6-2-2011

IEEE 802.15.4 Wireless Security: Self-Assessment Frameworks

Ryan Speers
Dartmouth College

IEEE 802.15.4 Wireless Security: Self-Assessment Frameworks

An Undergraduate Thesis

Dartmouth Computer Science Technical Report TR2011-687

Ryan Speers
Advisor: Sergey Bratus

June 2, 2011

Abstract

This thesis analyzes the security of networks built upon the IEEE 802.15.4 standard, specifically in regard to the ability of an attacker to manipulate such networks under real-world conditions. The author presents a set of tools, both hardware and software, that advance the state-of-the-art in reconnaissance and site surveying, intelligent packet generation, and launching of attacks. Specifically, tools provide increased hardware support for the KillerBee toolkit, a Scapy layer for forming 802.15.4 packets, reflexive jamming of packets, and other research enablers. This work aims to advance the ability of security auditors to understand the threats to IEEE 802.15.4 networks by providing auditors usable and low-cost tools to carry out vulnerability assessments.

Contents

Chapter 1

Introduction

Wireless isn't intrinsically bad, but it has lots of room for failure. As the types of devices which are networked expands from computers and phones to thermometers and power meters, wireless is an attractive choice for implementation. Wireless networking technologies are often faster to implement, especially where there is no network line available to the device's location. The "smart" power grid is receiving significant stimulus funding, and the short-lived nature of these funds may inadvertently lead to rapid rather than secure implementation [27]. In a world where proprietary encryption is sometimes being used as a selling point [27], it is vital for security researchers to be able to self-assess the security of these technologies.

The authors of "Wi-Foo: The Secrets of Wireless Hacking" emphasize "the value of viewing your network through the cracker's eyes" [39, 20] as it relates to 802.11 Wi-Fi networks. This paper will bring this ability to the 802.15.4 realm through research and usable tools that enable the discovery of devices, fingerprinting devices, intelligent crafting of packets, and constructing possible attacks.

The author shall address security issues in the 802.15.4 physical (PHY) radio-frequency (RF) and media-access control (MAC) layers. Understanding and being able to manipulate or defend these layers is critical to being able to exploit or defend the network (NWK) layer protocols built atop it such as ZigBee and 6lowpan. In choosing this approach, the author agrees with the statement that "Layer 1 security...should always be investigated first on wireless nets" [39, 20].

Chapter 2

Executing a Site Survey

2.1 Prior Work

Wireless security experts have emphasized the importance of determining where attackers can position physically, detecting neighboring networks, and baselining RF interference (to detect abnormal jamming levels in the future) [39, 20]. Techniques for network discovery have evolved from passive listening on networks to active techniques where a tool sends out probe requests and listens for a response. The active tools allow detection of some devices which are not beaconing (by requesting a beacon to be sent), whereas the passive techniques leave no trace and can detect clients, not just beaconing devices. Numerous tools using these techniques have been produced that allow network discovery to be done on 802.11 WiFi networks, including kismet, wellenreiter, airfraf, gtkskan, airfart, mognet, and wifiscanner.

However, there is a notable absence of such tools for auditing 802.15.4 and ZigBee networks. Joshua Wright's initial KillerBee project release (this version will be referred to as KillerBee 1.0 [43]) has produced tools that perform scanning for networks (`zbstumbler` and `zbfind`) as well as a tool for packet capture (`zbdump`). These tools supported only the Atmel RZUSBSTICK as a capture interface. The KillerBee 1.0 code provided the author a strong basis to build on. Kevin Finisterre has also written some Ruby scripts and C programs for trying to capture from a number of devices, specifically trying to deal with some of the USB bus issues encountered when running through USB hubs. The author reviewed his code, which is currently not released, but some of which is discussed at [15]. One of the main efforts of Mr. Finisterre's work is to be able to capture on many channels at once, using different network interfaces, which enables capture on frequency hoping devices as well as exploration of ZigBee-saturated areas. The author's related work is discussed in 2.3.2.

