Project Report
TIP-193

Hardware Security Analysis and Test Platform (HSATp) DRAM Bus Emissions Effort

B.V. John

8 May 2024

Lincoln Laboratory
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
LEXINGTON, MASSACHUSETTS

This material is based upon work supported by the Under Secretary of Defense for Research and Engineering under Air Force Contract No. FA8702-15-D-0001.

This report is the result of studies performed at Lincoln Laboratory, a federally funded research and development center operated by Massachusetts Institute of Technology. This material is based upon work supported by the Under Secretary of Defense for Research and Engineering under Air Force Contract No. FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Under Secretary of Defense for Research and Engineering.

Massachusetts Institute of Technology
Lincoln Laboratory

Hardware Security Analysis and Test Platform (HSATp) DRAM Bus Emissions Effort

B.V. John
Group 51

Project Report TIP-193

8 May 2024

This material is based upon work supported by the Under Secretary of Defense for Research and Engineering under Air Force Contract No. FA8702-15-D-0001.

Lexington Massachusetts

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From - To)*
08-05-2024	Project Report	

4. TITLE AND SUBTITLE

Hardware Security Analysis and Test Platform (HSATp) DRAM Bus Emissions Effort

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

B.V. John

5d. PROJECT NUMBER
TI15-1901

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

MIT Lincoln Laboratory
244 Wood Street
Lexington, MA 02421-6426

8. PERFORMING ORGANIZATION REPORT NUMBER

TIP-193

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
MIT Lincoln Laboratory
244 Wood Street
Lexington, MA 02421-6426

10. SPONSOR/MONITOR'S ACRONYM(S)
MIT LL

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT
DISTRIBUTION STATEMENT A. Approved for public release. Distribution is unlimited.

13. SUPPLEMENTARY NOTES

13. ABSTRACT

We present RAMBLE, a proof-of-concept method to transmit well-formed Bluetooth low energy (BLE) packets from a physically unmodified computer's memory bus. RAMBLE leverages DDR5-4800 memory operating at 4800 megatransfers per second. Because DDR performs two writes per clock cycle, DDR-4800 is clocked at 2.4 GHz, right in the middle of the industrial, scientific, and medical (ISM) frequency band. Carefully timing cache-bypassing writes enables modulation of the 2.4 GHz ISM-band clock signal to generate valid BLE packets. Because BLE packets are very short, the attack can be carried out in userspace without being interrupted by task switching. Although prior research has explored the feasibility of transmitting custom signals through EM side-channels, RAMBLE demonstrates the ability to extrapolate data from isolated, air-gapped networks using unmodified existing receivers and protocols, and with no dedicated transmission equipment. Alongside those other works that emanate using the DRAM bus, RAMBLE adds further urgency to the need to manage and mitigate malicious electromagnetic emissions.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**	None	36	**19b. TELEPHONE NUMBER** *(include area code)*
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED			

EXECUTIVE SUMMARY

We present RAMBLE, a proof-of-concept method to transmit well-formed Bluetooth low energy (BLE) packets from a physically unmodified computer's memory bus. RAMBLE leverages DDR5-4800 memory operating at 4800 megatransfers per second. Because DDR performs two writes per clock cycle, DDR-4800 is clocked at 2.4 GHz, right in the middle of the industrial, scientific, and medical (ISM) frequency band. Carefully timing cache-bypassing writes enables modulation of the 2.4 GHz ISM-band clock signal to generate valid BLE packets. Because BLE packets are very short, the attack can be carried out in userspace without being interrupted by task switching. Although prior research has explored the feasibility of transmitting custom signals through EM side-channels, RAMBLE demonstrates the ability to exfiltrate data from isolated, air-gapped networks using unmodified existing receivers and protocols, and with no dedicated transmission equipment. Alongside those other works that emanate using the DRAM bus, RAMBLE adds further urgency to the need to manage and mitigate malicious electromagnetic emissions.

This page intentionally left blank.

TABLE OF CONTENTS

This page intentionally left blank.

LIST OF FIGURES

This page intentionally left blank.

1. INTRODUCTION

There has been great interest in RF emanations from commodity computers. These emanations, if carefully controlled by malicious code, can transmit data over an airgap. Several previous works have explored the utility of the memory bus for this purpose, and have shown compelling results with large standoff distances or high data rates. However, all of those works have required customized receivers to properly receive and decode the emanations.

We have developed a technique to generate RF emanations from a memory bus that can be received and decoded by a commodity device speaking a standard RF protocol. Specifically, we present a proof-of-concept method to unidirectionally transmit Bluetooth low-energy (BLE) advertisement packets from a physically unmodified computer's memory bus. RAMBLE leverages DDR5-4800, operating at 2.4 GHz, directly adjacent to Bluetooth bands of interest. By carefully timing cache-bypassing writes, we are able to modulate the clock to generate valid BLE packets.

The remainder of the paper is organized as follows. Section 2 reviews related work on DRAM-based electromagnetic emissions. We describe the prerequisites of our attack in Section 3. Section 4 contains the technical background underpinning our attack. The attack itself is presented in Section 5 and its efficacy is evaluated in Section 6. We describe potential countermeasures in Section 7.

