Security Analysis of Taekwondo Sparring Gear

Elizabeth Zou, Claire Nord, Justin Lim, Peter Tran

May 15, 2019

1 Introduction

High-level taekwondo sparring competitions use electronic sparring gear, where points are awarded for kicks that register as strong enough (past a certain threshold of power). As stakes get higher, such as in the Olympics, there might be motivations to tamper with these systems for an unfair advantage. We analyzed the security of the TrueScore taekwondo sparring system with respect to its confidentiality, integrity, and availability. Our results show that the TrueScore system has several potential vulnerabilities in its e-sock magnets, the dongle-computer USB datastream, the memory used by the TrueScore software, and the RF communication between the hogus and controllers. We also describe avenues for future work such as spoofing the USB dongle and conducting RF replay attacks.

2 Background

To determine our threat model for this analysis, we first sought historical examples of cheating in professional sports with electronic scoring. We focused our research on electronic or mechanical cheating rather than doping.

The earliest recorded example occurred in a fencing match in the 1976 Montreal Olympics. At the time, épée matches were scored with an electronic system. The weapons were fitted with a “spring-loaded point connected by fine wires that run down a groove in the blade to a plug inside the bell-shaped hand guard”. The fencer also wore a “body wire” that ran from a spool on an electrically neutral strip on the competition ground through the inside of their jacket sleeve. Once the fencer was ready to start a match, they connected their body wire to the plug in their hand guard. The trunk of each fencer’s body was covered with a “metallic overjacket”, which would complete an electrical circuit and light up the scoreboard if it detected at least 750 grams of pressure [1].

In this fencing match between the British pentathlete Jim Fox and the Soviet Union pentathlete Boris Onischenko, judges discovered that Onischenko had a button installed in his épée that registered a hit every time he pressed it [2]. In particular, there was a piece of wire connected to a thumb button hidden under the épée’s leather binding [3]. Onischenko was disqualified from the Olympics, and Fox was flustered for the rest of the pentathlon.

We also found instances of cheating in taekwondo matches. In 2010, Taiwanese athlete Yang Shu-chun was disqualified from a match for placing additional electronic sensors (i.e. magnets) in a patch behind her heel [4]. These additional magnets could make it easier for her to score points. She was later disqualified from matches for three months by the World Taekwondo Federation (WTF). The WTF also disqualified her coach for 20 months and fined the Chinese Taipei Taekwondo Association $50,000 [5]. This event and the resulting controversy were dubbed ”Sockgate”.

3 Responsible Disclosure

Before attempting this project, we obtained verbal and written permission from Jin Song, the president of TrueScore, to proceed. Our project is based on the Gen1 Daedo equipment, and TrueScore has elected to

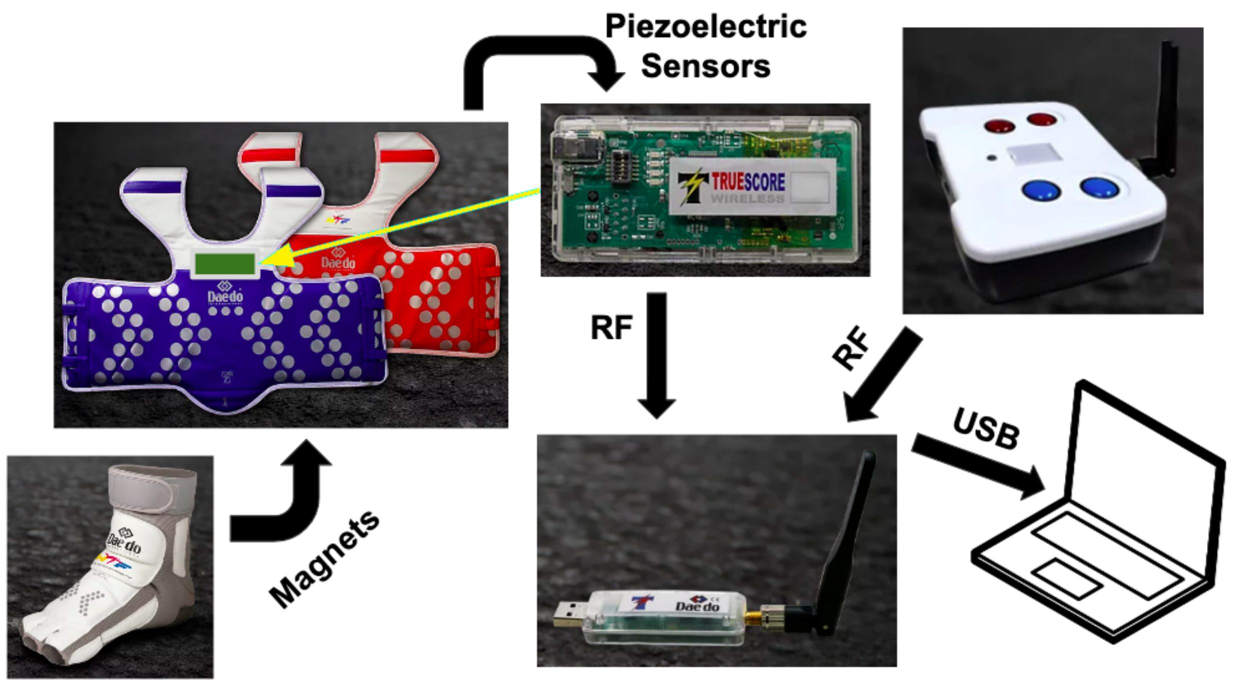


Figure 1: A general overview of the system.

delay the publication of this report by 6 months.

4 System Overview

The taekwondo electronic sparring gear system has six main components (Figure 1).

4.1 Electronic Hogu (Body Shield)

When the electronic hogu (Figure 2) is hit, one or more magnetometers in the hogu first determine whether the impact is from a kick. If it is from a kick, piezoelectric force sensors evaluate the power of the kick by the level of distortion of the hogu. A radio frequency (RF) transmitter then wirelessly sends the information to the connected computer.

4.2 Electronic Helmet

Similar to electronic hogus, head shots are automatically scored with electronic helmets (Figure 3).

4.3 Electronic Foot Sensor Socks

Athletes competing in sparring competitions that use TrueScore's electronic point scoring system are required to wear e-socks (Figure 4). In these electronic feet gear, magnets are embedded in parts such as the bridge and heel of the foot. Whether or not an impact is from a kick is then determined by whether or not the hogu detects a magnet in proximity of the impact location.