2.2 Capture Devices

2.2.1 ZigBee Chip Enclosures

The author designed and produced physical tools to aid in assessment, including hardened cases for ZigBee radio "dongles." A chip that is too fragile to use outside a lab setting is of little use for real-world security auditing. Enclosures were needed to make auditing tools requiring radio interfaces into truly field-usable security tools.

The removable casing for the Atmel RZUSBSTICK (see figure 2.1) was created to (1) protect the hardware from handling and some enviromental hazards, (2) give the appearance of a commonly accepted device, such as a USB memory stick, (3) be temporary and allow easy access to the chip for updating firmware, and (4) minimize cost. Because the RZUSBSTICK has an internal PCB trace antenna, it was important to ensure that the casing did not decrease the radio strength by an unacceptable amount.

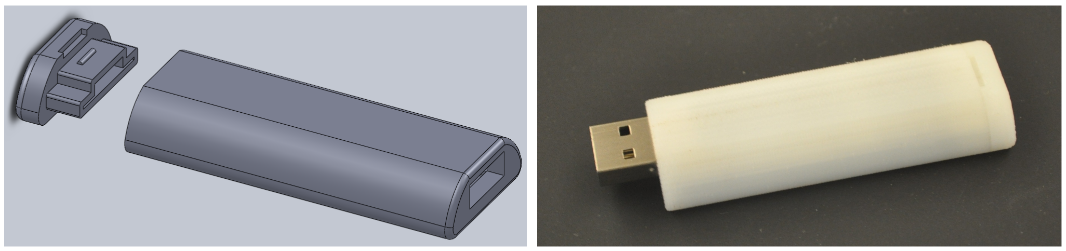
Figure 2.1: Author's Two-Part Casing for an Atmel RZUSBSTICK

The final design consists of a casing and a cap prototyped on a 3D printer. The case has a hole to allow the USB plug to protrude, and the case is contoured to reduce the amount of plastic required. Slight grooves run in the bottom of the case to allow the RZUSBSTICK to easily slide in and out while accomodating protrusions on the bottom of the device. Air space is left above the device to prevent overheating of components. Tests using a DS18B20 digital thermometer probe showed only a 1 °F rise in temperature over 20 minutes while the device was inside the casing transmitting, channel hopping, and receiving actively. This is comparable to the 1 °F rise seen when the device was in the open air, even when it was not actively transmitting or receiving. The cap slides into the end after the device is inserted and clips into shallow recesses in the case (see Figure 2.2). The cap has grooves for easy removal with a fingernail, but otherwise matches the rest of the case.

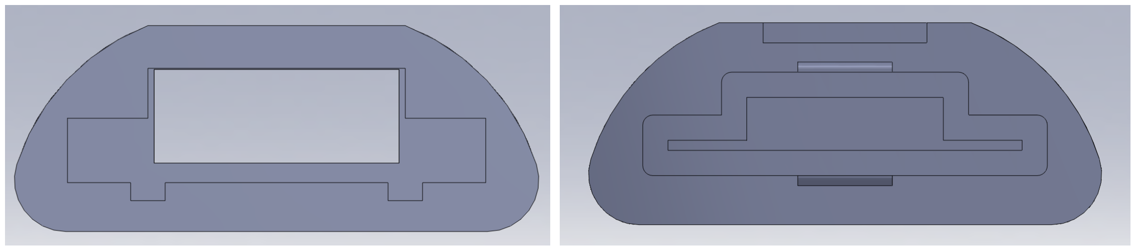
Figure 2.2: Atmel RZUSBSTICK Casing Interlock Detail

The design uses 0.567 in^3 of material and 0.107 in^3 of support material. On a FDM2000 printer at a slice height of 0.01 in, the build time is calculated at almost 3 hours, however it seems to be below 2 hours in our tests. The materials cost for printing one was achieved at \$3.37 per unit.