This page intentionally left blank.

2. RELATED WORK

There are many potential sources of unintended or malicious emissions from an ordinary computer, including microphones, USB buses, monitors, fans, and hard drives. We do not intend to slight these other works by omitting them in our review here, but for the sake of brevity, we focus on works that use the memory bus to generate emanations.

GSMem [1] by Guri et al. is a key paper regarding memory bus emanations. The paper demonstrates how to actuate the memory bus via amplitude shift keying (ASK) to modulate an AM signal on top of the 850 MHz bus frequency of DDR3 memory. They modified the baseband modem of a GSM cell phone to decode the ASK signal. It is important to note that the paper uses the 850 MHz GSM band, but the amplitude shift keying approach does *not* generate GSM waveforms.

Guri subsequently used DDR4 memory in Air-Fi [2] to reprise GSMem's amplitude shift keying technique on a 2400 MHz memory bus. Air-Fi demonstrated how to emanate on specific Wi-Fi bands by tuning the memory bus core frequency in BIOS. Unlike GSMem, no modifications had to be made to the receiver. Instead, they selected one of a handful of Wi-Fi devices with rudimentary spectrum analysis features. They could thus pull out the ASK signal directly. As with GSMem, Air-Fi does *not* generate 802.11 compatible waveforms.

BitJabber by Zhan et al. [3] focuses on increasing the data rate of the side channel. Like GSMem, it uses DDR3 memory and a center frequency in the 800 MHz range. BitJabber uses frequency shift keying, careful instruction selection, and sophisticated target address discovery to increase the signal strength. BitJabber uses a specialized antenna and hardware, not a modified cell phone, as its receiver. Like BitJabber, we deliberately introduce sideband noise to shift the emission frequency.

EMLoRa by Shen et al. [4] is an interesting complement to BitJabber: instead of throughput, EmLoRa focuses on standoff distance by using AM "chirps." The data rates are deliberately low, but EMLoRa demonstrates standoff distance substantially higher than that of GSMEM. Although inspired by an existing protocol, their technique requires a specialized receiver as well.

Lastly, NoiseSDR [5] presents an apotheosis of EM leakage. They demonstrate their technique across multiple processor types, including the x86-64 instruction set used by this work. The Android version of their toolchain is available on GitHub as well. The paper does not demonstrate the particular waveform we generate, nor does it use the same x86-64 processor instructions to actuate memory, but our work could conceivably be ported to the NoiseSDR model on an appropriate platform in the future.

This page intentionally left blank.

3. ATTACK MODEL

This work extends prior side-channel based data exfiltration works, and in particular shows how a computer with a RAMBLE malware can exfiltrate data to unmodified, BLE-capable devices. Further, as receiving BLE packets does not require the receiver software to have low-level access to the hardware, the receiver malware can be significantly less invasive than prior works (e.g., [1]), as it only has to use standard APIs for discovering local BLE devices. It is even possible to exfiltrate data without a specified receiver at all: using the Apple Find My network, you can spoof an offline AirTag causing nearby iPhones to automatically report the spoofed AirTags location to the cloud, enabling data exfiltration with no local presence [6].

The core of this attack only requires unprivileged userland code, but there are a few limitations. In order to generate BLE waveforms, the system must be using a sufficiently fast memory technology (such as DDR5), and the processor must support writing directly to memory (such as those with the `MOVDIRI` or `MOVDIR64B` CPUID feature flags). There are also a few BIOS-level configuration parameters that are critical for a malware to successfully exfiltrate data in the band needed for BLE packets. The DRAM must be operating at 4800 MT/s, which is a standard option but not guaranteed to be selected. The system must not have spread spectrum clocking (SSC) enabled. SSC is designed to spectrally spread the RF emitted from a system and inhibits the current implementation of RAMBLE by spreading the bandwidth by more than the width of a single BLE channel. It may be possible to time BLE transmissions at opportune phases of the spreading function, though this is a topic for future research.

Although this combination of configurations may be relatively rare to find in the wild, systems with the necessary hardware are only going to be more prevalent as the world continues to upgrade to modern technologies. As for setting the BIOS configuration parameters, the authors note that there is malware in the wild capable of reconfiguring BIOS/UEFI (e.g., [7]), and as such, the authors hypothesize that it is possible for an attacker to set the required DRAM speed and disable SSC.

This page intentionally left blank.

4. TECHNICAL BACKGROUND

4.1 DDR5 MEMORY

Double data rate (DDR) memory modules are commonly integrated into modern computers. They operate by transmitting data and commands on both the rising and falling edges of the clock, thus the data rate is double the clock speed. DDR5 memory is the latest standard, featuring higher performance with lower power than its predecessors, and comes available in a variety of speed ratings. DDR5 memory modules have two parallel 32-bit wide data channels, allowing for two independent simultaneous data transfers. In comparison, DDR4 and prior versions only had a single 64-bit data channel.