4.4 Wireless Joysticks

During sparring matches, judges with joysticks (Figure 5) assist in the electronic scoring process. For example, punches are manually scored (there are no magnets in the gloves, and punches are worth less than

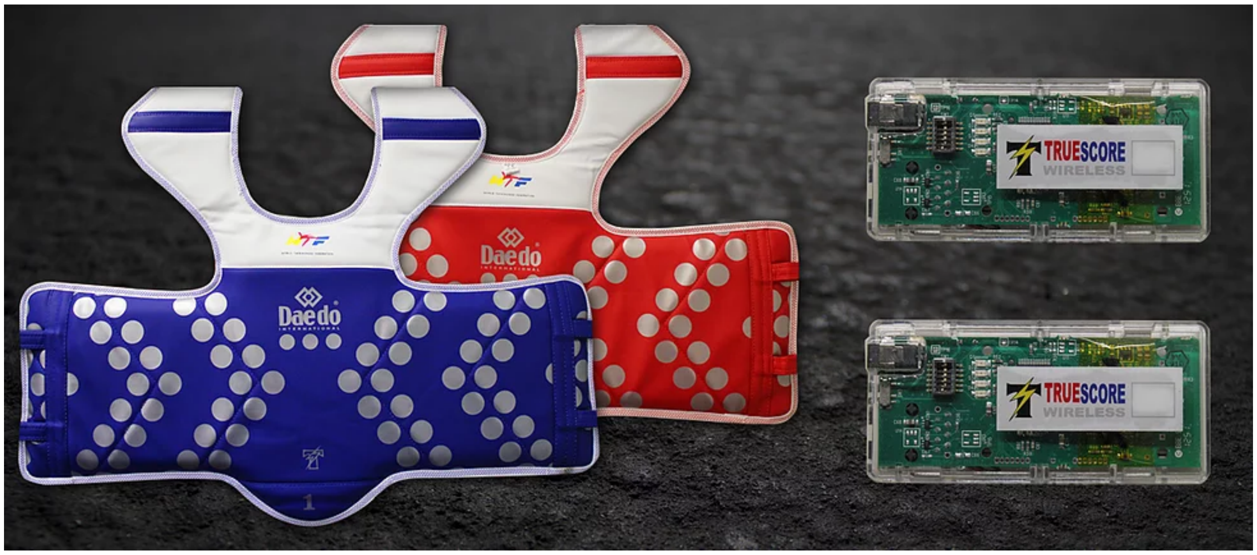


Figure 2: The electronic hogu (e-hogu).

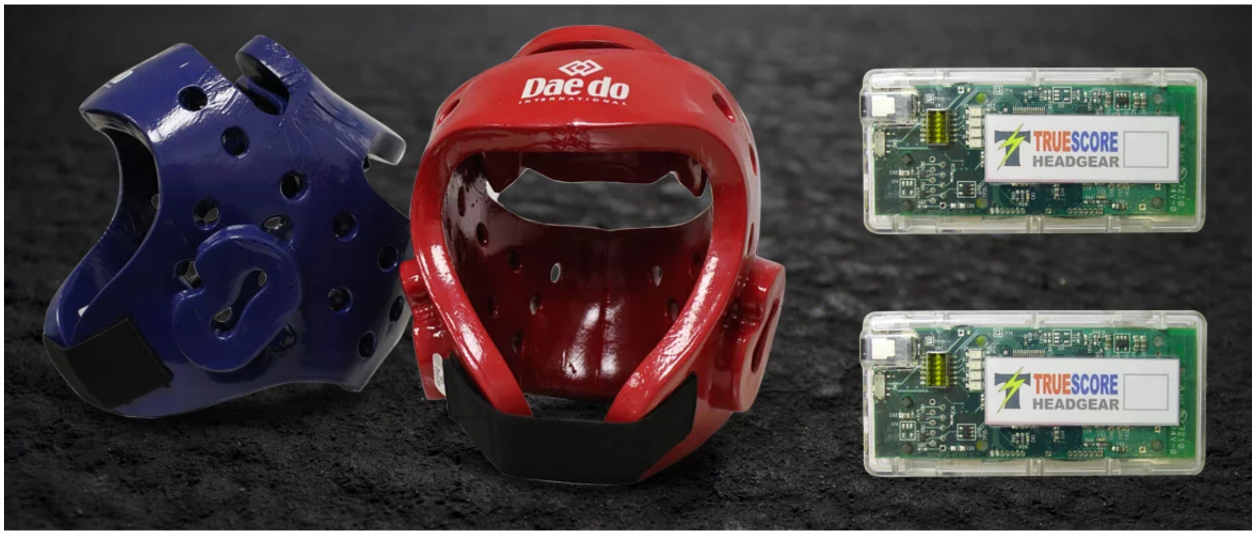


Figure 3: The electronic helmet (e-helmet).

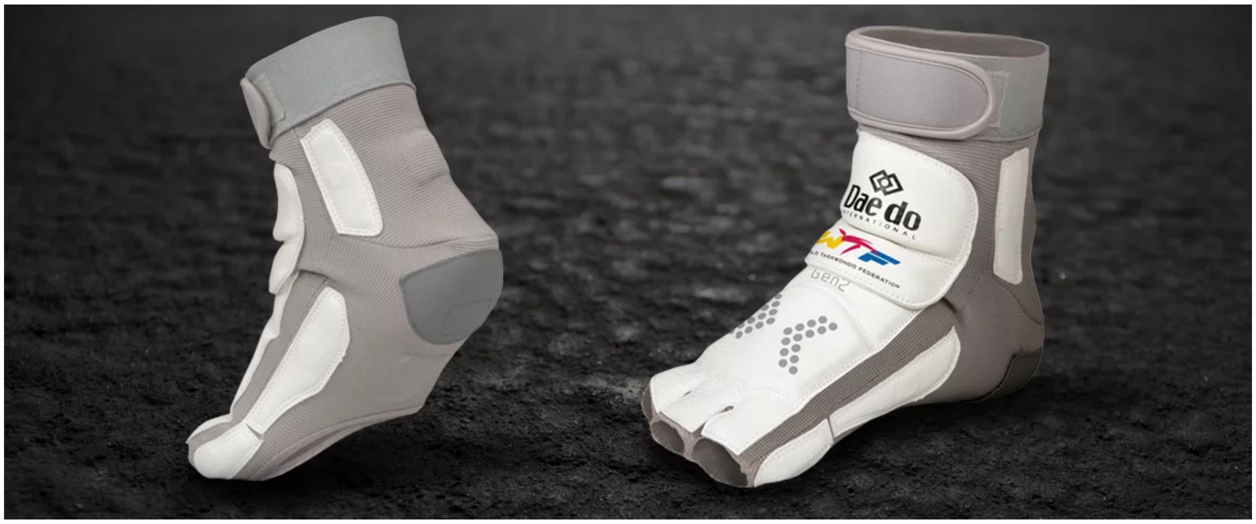


Figure 4: The electronic foot sensor socks (e-socks).