2.2.2 "zbplant": Standalone Sniffer Platform

The author prototyped a stand-alone data interceptor that could be emplaced by an auditor. A small and wireless capture device could be concealed somewhere that an attacker could not stay (perhaps due to surveillance or other physical security) and would enable the attacker to attain more distant access to the target's radio frequencies. While such devices are available for WiFi networks (one called the "Fon" is a commercial device with new software called "Interceptor" installed on it [23]), the author is aware of only one limited wired analyzer for ZigBee networks. This device, called the Q51 PANalyzer [5] requires Windows configuration and access to a wired Ethernet jack as well as power via a wall plug or power-over-ethernet. Such a device is not a candidate for "covert" use, especially with needing to have a constant ethernet port available as it streams data live to a remote viewer.

In addition to the ability to perform passive collection over long periods of time with less chance of detection, this device could be paired with software that could correlate data from multiple units to triangulate nodes based on RSSI, or simply see which devices are visible in various areas (basic topology based on coverage zones). Especially because devices using technologies like ZigBee often interact with the physical world, geo-location of network nodes is crucial to understanding how the topology of a network relates to a physical installation. Understanding where the coordinator and end-node devices physically sit can lead to identifying weak points in the mesh network as well as providing an idea of what some devices may be monitoring or doing (especially if their traffic is encrypted or non-descriptive). For example, geo-locating a number of end-node devices broadcasting similar traffic near the doorways of a facility may indicate these are door security sensors, as opposed to temperature sensors for an HVAC control system. Also, geo-location can help a user choose the best position to listen to the traffic of a specific node.

Furthermore, the more precisely a node's location is pinpointed, the easier it is for an attacker to perform what is referred to as a node capture attack. Perrig, Stankovic, and Wagner consider "node capture [to be] one of the most vexing problems in sensor network security" [32, 56], and Travis Goodspeed's work on ZigBee chips [19, 20] is one of many examples why this is such a large issue. As can be seen, the additional understanding provided by geo-location of nodes is crucial to fully analyzing a network. Unfortunately, determining radio direction in a ZigBee network is very difficult. ZigBee devices are inherently quiet and use limited transmit power, by design. Furthermore, many networks have nodes in fixed positions. Thus, analysis of packet captures over time may outperform traditional direction finding techniques such as real-time signal strength monitoring.

The author's "zbplant" device (see Figure 2.3) seeks to overcome both the difficulty of obtaining data covertly as well as provide multiple data points to enable direction-finding. This prototype is built on the Arduino platform and consists of components providing ZigBee, Global Positioning System (GPS), power, and storage. The Arudino platform was chosen because of its ubiquitous use in protyping and the number of peripherals (storage/GPS/wifi interfaces) which are tested and have libraries written for the platform.

However, one challenge was capturing 802.15.4 packets in promiscuous mode (i.e. getting all packets including ones not addressed to the device) onto the Arduino platform. Originally, the plan was to do this using an Arduino with an XBee radio chip (which is the standard Arduino solution for ZigBee/802.15.4), but upon further research and discussion with the manufacturer, it became clear that the XBee was unable to do promiscuous capture under any configuration. The design needed direct access to the registers of a 802.15.4 radio chip. Akiba, the author of the FreakLab's blog, recently produced an Arduino clone called "Freakdruino," which integrates an Atmel AT86RF230 radio chip on the SPI bus [7].