The initial release of DDR5 debuted with data rates up to 4800MT/s (million transfers per second) and has provisions for speeds of up to 8400MT/s [8]. As the data rate is twice the clock frequency, DDR5-4800 operates with a 2.4 GHz clock. This also means that DDR5-4800 modules emit EMI at 2.4 GHz, which can cause interference problems within the 2.4 GHz ISM band. To mitigate this, many motherboards implement spread spectrum clocking (SSC) to distribute the EMI energy across a larger bandwidth, making it easier to meet EMI compliance standards [9]. However, this feature is not mandatory and can often be easily disabled in BIOS settings.

4.2 BLUETOOTH LOW ENERGY

BLE is a wireless personal area network technology that operates in the same 2.4 GHz ISM band as classic Bluetooth, but is an independent and incompatible technology [10]. Its standard data rate is 1 Mbit/s, but has alternative modes from 125 kbit/s ("LE Coded S=8") to 2 Mbit/s ("2M PHY"). We focus on the low-speed S=8 mode in this paper, as its decreased symbol rate and increased error correction improve the transmissions range.

BLE operates with 40 channels, each 2 MHz wide with no guard band, evenly spaced across center frequencies from 2.402 GHz to 2.480 GHz. They are numbered sequentially excluding the three primary advertising channels: channel 37 at 2.402 GHz, 38 at 2.426 GHz, and 39 at 2.480 GHz, as shown in Figure 1. BLE transmitters alternate between these three primary advertising channels to initiate all transactions before hopping to a data channel to continue the transaction.

Every BLE packet contains a four-byte access address used to identify a single connection between two devices followed by 2-258 bytes of data. This is preceded by a preamble for synchronization and followed by a CRC field for error detection. Every connection begins with an advertisement on one of the primary advertising channels, and if more than a single packet of data is required, the connection migrates to the data channels as specified in the advertising packet. There are several types of advertising packets defined in the Bluetooth 5 specification. Of these, the `ADV_EXT_IND` combined with `AUX_ADV_IND` can be used to broadcast advertisements in LE Coded S=8 mode unidirectionally, without requiring any form of handshake.

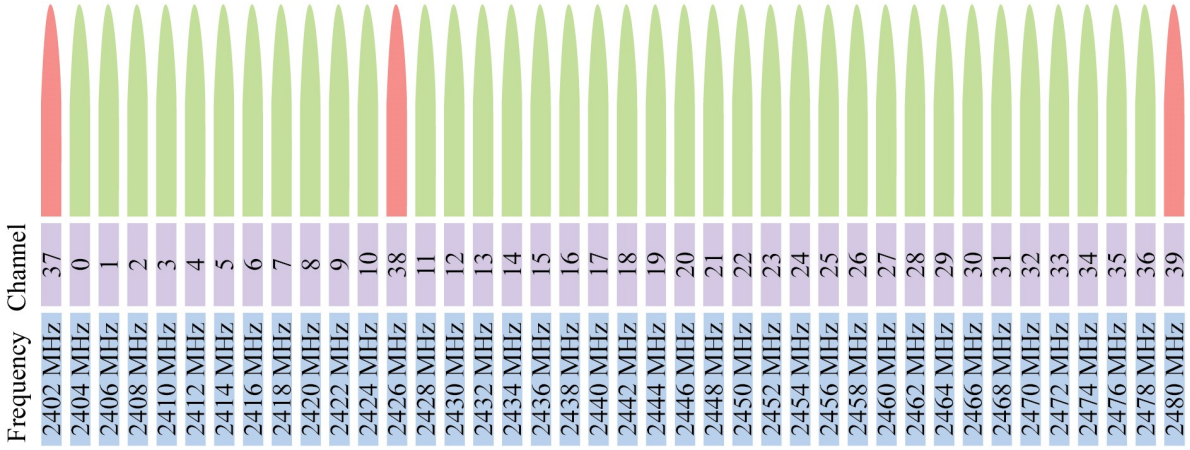

Figure 1. Layout of all 40 BLE channels. Channels 37–39, highlighted, are the advertising channels.

4.2.1 Modulation

The BLE standard requires Gaussian frequency shift keying (GFSK), which is a binary frequency shift keying (BFSK) pulse shaped with a low-pass Gaussian filter. The filter smooths the symbol transitions to reduce interference with neighboring channels. It is thus possible to transmit with BFSK without affecting data rates to the intended receiver, at the cost of increased interference to those transmitting on nearby frequencies.

4.2.2 Encoding

Standard and coded BLE modes operate with 1μs long symbols. A logical "0" is encoded as a tone in the range of -1 MHz to -0.185 MHz below the channels center frequency, and a logical "1" is encoded as a tone in the range of 0.185 MHz to 1 MHz above the center frequency.

LE Coded transmissions have improved range at the cost of a lower data rate than the base rate 1M PHY. In particular, the LE Coded S=8 mode decreases the throughput by a factor of eight. First, it uses forward error correction (FEC) to double the number of symbols per data bit. It then uses a pattern mapper, where every 0 bit is encoded as binary 0011 and 1 bits are encoded as 1100, thus further quadrupling the number of symbols per bit. As the pattern mapper guarantees that every symbol is sent twice in a row, we instead consider the baud rate of S=8 mode to be 500kbaud (2μs/symbol), and the pattern mapping to be $0 \rightarrow 01$, $1 \rightarrow 10$.