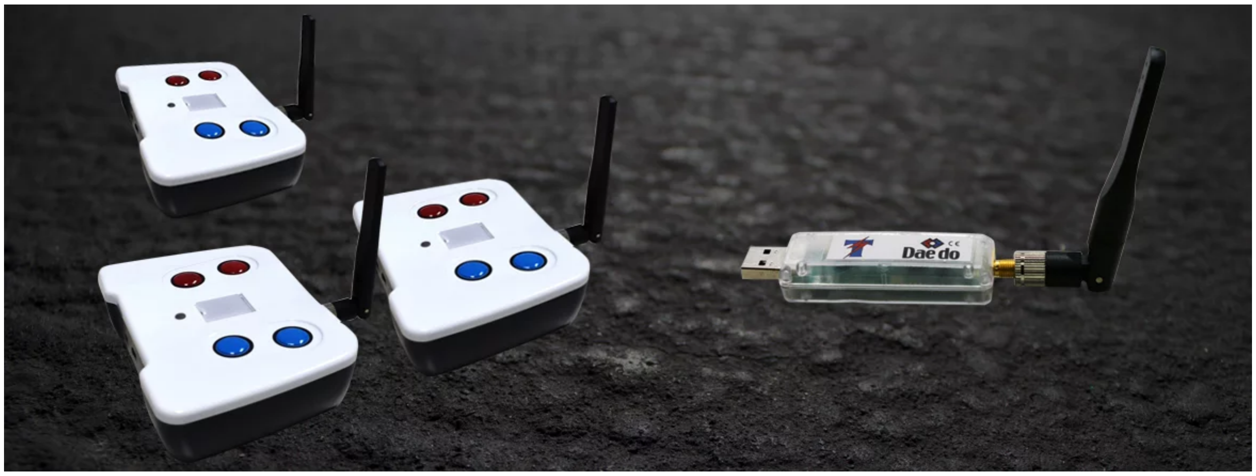


Figure 5: The wireless joysticks and wireless receiver USB dongle.

kicks), and judges can also add technical points for difficult kicks (such as a spinning hook kick compared to the roundhouse kick). These joysticks have buttons that score different point values for both fighters.

4.5 Wireless Receiver USB Dongle

A wireless receiver (Figure 5) is connected to the scoring computer and receives signals from the hogus and the joysticks. When it receives a signal, it decodes the signal and sends it to the TrueScore software.

4.6 TrueScore Software

The software on the computer (Figure 6) calculates the amount of force with which the hogu has been hit. If the force exceeds a pre-specified threshold, the software awards points to the fighter who made the hit.

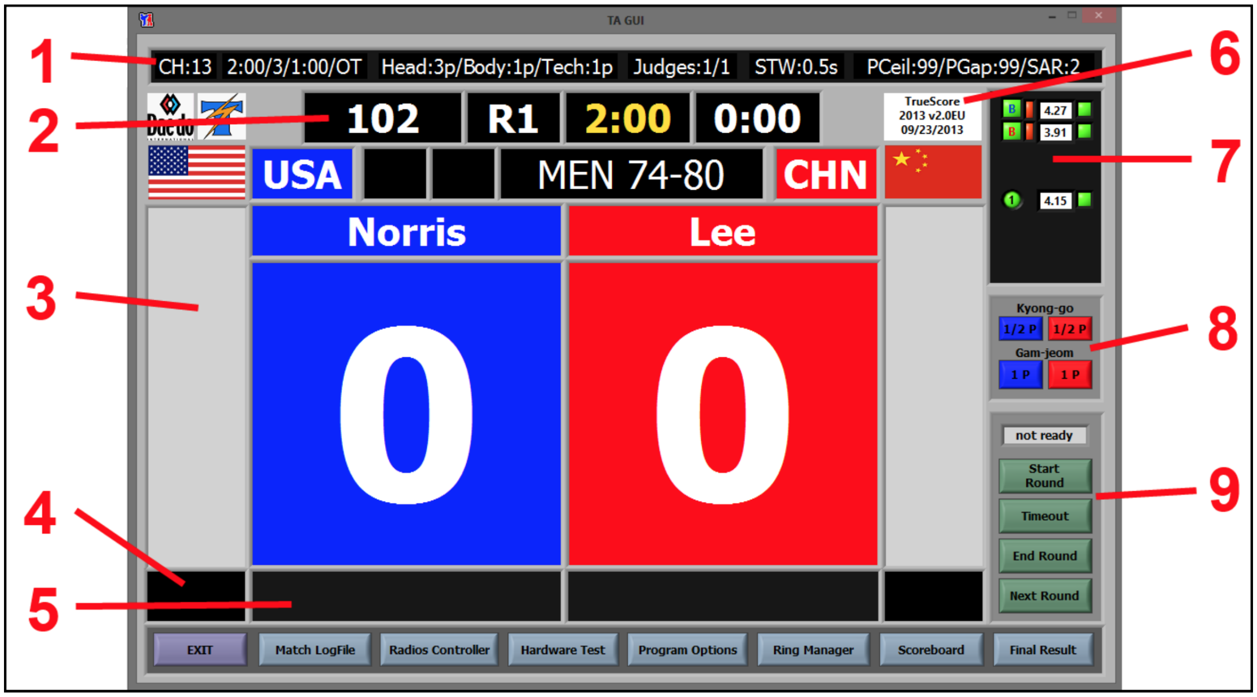


Figure 6: The TrueScore software's control panel.