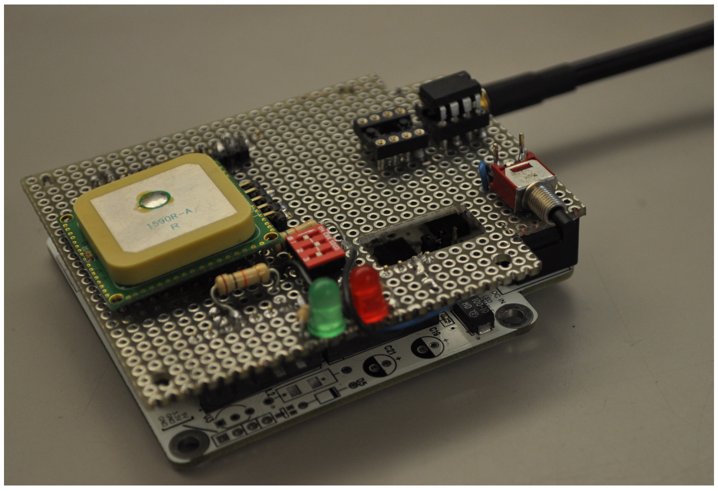
Figure 2.3: zbplant Prototype Device

A GPS chip is added to correlate the location of the unit, in case it is placed somewhere mobile, or so that a user can drop a number of these without having to record precisely where each one is located. In the prototype, the chip is set to send position messages to the Arduino at 1Hz as this helps to reduce battery drain and processing time, and is plenty frequent unless the sniffer is in a fast moving vehicle or aerial platform. The Arduino program ("sketch") only writes the location fix to the log when it changes, instead of coupled with every packet. When the data is being read from the device, the downloader needs to remember the last location update read and use this to mark the location for a specific packet. However, this is easily implemented and reduces the storage requirements on the zbplant device. The initial GPS chip used was a 65 Channel SUP500F 10Hz GPS Receiver with Smart Antenna [2] configured to 1Hz and outputting NEMA-0183 GPRMC and GPGGA sentences which communicate the position, date, time, and altitude data from the GPS chip. These sentences are read over Arduino NewSoftSerial (to avoid sitting on the main hardware serial bus used for the USB FTDI chip) and are fed to the TinyGPS library for parsing [21, 22]. However, due to issues with the reliability of the SUP500F receiver, the author switched the design to use the 66 Channel LS20031 GPS 10Hz Receiver [29]. This receiver's specifications have a little higher power usage, but the reliability also improves greatly. The device is configured the same as the SUP500F described above, with the circuitry for power and input/output levels redesigned to the appropriate levels for the LS20031 (see Figure 2.4).

Data logging is supplied in the prototype by external EEPROM modules. The design uses Microchip 24LC1025 I2C (2-Wire Serial) CMOS Serial EEPROM modules with a 1M (128K x 8) capacity in 8-DIP packages [30]. The address space allows for 4 of these chips to be wired to the I2C bus (see Figure 2.5), and each chip has an internal addressing boundary limitation which divides it into two segments of 512K bits. Data cannot be read/written sequentially across this boundary or across devices. As such, logic is required in the Arduino sketch to implement reliable logging and reading that packs data into all available bytes. This presents complexity especially as page writes are limited by the Arduino Wire (I2C) library to 30 bytes, and the 24LC1025 EEPROM has a 128 byte page size (see Figure 2.6). This effectively limits logging to byte writes if one wishes to use more than the first 30 bytes of the pages. In order to allow the logging to recover from resets and power loss, the zbplant