4.3 TERMINOLOGY

In this paper, we refer to several different layers of operations required to send BLE packets, and these layers have some overlapping terminology. In order to disambiguate terms, we will use the following definitions. A "burst" is a sequence of 16 clock transitions transferring 32 bits of data each between the DRAM module and memory controller. A burst transfers 64 bytes of information

in total, which is the same amount written by a single "write" operation issued by the processor. A "chunk" is a sequence of n writes issued in a single batch.

When discussing encoding data for transmission over BLE Coded S=8, a "symbol" is a 2μs interval where a low or high tone encodes a logical 0 or 1, respectively. A "packet" is then a sequence of symbols that make up a single transmission, as defined in the BLE specification [10].

This page intentionally left blank.

5. TRANSMITTER

As with prior works, RAMBLE uses uses periods of intense memory accesses to amplitude modulate the carrier signal provided by the 2.4 GHz DRAM bus clock. This carrier modulation creates lobes in its sidebands at an offset equal to (and multiples of) the rate of modulation. This allows for arbitrary tones to be generated around the carrier frequency, simply by performing memory accesses at a rate equal to the delta in frequency between the desired tone and the carrier frequency.

We use the x86-64 `MOVDIR64B` instruction to reliably stimulate the memory bus [11]. This direct-store instruction atomically writes 64 bytes to memory using the write combining protocol, so the data bypasses the cache hierarchy in its entirety and is written directly to memory. This allows us to easily saturate the memory bus when desired while still maintaining fine granularity; all memory controller transactions are a multiple of 64 bytes on our target system. This is also an improvement over using the `MOVNTI` instruction family discussed in prior works [3], as `MOVDIR64B` can be used indefinitely, whereas `MOVNTI` requires a `SERIALIZE` instruction every eight writes, which significantly slows down the maximum possible generated frequency.

To make sure that sequential writes are not clobbered, every `MOVDIR64B` instruction executed is written to a new address. For simplicity, this address is looped through a large buffer, incrementing the address by 1024 on every write to ensure that the writes alternate between the two DDR5 channels [12], thus maximizing the rate of memory accesses and by extension the signal strength. We note that although prior works [3] targeted a set of addresses distributed across memory banks, we did not find a significant impact on signal strength from addresses located in the same bank, so we did not implement similar measures.

We find that the values being written to memory have a significant impact on the resulting signal. This makes sense, as the RF energy being emitted by the RAM bus is going to come primarily from the wires transitioning states, and thus picking values that maximizes the number of transitions will maximize the energy emitted. As each DDR5 channel is 32 bits wide, writing a repeating sequence of `0x00000000_ffffffff_...` (32 bits low, 32 bits high) will transition every data wire on the channel on every half clock cycle, thus again maximizing the amplitude of the signal.

Controlling the timing of the memory accesses is fairly straightforward. To generate a tone with a delta of f from the DDR5 operating frequency, we must amplitude modulate our batches of writes at a frequency of f, and thus dispatch a new batch every $1/f$ seconds. Given that our target frequencies are measured in MHz, our timing requirements are on the order of tens of nanoseconds, so we use the `RDTSCP` instruction along with knowledge of the CPU's operating frequency to dispatch batches at precisely the right time.

The number of writes per batch is a critical parameter to maximize amplitude. A 50% duty cycle will achieve this, and the DRAM's clock rate and number of cycles required per burst can help estimate the optimal number of writes to issue per period. However, due to the various unknown overheads in the system, we found that the optimal number of writes was generally slightly below the theoretically computed value, and thus use empirical testing to optimize this value.

Appendix 1 includes sample code to generate a 19.5 MHz tone assuming a 3.2 GHz processor and a batch size of 8 writes per batch.

As discussed earlier, BLE symbols are a 2 μs long tone up to either 1 MHz above or below the channel center frequency. In order to achieve phase coherency, we choose tones situated at +/-0.5 MHz from the carrier frequency, as they will have an integer number of oscillations per symbol period. Thus to send a 2 μs tone at 2400 + $(n/2)$ MHz, transmit n batches of writes separated by $2/n$ μs. This system allows for a simple FSK implementation, as every symbol lasts precisely 2 μs, and it is easy to switch between frequencies to BFSK encode a precomputed BLE packet.

6. EVALUATION

6.1 EXPERIMENTAL SETUP

The transmitting system was tested in an open-air chassis with a MSI MEG Z690 ACE motherboard, a i9-12900k processor, and a single stick of 16GB DDR5-6000 memory (model number F5-6000J3636F16G). This system ran on Ubuntu Linux 22.04.2 LTS. SSC was disabled in the BIOS, and the RAM was locked to 4800 MT/s.

The entire system was placed in an RF isolation chamber, with multiple sensors available. We used an Ettus USRP B210 to capture raw waveforms for subsequent analysis, and a pair of Nordic Semi nRF52840-DK BLE USB dongles placed at various locations within the RF chamber to receive the BLE packets.