5 Approaches

5.1 Integrity: Enhance Socks with Magnets

Our most basic approach was to replicate the Sockgate attack by hitting the hogu with more than one e-sock worth of magnets. As expected, this made it easier to score a kick, but did not increase the score of a given kick. This is because the magnetometer(s) in the hogu are for detecting proximity, while the piezoelectric force sensors in the hogu determine the kick score. These additional magnets only affect the proximity detection.

5.2 Confidentiality: Dongle-Computer Communications

We listened to the messages being sent between the dongle and the Truescore software by using Device Monitor Studio. After running experiments and recording the data captured, we discovered that the dongle sends a constant stream of 13-byte messages to the software, which signals that it is connected and running; these messages also serves as counters that keep track of the relative timings of events. Each event triggers an additional 14-byte message, whether it's a hit on a hogu, or a button press on a controller.

Below is our decoded results for messages triggered by an event:

- The first two bytes indicate the source of the message, whether it's coming from the client (the hogus or the judge controllers) or the server (the constant message stream from the dongle).
- The next two bytes describe the force measurement on the hogu, or which button was pressed on the judge controller.
- The next two bytes contain the hogu or controller ID.
- This is followed by two constant bytes that we observe in every message.
- The next byte serves as a counter.

Hogu →	**Source (client**	**Force**	**Hogu ID**	**const**	**counter**	**??**	**const**
Controller →	**vs. server)**	**Button ID**	**Controller ID**				
Example bytes	44 3D	02 C8	4E 7E	04 9A	05	B6	FF 0D FE 01
Example decode	Client	Force: 44	Hogu 'N126'		5th hit		

Figure 7: Decoded results of the communication between the dongle and and the software.

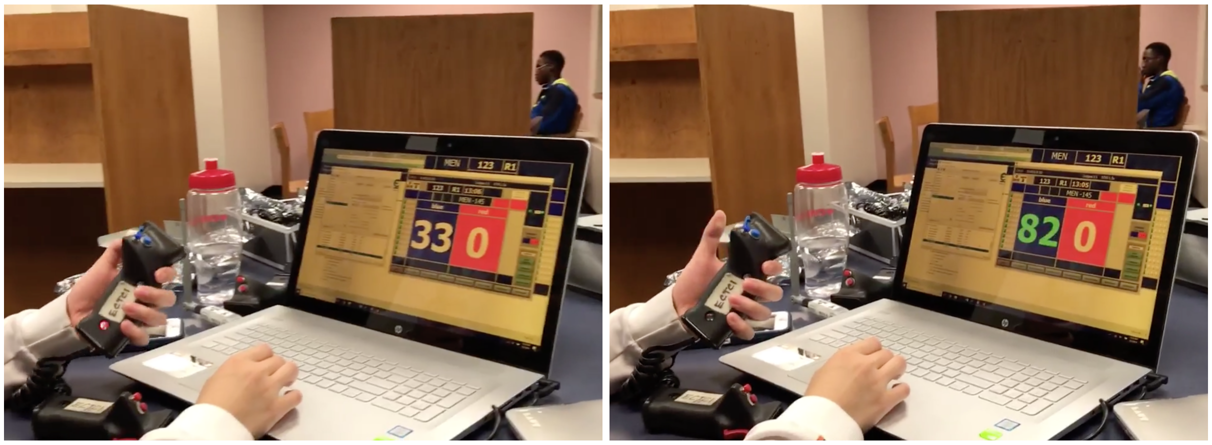


Figure 8: *Left:* The scoreboard before the button on the judge controller was pressed. *Right:* The scoreboard after the button on the judge controller was pressed.

- After this, we have a byte that we're unsure of its purpose. One of our guesses is that it is an encrypted counter or a MAC of the message to prevent replay attacks.
- Finally, we have four bytes that mark the ending of a message, which are constant except for rare occurrences. We're not sure why this is the case.

A tabular representation of this decoding scheme and an example message from a kick on a hogu can be found in Figure 7. Decoded, we see that the kick registered a force of 44 units on hogu N126, and that it was the 5th hit scored on that hogu.

We hope to use our results from observing the data stream between the dongle and the software in a few of our other approaches, particularly our attempt to spoof the dongle and our RF attacks.

5.3 Integrity: Memory Editing & Software

Using the open-source Cheat Engine [6] project, we observed the way in which TrueScore uses the memory allocated to it. We found that an intruder could easily determine which memory addresses were responsible for scoring purposes. Each memory address is represented by eight hexadecimal characters, and using Cheat Engine we were able to identify the specific memory address that stores the blue competitor's score. This makes it easy for an intruder to pick the target memory block, which is the vulnerability that we used.

Using the `ReadProcessMemory` and `WriteProcessMemory` functions in the standard Windows library, we were able to manipulate the value of various memory locations by running a C++ script in the background. Here are some that we implemented:

1. By editing the value of the correct memory address, we can set the blue competitor's score to arbitrary numbers, including negative numbers and large numbers (Figure 8).
2. We were able to force the blue competitor's score to remain at zero regardless of what the competitor did. The images below demonstrate this exploit in action (Figure 9).

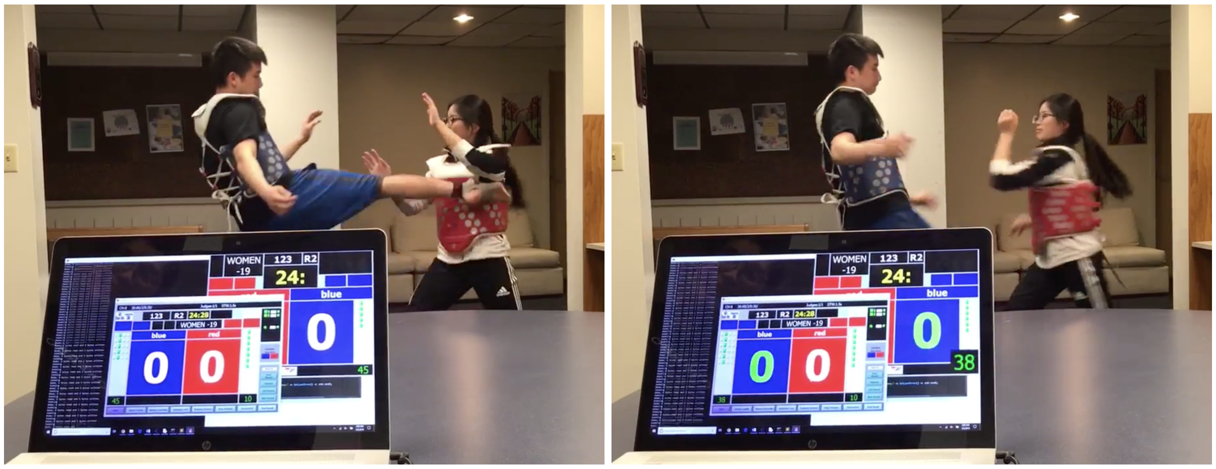


Figure 9: *Left:* Peter, the blue competitor, kicking Elizabeth. *Right:* Peter's kick is registered as having a force of 38 units, and the blue competitor's score lights up, but his score remains at 0.

