from an air-gapped network through status LEDs on routing equipment within the network[9]. The LEDs can then be recorded and computer vision techniques can be used to decode this data on the receiver side.

6.2 Privilege Escalation

There has been work done on privilege escalation vulnerabilities in Linux systems. In particular, one paper talks about kernel exploits. [12] Privilege escalation can be used to gain root access to a system to be able to run commands and access files normally denied. The specific kernel exploit this paper talks about is Dirty COW exploit.

The Dirty COW exploit took advantage of a race condition with copy on write. It allowed for an attacker to overwrite any file they chose without system permissions. An attacker can change the root password and gain access to the whole system. Almost all Linux

systems are vulnerable to this attack and many have been infected. Patches and updates exist to prevent the exploit but it requires all these systems to be updated.

Another type of attack is wild-card injection. An attacker can use a wild-card injection exploit to execute arbitrary code as root. One way to do this is to create a file name such that it includes parameters for a command that can run as root. A cron job to execute a TAR command can include parameters to execute a piece of code giving the attacker root privileges.

Another attack abuses the use of SUID programs that allow a low level user to execute a program that runs with root privileges. If this functionality is given to commands like nmap then its interactive mode will allow an attacker to execute arbitrary code as root.

The paper also talks about the importance of physical security. An attacker is more likely to gain access to a system is they can access USB ports, drives, and other hardware. Limiting an attacker to a keyboard may limit the exploits available, but it does not make it impossible. One such vulnerability lies in a system that uses LUKS encryption. An attacker can gain access to root level memory by simply pressing the enter key 93 times during a password check. From there the attacker can execute code to gain root access to the system.

7 COUNTER MEASURES

Through this paper we have described and explored the risks-posed to air-gapped computer networks through viruses which are able to leak data through over-air covert-channels. In this section we briefly explore possibilities of mitigating these types of attack.

Because most over-air channels tend to have a limited range, a zoning approach can be used as an effective mitigation method against known attacks[7]. In this method one seeks to control the devices which are allowed within a certain physical proximity of the air-gapped network. This serves to prevent these devices from being close enough to serve as a way-station between the cover-transmission and the attacker at large.

Zoning is not always feasible for some of the longer range covert-channels. Another possibility is the use of Faraday cages in order to prevent the transmission of data outside of physical zones. However, enclosing individual devices within Faraday cages does not seem practical and encasing entire rooms in effective Faraday cages seems a daunting task. Additionally, full room Faraday cages fail to defend against a listening device within the room.

Another possibility is the used of reactive jamming techniques. By monitoring a wide frequency range through sweeping or through random sampling it may be possible to detect abnormal behavior within the RF space in the room. Once this abnormal behaviour a jammer targeted at the abnormal frequency range could be used to effectively prevent data transmission on the covert-channel.

The final possibility is possibly the most obvious. Because all of these attacks require some form of malicious software to be running on the device effective anti-viruses which are built to monitor for behaviour which indicates known covert-channel attacks can be used to detect and stop viruses using internal monitoring. The most substantial problem this faces is the difficulty of detecting viruses and the up-keep required to keep an anti-virus on an air-gapped network updated.

8 CONCLUSION

This paper describes an approach to leak data from an air-gapped network using a virus and radio frequencies. From our experiments and tests we have found that without an antenna, the GPIO pins are not as strong and the transmission only goes a few inches away. Future work can look into the use of radio frequencies to transmit data using other means such as cables or other internal hardware like the GPU. Other covert channels can be further investigated to determine the best approach to attack an air-gapped network, while maximizing the distance of transmission.

REFERENCES

[1] Gnu radio. *http://www.gnuradio.org/*.

[2] Rf testing methodology by nccgroup.

[3] Blargh. Decoding fsk. *https://blog.habets.se/2017/04/Decoding-FSK.html*.

[4] Commission, F. C., et al. Fcc online table of frequency allocations. *Online:[http://www. fcc. gov/oet/spectrum/table/fcctable. pdf]* (2008).

[5] Cozzi, E., Graziano, M., Fratantonio, Y., and Balzarotti, D. Understanding linux malware. In *2018 IEEE Symposium on Security and Privacy (SP)* (2018), IEEE, pp. 161–175.

[6] Guo, H., Cheng, H. K., and Kelley, K. Impact of network structure on malware propagation: a growth curve perspective. *Journal of Management Information Systems 33*, 1 (2016), 296–325.

[7] Guri, M., Kachlon, A., Hasson, O., Kedma, G., Mirsky, Y., and Elovici, Y. Gsmem: Data exfiltration from air-gapped computers over {GSM} frequencies. In *24th {USENIX} Security Symposium ({USENIX} Security 15)* (2015), pp. 849–864.

[8] Guri, M., Monitz, M., and Elovici, Y. Usbee: air-gap covert-channel via electromagnetic emission from usb. In *2016 14th Annual Conference on Privacy, Security and Trust (PST)* (2016), IEEE, pp. 264–268.

[9] Guri, M., Zadov, B., Daidakulov, A., and Elovici, Y. xled: Covert data exfiltration from air-gapped networks via router leds. *arXiv preprint arXiv:1706.01140* (2017).

[10] Kumar, U., and Umashankar, B. Improved hamming code for error detection and correction. In *2007 2nd International Symposium on Wireless Pervasive Computing* (2007), IEEE.

[11] Larimer, J. An inside look at stuxnet. *IBM X-Force* (2010), 1–37.

[12] Long, M. Attack and defend: Linux privilege escalation techniques of 2016. *SANS Institute* (2016).

[13] Seaborn, M., and Dullien, T. Exploiting the dram rowhammer bug to gain kernel privileges. *Black Hat 15* (2015).

[14] Yu, S., Gu, G., Barnawi, A., Guo, S., and Stojmenovic, I. Malware propagation in large-scale networks. *IEEE Transactions on Knowledge and data engineering 27*, 1 (2015), 170–179.