6.2 TONE GENERATION

Before generating BLE packets, we first characterized single tone generation using code much like that shown in Appendix 1. This step was important to optimize parameters such as the target frequencies selected for each channel (to avoid harmonics giving a false bit value) and then the number of writes for a given frequency. Figure 2 shows two sample tones for channel 37 (2.402 GHz), which shows how the 1.5 MHz tone falls cleanly in the middle of the 0-bit band with only minor interference in the upper range of the 1-bit band, and the 2.5 MHz tone strongly signals a 1-bit with very little noise in the 0-bit band. We then investigated the optimal number of writes for each tone. Figure 3 demonstrates how different tones require a different number of writes to optimize their signal strength.

6.3 BLUETOOTH LOW ENERGY — DIRECT TEST MODE

To demonstrate BLE, we used BLE direct test mode (DTM) to directly measure received packets in almost the same way that BLE devices are compliance tested. This allowed us to transmit packets on arbitrary channels and collect them at the maximum bandwidth, without having to deal with link layer/advertising overhead. With this tool, we could measure the exact rate of successful BLE transmission on a per-channel basis, allowing us to show how under our test setup some channels are far better than others.

Our test system was configured to transmit a standard BLE DTM packet every 1.4 ms, which included 784 μs for the packet, and 625 μs of downtime between packets (the minimum delay between packets per [10] is 150 μs, but we found better results with a slightly longer delay). This yielded roughly 700 packets transmitted per second, or 42,500 packets per minute. We then recorded received packets for a one minute span for every combination of channel and memory data pattern (discussed below), and found that under ideal conditions we could achieve a peak received packet throughput rate of 17,614 packets per minute, or 41%.

We found that different channels had significantly varying receive rates (see Figure 4) that did not correlate with the measured signal strength per channel seen in Section 6.2. We hypothesize