These approaches can potentially run undetected in the background of a computer compromised by malware. Our recommendation to prevent memory editing attacks is to run the TrueScore software in a virtual machine, which would add an additional barrier to such memory editing methods.

Another weakness of our approach is that the memory address changes between each execution of the TrueScore software. In order for our method to work, the operator must intentionally identify the relevant memory addresses for their particular instance of TrueScore before editing their values. However, in our testing, we noticed that the last four characters of the memory addresses did not change between executions. For example, the memory address storing the blue competitor's score seemed to have the form `0x----9308`, even after restarting TrueScore. This value may be specific to the machine running it, however, so its utility to an adversary without access to the actual computer running TrueScore may be limited.

5.4 Integrity: Spoofing USB Dongle

After decoding the dongle-computer communications in section 5.2, we attempted to create a mock dongle that would be recognized by the TrueScore software as a real dongle. If we succeeded in this, we could send messages to the software that appear to come from judge boxes or hogus. We could press a button on the mock device, for example, that sends a message reporting a kick.

We ultimately were not able to get this device to be recognized by the TrueScore software by the end of the project. Here, we document the approach we took and what we learned about the dongle in our research.

The TrueScore USB dongle communicates via the USB protocol [7] with the computer. The device is defined by its device descriptors, which can be accessed with libusb [8] through PyUSB [9].

```
DEVICE ID 0403:6001 on Bus 020 Address 029 =================
 bLength                 :   0x12 (18 bytes)
 bDescriptorType         :    0x1 Device
 bcdUSB                  :  0x110 USB 1.1
 bDeviceClass            :    0x0 Specified at interface
 bDeviceSubClass         :    0x0
 bDeviceProtocol         :    0x0
 bMaxPacketSize0         :    0x8 (8 bytes)
 idVendor                : 0x0403
 idProduct               : 0x6001
 bcdDevice               :  0x400 Device 4.0
 iManufacturer           :    0x1 FTDI
```

```
iProduct                :    0x2 USB <-> Serial
iSerialNumber           :    0x0
bNumConfigurations      :    0x1
 CONFIGURATION 1: 90 mA ===================================
  bLength               :    0x9 (9 bytes)
  bDescriptorType       :    0x2 Configuration
  wTotalLength          :   0x20 (32 bytes)
  bNumInterfaces        :    0x1
  bConfigurationValue   :    0x1
  iConfiguration        :    0x0
  bmAttributes          :   0x80 Bus Powered
  bMaxPower             :   0x2d (90 mA)
   INTERFACE 0: Vendor Specific ============================
    bLength             :    0x9 (9 bytes)
    bDescriptorType     :    0x4 Interface
    bInterfaceNumber    :    0x0
    bAlternateSetting   :    0x0
    bNumEndpoints       :    0x2
    bInterfaceClass     :   0xff Vendor Specific
    bInterfaceSubClass  :   0xff
    bInterfaceProtocol  :   0xff
    iInterface          :    0x2 USB <-> Serial
     ENDPOINT 0x81: Bulk IN ================================
      bLength           :    0x7 (7 bytes)
      bDescriptorType   :    0x5 Endpoint
      bEndpointAddress  :   0x81 IN
      bmAttributes      :    0x2 Bulk
      wMaxPacketSize    :   0x40 (64 bytes)
      bInterval         :    0x0
     ENDPOINT 0x2: Bulk OUT ================================
      bLength           :    0x7 (7 bytes)
      bDescriptorType   :    0x5 Endpoint
      bEndpointAddress  :    0x2 OUT
      bmAttributes      :    0x2 Bulk
      wMaxPacketSize    :   0x40 (64 bytes)
      bInterval         :    0x0
```

Many of the fields for this device differ from a common prototyping board like the Teensy [10]. We modified the TeensyDuino core library [11] to adapt the Teensy's USB descriptors and protocol to match the TrueScore dongle. Here are a few notable changes.

- **bDeviceClass**: The device class is not specified at the device level (e.g. Teensy is `0x2`), but rather at the interface level.
- **idVendor**, **idProduct**: We changed the Teensy's Vendor and Product IDs to match.
- **iManufacturer**: The TrueScore dongle is based on a TinyOS [12] board with an FTDI USB to serial chip, so this field tells the computer to use FTDI drivers to establish communication with it. The Teensy does not use an FTDI chip, which may have been why we had such a hard time getting the Teensy to be recognized by the laptop's FTDI drivers.
- **iSerialNumber**: Changing this field to `0x0` made it show up on the Windows Device Manager list.
- **bmAttributes**, **bMaxPower**: These fields differed between the Teensy and the TrueScore dongle, but we didn't experiment with changing them out of fear of breaking the devices. Perhaps these are used to recognize the USB device.

- **bInterfaceClass**, **bInterfaceProtocol**: These two fields are likely the most important for spoofing the dongle. They illustrate that TrueScore is using a vendor-specific communication protocol different than the TeensyDuino's more standard CDC Data protocol [13].

Other details we modified to make the Teensy descriptor more similar to the TrueScore dongle were to change the order of the endpoints (Bulk IN before Bulk OUT) and remove a second CDC Interrupt interface. We were not able to change the **bEndpointAddress** locations for each of the endpoints and still successfully send and receive data. We believe that this is because these locations are defined in a lower level of the architecture, as discussed in a forum post by the creator of the Teensy [14]:

There are a few essential parts to understand about the USB module... It needs a specific memory layout. Since it doesn't have any dedicated user-accessible memory, it requires that the user specify where things should be. There are specific valid locations for its Buffer Descriptor Table (more on that later) and the endpoint buffers. (Paul Stoffregen, 2012)

We address future work in this area in Section 6.

5.5 Availability: RF Denial of Service

The TrueScore software allows the user to choose a frequency channel from 1 to 15. When we scanned through the 2.4 GHz frequency band, we found that the frequency channels from the software end up aligning very closely to ones for WiFi. Because of that, we know which frequencies the transmitter and receiver operate on and we can enact a RF denial of service attack simply by flooding the frequency channel either with a stronger signal or massive amounts of noise.

To actually jam the RF communications, we would just use a signal generator and continuously produce random noise at that specific frequency. However, we decided against this though because jamming radio frequencies is illegal in the United States. Additionally, since the transmitter and receiver operate on the same frequency range as WiFi, jamming the frequency may cause problems for others in the vicinity. Another problem with sharing the same frequency range as WiFi is that if we choose a channel that has a lot of traffic a significant amount of the signals being transmitted are not picked up by the receiver. This is easily remedied by choosing a channel with less traffic which can easily be found using a WiFi analyzer.

6 Future Work

6.1 Confidentiality: RF Snooping and Replay Attack

The wireless receiver uses Texas Instrument's CC2420 chip to communicate in the 2.4 GHz unlicensed ISM band. This chip has a frequency range of 2400 MHz to 2483.5 MHz [15]. In order to try and listen in on the communication between the transmitter and receiver, we first need to determine what frequency the two devices were using to communicate. To do this we use a HackRF One, which is a software defined radio, to sweep through the frequencies from 2400 MHz to 2483.5 MHz.

We observed the frequency spectrum before and after communicating with the wireless dongle and saw a peak around 2460 MHz that was not there before we started sending signals (Figure 10 & 11). The peaks also happened to occur whenever we pressed a button on the judge controller. Although there was not a peak every single time we pressed a button, we are still pretty confident that we found the correct frequency and that the HackRF just missed the signal.

Because we now know what frequency the transmitter and receiver were communicating on, we tried a simple replay attack by recording the signals and replaying it exactly to the receiver. Our replay attacks have not been successful so far, so our next steps would be to actually demodulate the signal to actually get the bits being sent. This would allow us to actually see if there some sort of mechanism like a counter preventing us from simply replaying the message.

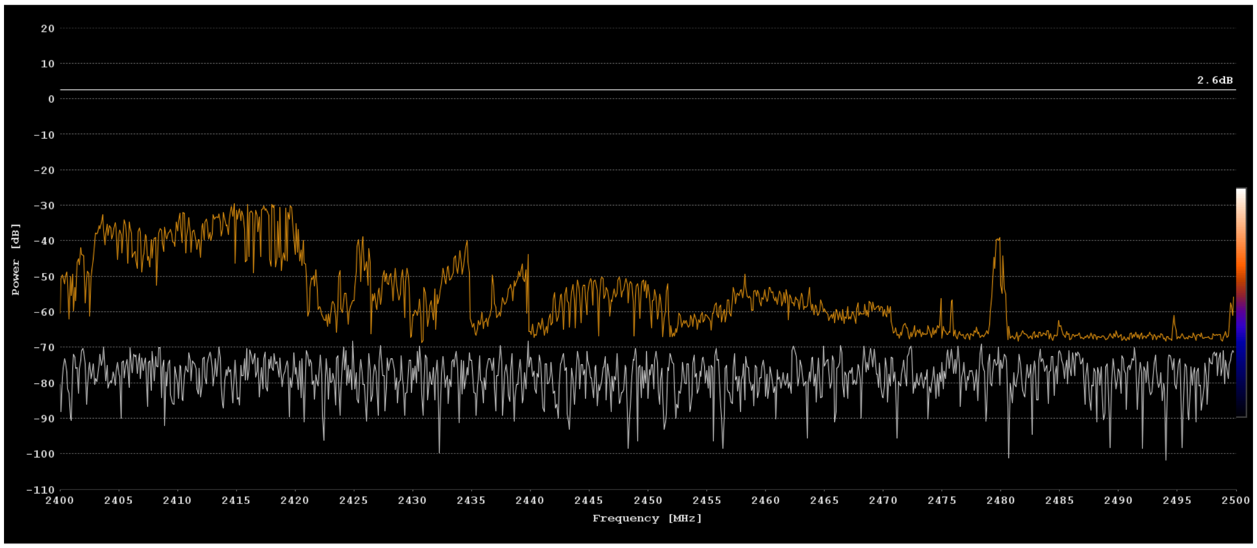


Figure 10: An example of the frequency spectrum before using the TrueScore software.