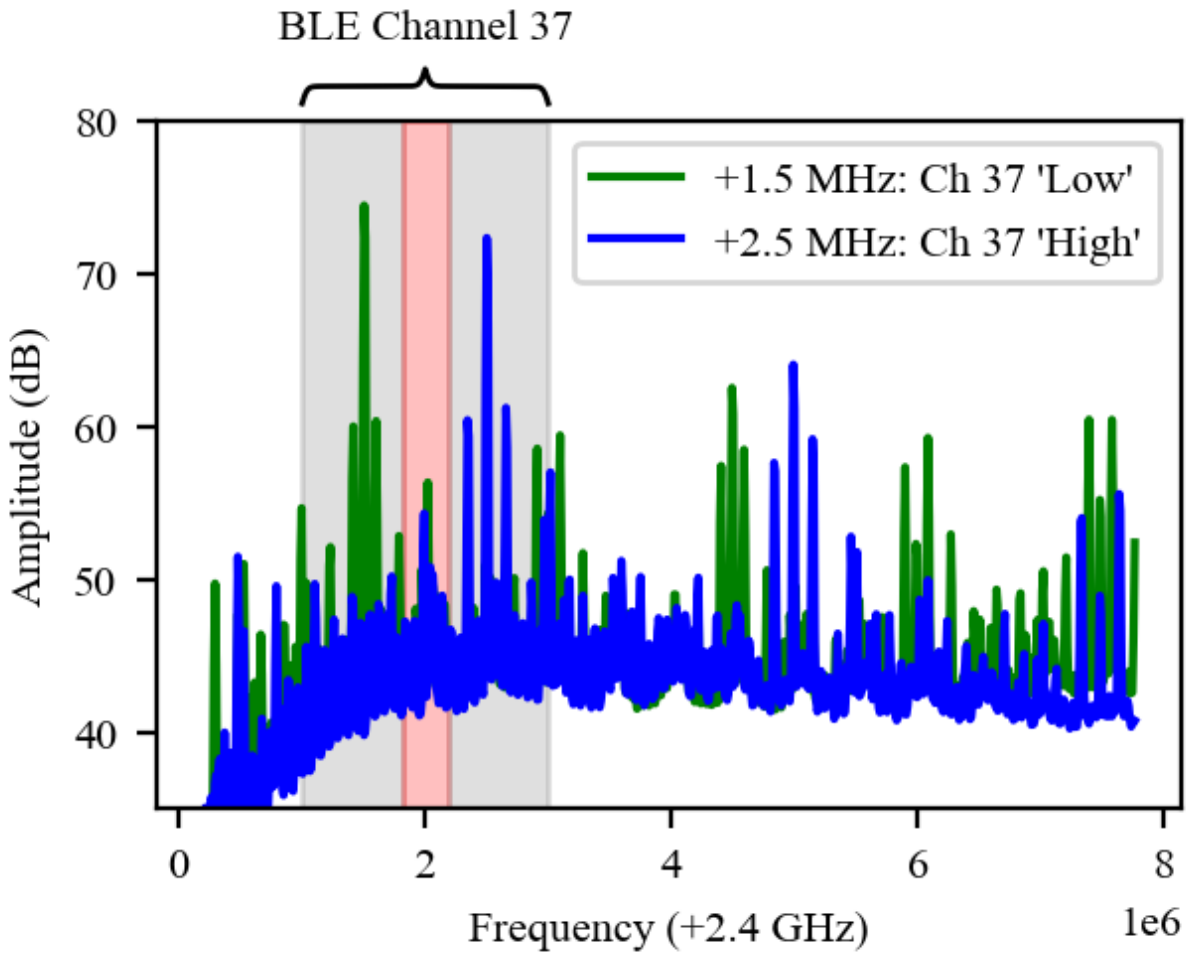

Figure 2. Spectra of 1.5 MHz and 2.5 MHz tones for transmitting 0 and 1 symbols on channel 37.

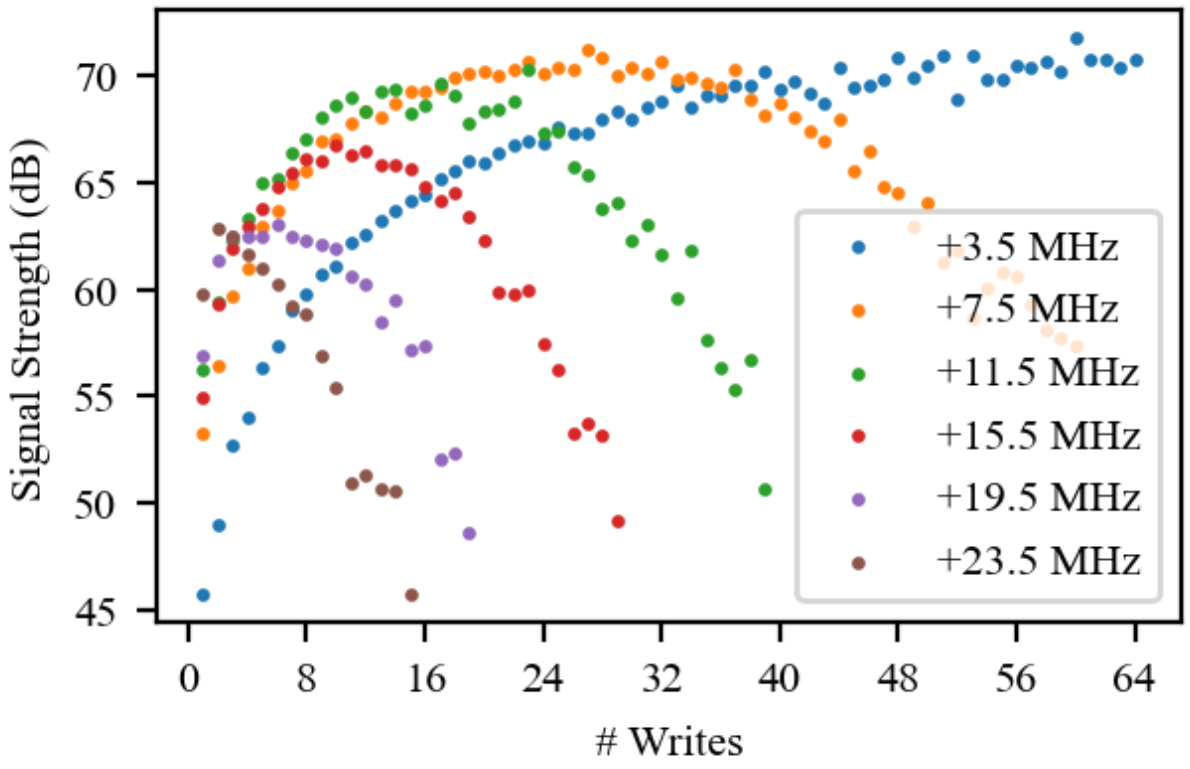

Figure 3. Signal strength of various tones vs. number of writes, showing that lower frequency tones require more writes to maximize amplitude.

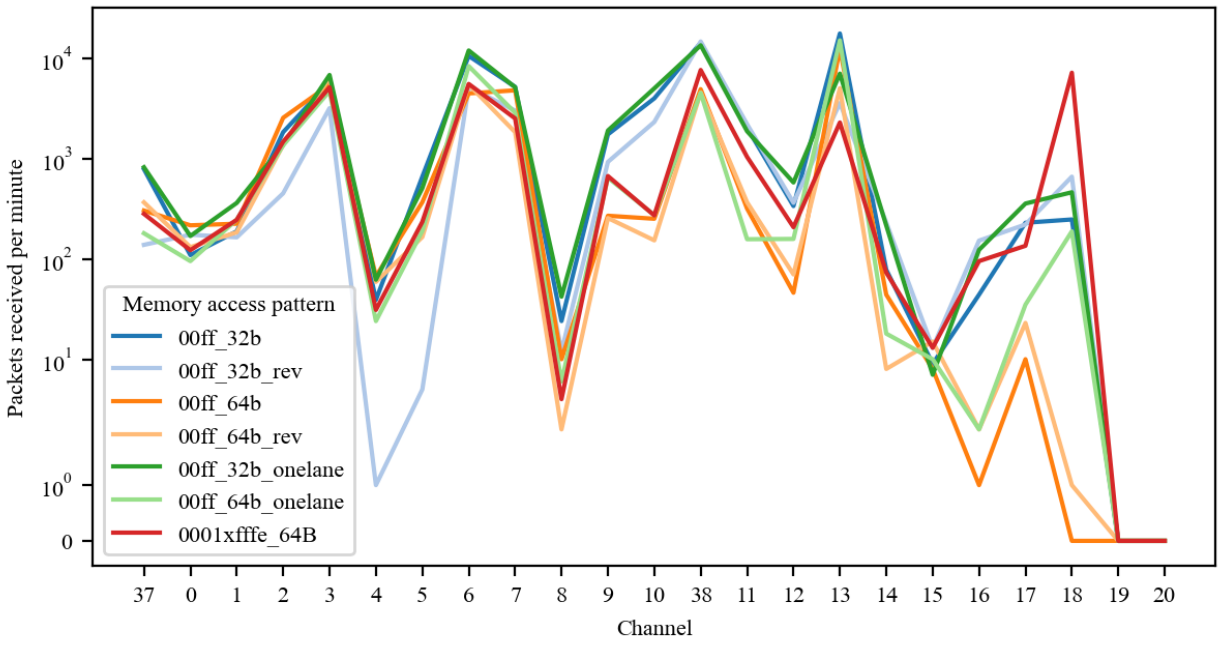

Figure 4. Rate of packets received (per minute) across the lower 23 channels, comparing various access patterns.

that this could be related to multipath effects in the enclosed RF chamber, as the received packet rate was strongly affected by even minor adjustments in antenna positioning, and certain channels had much better reception rates in some locations while others were better in other locations. The entirety of the presented dataset shows the best result for a given experiment from the two stationary antennas.

We also investigated various options for the values written to memory ("access pattern"), Figure 4 showing some of the more promising candidates. Most access patterns were based on a sequence of 32 (`00ff_32b*`) or 64 (`00ff_64b*`) 0 bits followed by an equal number of 1 bits, repeated until the full 64-byte sequence was complete. We tested sending only that constant value (`00ff_32b`, `00ff_64b`) for all transactions; inverting all bits on every other 64 byte write (`* rev`); and even writing to only a single DDR5 channel (`* onelane`) by writing to locations that avoided flipping bit 9 of the physical address [12]. Finally we had `0001xfffe_64B`, a pattern where we wrote all 0s for 64 bytes followed by all 1s for 64 bytes.

Overall, the 32 bit patterns tended to do the best as expected, though the 64-bit patterns were rarely far behind and sometimes surpassed the 32-bit patterns. We have yet to determine why the `0001xfffe_64B` pattern did so well on channel 18, or more broadly so well in general[1]. We also note that the memory access pattern had some impact on which of the two receivers got a clearer

[1] Neither pattern was all 0s or all 1s, as that can trigger a special DDR5 power-saving "write pattern" mode that doesn't send the entire data stream. Instead, we always flipped the LSB so as to avoid triggering this mode.

signal, implying that the memory pattern has some form of beam steering effect. This aligns with the fact that each DDR5 channel is a parallel but physically separate channel, meaning that when their data wires transition, they can constructively or destructively interfere depending on the path the RF takes and the precise phase alignment of the two channels. The precise relationship between memory patterns and optimal RF signal generation is a topic for future research.

This page intentionally left blank.

7. COUNTERMEASURES

There are several possible defensive countermeasures available to guard against RAMBLE attacks. First is shielding—the signal strength of this side channel is fairly weak, so good metallic shielding around the computer can effectively eliminate the possibility of signal reception. If there are any gaps in the shielding, be it an incorrectly assembled case, one with an acrylic or glass panel, or for any other reason, the signal can leak out and be received from a distance of a few meters. As a backup measure, enforcing a minimum physical separation of BLE devices from all secure computers can help ensure the signal is below the noise floor before it reaches any potential receiver. This method is commonly used in extremely high security environments—after all, preventing all electronic devices from approaching a system guarantees that there are no receivers around to detect the signal, and thus data will be unable to be exfiltrated. The standoff distance required for such an approach may be a moving target as the field continues to explore the space.

Passive BLE monitors are an enticing idea, but likely not a feasible solution. As discussed above, the reception rate is highly location dependent, and so it would take several BLE monitors in immediate proximity to the computer to guarantee all BLE packets are detected. Likewise, prior works have discussed runtime detection, but the operations used in this work are fairly simple and could be hidden in a legitimate-looking software. Finally, although this work specifically sought to transmit well-formed BLE packets to otherwise unwitting, ordinary receivers, in an adversarial situation where a custom receiver and wire format is within the design space, detecting BLE specifically becomes less important.

This page intentionally left blank.