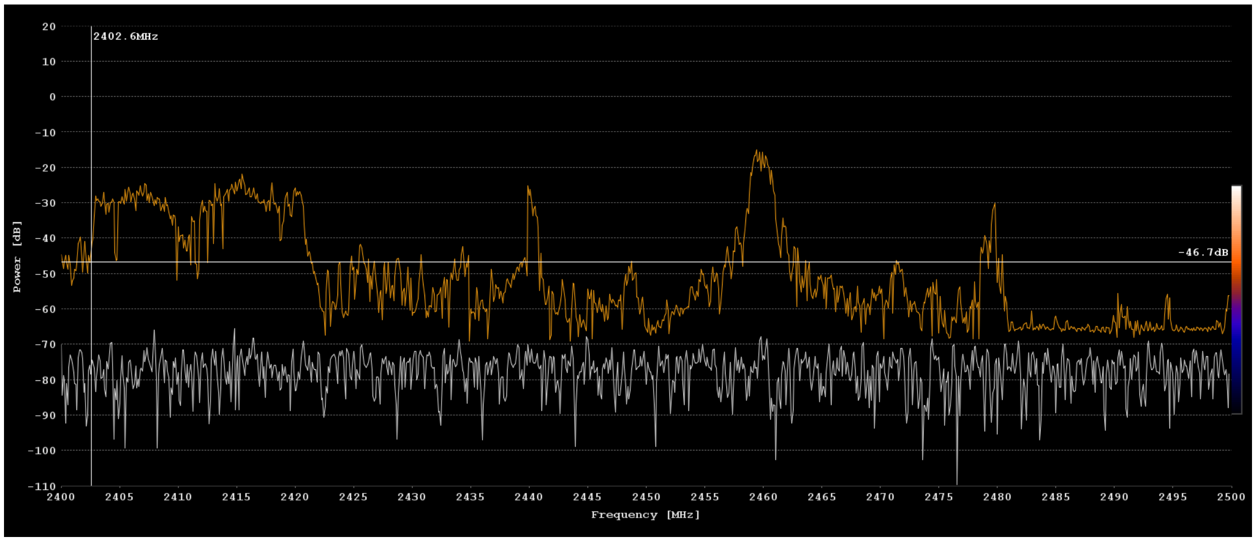
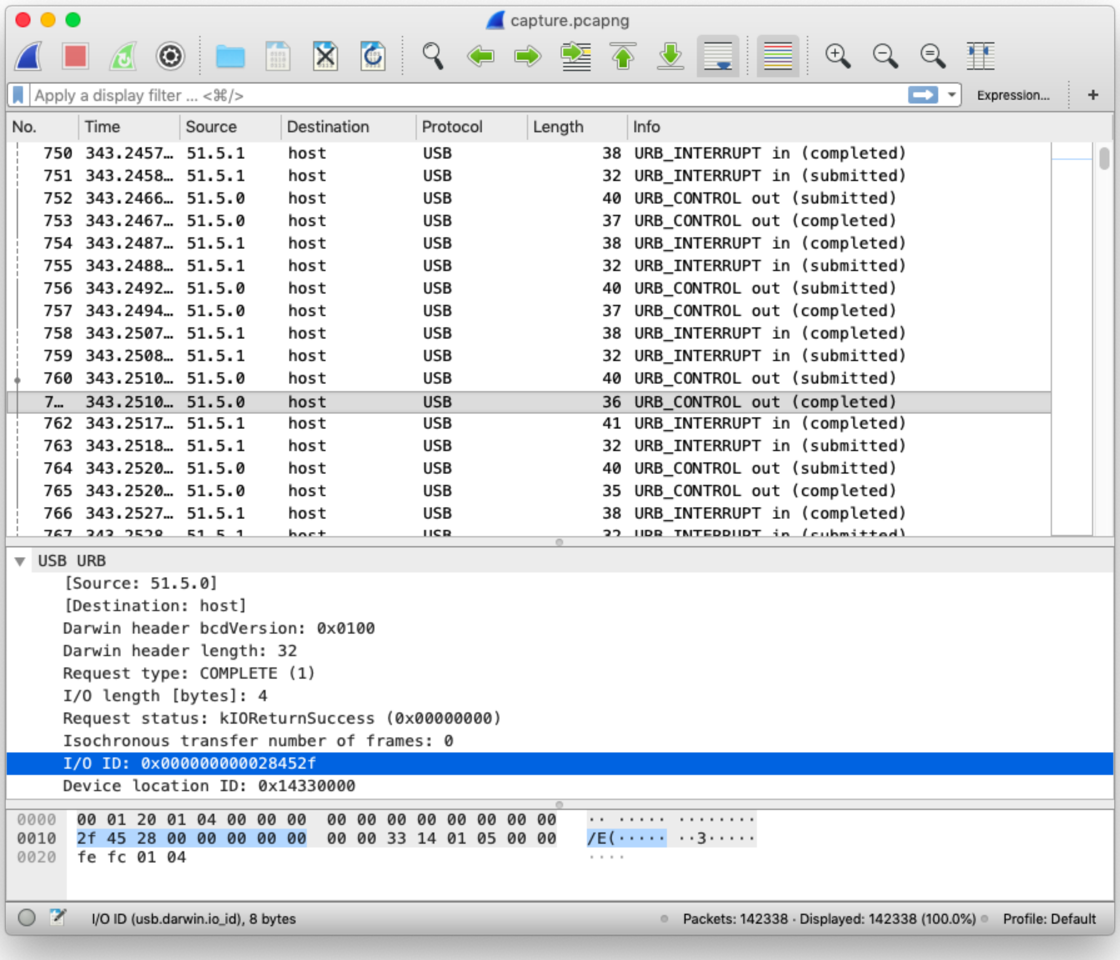


Figure 11: An example of the frequency spectrum after using the TrueScore software. The main difference is the peak at around 2460 MHz.



```
No.   Time        Source   Destination  Protocol  Length  Info
 750  343.2457…   51.5.1   host         USB           38  URB_INTERRUPT in (completed)
 751  343.2458…   51.5.1   host         USB           32  URB_INTERRUPT in (submitted)
 752  343.2466…   51.5.0   host         USB           40  URB_CONTROL out (submitted)
 753  343.2467…   51.5.0   host         USB           37  URB_CONTROL out (completed)
 754  343.2487…   51.5.1   host         USB           38  URB_INTERRUPT in (completed)
 755  343.2488…   51.5.1   host         USB           32  URB_INTERRUPT in (submitted)
 756  343.2492…   51.5.0   host         USB           40  URB_CONTROL out (submitted)
 757  343.2494…   51.5.0   host         USB           37  URB_CONTROL out (completed)
 758  343.2507…   51.5.1   host         USB           38  URB_INTERRUPT in (completed)
 759  343.2508…   51.5.1   host         USB           32  URB_INTERRUPT in (submitted)
 760  343.2510…   51.5.0   host         USB           40  URB_CONTROL out (submitted)
  7…  343.2510…   51.5.0   host         USB           36  URB_CONTROL out (completed)
 762  343.2517…   51.5.1   host         USB           41  URB_INTERRUPT in (completed)
 763  343.2518…   51.5.1   host         USB           32  URB_INTERRUPT in (submitted)
 764  343.2520…   51.5.0   host         USB           40  URB_CONTROL out (submitted)
 765  343.2520…   51.5.0   host         USB           35  URB_CONTROL out (completed)
 766  343.2527…   51.5.1   host         USB           38  URB_INTERRUPT in (completed)
 767  343.2528…   51.5.1   host         USB           32  URB_INTERRUPT in (submitted)

▼ USB URB
    [Source: 51.5.0]
    [Destination: host]
    Darwin header bcdVersion: 0x0100
    Darwin header length: 32
    Request type: COMPLETE (1)
    I/O length [bytes]: 4
    Request status: kIOReturnSuccess (0x00000000)
    Isochronous transfer number of frames: 0
    I/O ID: 0x000000000028452f
    Device location ID: 0x14330000

0000  00 01 20 01 04 00 00 00  00 00 00 00 00 00 00 00   ·· ······ ········
0010  2f 45 28 00 00 00 00 00  00 00 33 14 01 05 00 00   /E(····· ··3·····
0020  fe fc 01 04                                        ····

I/O ID (usb.darwin.io_id), 8 bytes    Packets: 142338 · Displayed: 142338 (100.0%)    Profile: Default
```

Figure 12: An example of a Wireshark trace from the TrueScore dongle.