8. CONCLUSION

Several foundational papers have explored the possibilities of the DRAM bus in commodity computers as an EM emitter. We present RAMBLE, to our knowledge the first such effort to successfully transmit well-formed packets on a common wireless protocol. Through careful and precise choices of timing, memory access patterns, and write buffer contents, we have demonstrated the ability to transmit Bluetooth low-energy (BLE) packets that can be received and processed by an unmodified, commodity BLE device.

RAMBLE demonstrated a transmission success rate of up to 41%, or 300 BLE packets per second, which shows promise for a very high data exfiltration rate—on the order of 500 kbaud or more. Furthermore, it can be done without requiring custom receiver, and instead can rely on devices potentially already near the target.

We feel the design space is not yet fully explored. Better understanding of channel interference, write pattern effects on buses, and tighter timing may allow us to increase the viable standoff distance.

This page intentionally left blank.

REFERENCES

[1] M. Guri, A. Kachlon, O. Hasson, G. Kedma, Y. Mirsky, and Y. Elovici, "{GSMem}: Data exfiltration from {Air-Gapped} computers over {GSM} frequencies," in *24th USENIX Security Symposium (USENIX Security 15)* (2015), pp. 849–864.

[2] M. Guri, "Air-fi: Generating covert wi-fi signals from air-gapped computers," *arXiv preprint arXiv:2012.06884* (2020).

[3] Z. Zhan, Z. Zhang, and X. Koutsoukos, "Bitjabber: The worlds fastest electromagnetic covert channel," in *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, IEEE (2020), pp. 35–45.

[4] C. Shen, T. Liu, J. Huang, and R. Tan, "When lora meets emr: Electromagnetic covert channels can be super resilient," in *2021 IEEE Symposium on Security and Privacy (SP)*, IEEE (2021), pp. 1304–1317.

[5] G. Camurati and A. Francillon, "Noise-sdr: Arbitrary modulation of electromagnetic noise from unprivileged software and its impact on emission security," in *2022 IEEE Symposium on Security and Privacy (SP)*, IEEE (2022), pp. 1193–1210.

[6] F. Bräunlein, "Send my: Arbitrary data transmission via apple's find my network," (2021), URL https://positive.security/blog/send-my.

[7] "Guidance for investigating attacks using cve-2022-21894: The blacklotus campaign," (2023), URL https://www.microsoft.com/en-us/security/blog/2023/04/11/guidance-for-investigating-attacks-using-cve-2022-21894-the-blacklotus-campaign/.

[8] JEDEC Solid State Technology Association, *JESD79-5B_v1.20: JEDEC Standard for DDR5 SDRAM* (2022).

[9] "Altpll (phase-locked loop) ip core user guide: Spread-spectrum clocking," URL https://www.intel.com/content/www/us/en/docs/programmable/683732/17-0/spread-spectrum-clocking.html.

[10] Bluetooth Core Specification Working Group, *Bluetooth Core Specification v5.4* (2023).

[11] Intel, *Intel 64 and IA-34 Architectures Software Developer's Manual (325462-081US)* (2023).

[12] Y. Tobah, "Discussions regarding DDR5 addressing schemes," Personal communication (2022).

This page intentionally left blank.

A SAMPLE RAMBLE CODE

The following code generates a 19.5 MHz tone assuming a 3.2 GHz processor clock, and a batch size of 8 writes per batch.

```
#define mem_size 32*1024*1024*32

uint64_t cycles = 164; //3.2 GHz / 19.5MHz
uint64_t stepsize = 1024;
long long int i = 0;

char *memory_base = (char *) aligned_alloc(64, mem_size);

unsigned dummy;
uint64_t pattern[] = {
    0xffffffff00000000, 0xffffffff00000000,
    0xffffffff00000000, 0xffffffff00000000,
    0xffffffff00000000, 0xffffffff00000000,
    0xffffffff00000000, 0xffffffff00000000};

uint64_t nextTimeStep = __rdtscp(&dummy);

while (true) {
    char *mem2 = memory_base + (i & 0xfff)*64;
    asm volatile (
        "movdir64b %[pattern], %[loc] \n\t"
        "add %[stepsize], %[loc] \n\t"
        "movdir64b %[pattern], %[loc] \n\t"
        "add %[stepsize], %[loc] \n\t"
        "movdir64b %[pattern], %[loc] \n\t"
        "add %[stepsize], %[loc] \n\t"
        "movdir64b %[pattern], %[loc] \n\t"
        "add %[stepsize], %[loc] \n\t"
        "movdir64b %[pattern], %[loc] \n\t"
        "add %[stepsize], %[loc] \n\t"
        "movdir64b %[pattern], %[loc] \n\t"
        "add %[stepsize], %[loc] \n\t"
        "movdir64b %[pattern], %[loc] \n\t"
        "add %[stepsize], %[loc] \n\t"
        "movdir64b %[pattern], %[loc] \n\t"
        "add %[stepsize], %[loc] \n\t"
        :
        : [loc] "r"(mem2), [stepsize] "g"(stepsize),
          [pattern] "m"(pattern) );
    nextTimeStep += cycles;

    i++;
    while (nextTimeStep > __rdtscp(&dummy))
        {} //wait
```

```
}
```