6.2 Integrity: Complete Spoofed USB Dongle

Given that the TrueScore dongle uses a vendor-specific interface and protocol, we would likely have needed to reverse-engineer this entire protocol to succeed in mocking the device, similar to how the Microsoft Kinect was reverse-engineered [16]. This process involves meticulously observing the messages passed between the laptop and device in a variety of states through a program like Wireshark [17] (see Figure 12) and re-creating them.

If we were to do this project again, we would start with hardware as similar to the target device as possible. In particular, it would have been better to use a device with an FTDI USB to Serial chip. We decided to use Teensy/TeensyDuino because we were more familiar developing with it than TinyOS/NesC, but this put more burden on us to accommodate the quirks of the device's firmware and drivers.

The code for this work is available in the private repositories `https://github.com/cmnord/hogu-security` and `https://github.com/cmnord/teensy-cores` upon request.

6.3 Gen 1 vs Gen 2 Limitations

TrueScore has two versions of their Taekwondo scoring system. Gen 1 is the first version of the system, while Gen 2 is the new and improved system replacing Gen 1. We had permission to work with the Gen 1 version of TrueScore since it is less of a risk to TrueScore's main product. Because we only worked on the Gen 1 system, some of our approaches might not carry over to the new Gen 2 system easily.

The approach that will probably still work the same on both Gen 1 and Gen 2 would be adding more magnets to the socks. Adding more magnets will probably still help because it is a key part of how the hogus actually detect a hit. Unless the system completely moves away from using magnets, the use of additional magnets will probably always help.

All of the other approaches are all still plausible, but they might require a bit more effort on the Gen 2 system. For example, the communication protocol between the wireless dongle and the computer may differ in Gen 2. This could make decoding these messages more difficult or impossible. Additionally, it might be harder to adjust the memory of the Gen 2 software due to counter measures such as using a virtual machine or including integrity checks. Although we did not successfully create a fully spoofed USB dongle and snoop or attack the RF communication, these both remain possibilities in Gen 1 and Gen 2.

7 Conclusion

These days, sports are not just a form of exercise or entertainment. They are a source of national pride, and with this comes the incentive to cheat at the highest levels of competition. For sports with electronic scoring systems, such as Taekwondo and fencing, the integrity of these systems have to be examined to ensure an accurate and fair competition.

The TrueScore system has several vulnerabilities that an adversary could use to manipulate the system. In our testing, we considered potential attacks using the magnets in the e-socks (Section 5.1), the dongle-computer datastream (Section 5.2), editing the memory used by the TrueScore software (Section 5.3), spoofing the USB dongle (Section 5.4), and the RF communication between the hogus and controllers (Section 5.5). In Section 6, we describe some methods we are currently investigating, such as replay attacks over RF.

Many companies prioritize new features over considering the security of existing ones, but we believe that they should go hand in hand. As we have seen in election security research, replacing humans in a system with hardware and software does *not* necessarily improve the security of the system, it just changes what and who is being trusted. Some martial arts anthropologists have even said that "in today's age of computer games, the taekwondo community [has become] infected with a blind faith in technology. The belief in the superiority of computer-controlled systems over experienced, sound human judgment brought an array of new, non-anticipated problems and side-effects to the sport." [18].

From this research, we've concluded that the most reliable way to improve security is to carefully analyze and improve the *processes* and *procedures* in a system.

8 Acknowledgments

We would like to first thank **TrueScore** for giving us permission to work on this project. We are grateful for the opportunity to further study the system that enables the sport we enjoy so much. We would further like to thank the **6.857 staff** for teaching us not just about cryptography, but about how to evaluate the security of a complex system. Last, we would like to thank **Master Dan Chuang**, **CW Taekwondo at Boston**, and the **Eastern Collegiate Taekwondo Conference** for lending us the equipment needed for us to carry out this project.

References

[1] Joseph Durso. Soviet fencer disqualified for cheating. *The New York Times*, July 1976.

[2] Great british olympians. web.archive.org/web/20110629071717/http://www.times-olympics.co.uk/historyheroes/stgbo04.html, June 2011.

[3] Simon Burnton. 50 stunning olympic moments no18: Boris onischenko cheats, gb win gold. *The Guardian*, 2012.

[4] Taiwan taekwondo athlete in asian games sock sensor row. *BBC News Asia-Pacific*, November 2010.

[5] Taiwan taekwondo athlete yang suspended for three months (update). *Focus Taiwan News Channel*, December 2010.

[6] Cheat engine. https://www.cheatengine.org/.

[7] Craig Peacock. Usb in a nutshell. *Beyond Logic: https://www.beyondlogic.org/usbnutshell/usb1.shtml*, 2002.

[8] J Erdfelt and D Drake. Libusb homepage. *http://www.libusb.org*.

[9] Pyusb. https://github.com/pyusb/pyusb.

[10] USB Teensy. Development board. *https://www.pjrc.com/teensy*, 2012.

[11] Paul Stoffregen. Teensy core libraries for arduino. https://github.com/PaulStoffregen/cores.

[12] Philip Levis, Sam Madden, Joseph Polastre, Robert Szewczyk, Kamin Whitehouse, Alec Woo, David Gay, Jason Hill, Matt Welsh, Eric Brewer, et al. Tinyos: An operating system for sensor networks. In *Ambient intelligence*, pages 115–148. Springer, 2005.

[13] Usb defined class codes. https://www.usb.org/defined-class-codes#anchor_BaseClassFFh.

[14] Custom usb device based on teensy 3.2. https://forum.pjrc.com/threads/49045?p=164512&viewfull=1#post164512.

[15] *2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver*.

[16] Lady Ada. Hacking the kinect. *https://learn.adafruit.com/hacking-the-kinect/overview*, July 2012.

[17] Wireshark. https://www.wireshark.org/.

[18] Udo Moenig. Rule and equipment modification issues in world taekwondo federation (wtf) competition. *Ido Movement for Culture. Journal of Martial Arts Anthropology*, 15(4):3–12, 2015.