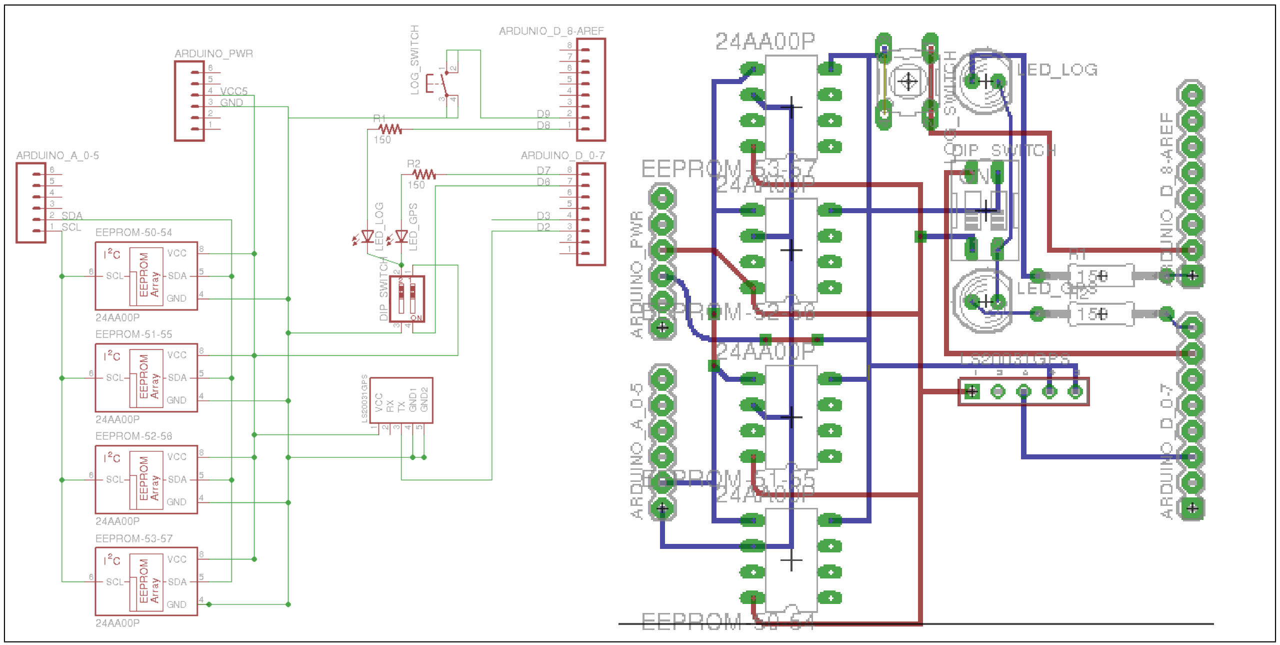

Figure 2.4: zbplant Expansion Board Schematic and Board Layout

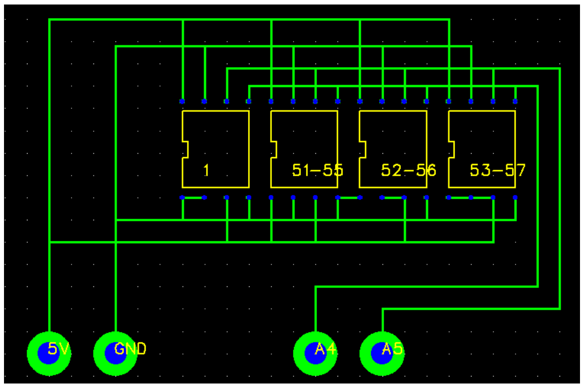

Figure 2.5: Wiring of zbplant EEPROM Modules

stores logging state information in the first page of memory. When reads or writes are done, this information is read, used, and updated. The function `boolean i2c_eeprom_log(char *data, int i)` handles the writing of data with the complexity abstracted by "rolling over" between pages, internal boundaries, and device boundaries as needed.

The author's prototype is not equipped with hardware (such as wifi or cell chips) to report its findings back wirelessly in real time largely due to cost and power consumption issues, although future work adding such optional expansions may be beneficial to some use cases. Currently, this device is supported with a KillerBee driver (see 2.3.1). This driver, `dev_freakduino.py`, implements the `SNIFF` and `SET_CHAN` capabilities. Functionality includes promiscuous capture and the input of captured data to a centralized database schema (see 2.4) or pcap file over a USB link (real time or retreived from the device's onboard storage). The author provides a utility called `zbdumpeeprom.py` to create a pcap file from packets retrieved from the device. A database logging implementation could easily be implemented

```
#define EEPROM_PAGE_LENGTH  128  //number of bytes in one page
#define EEPROM_CONSEC_PAGES 512  //number of consecutive pages in EEPROM
#define EEPROM_DEV_COUNT    2    //number of installed EEPROM devices
#define EEPROM_DEV_0        0x50 //i2c address of EEPROM first device
```

Figure 2.6: zbplant EEPROM Logging Defines

based on this code.

Future work could port the zbplant features to other platforms like the Tmote Sky, although support for features like GPS and Wifi interfacing would be more difficult. However, different use cases may find different implementations with varying feature sets and form factors to be useful.

2.2.3 Tmote Sky/TelosB GoodFET Drivers

As mentioned above, the Tmote Sky (also branded as the TelosB) is another platform that supports 802.15.4 radio. It consists of a Texas Instruments MSP430 family microcontroller and a Chipcon CC2420 radio chip. These were originally designed at UC Berkeley as a platform for wireless sensor network research. Due to this heritage, some have temperature, light, and humidity sensors on board, as well as EEPROM for data logging. The GoodFET project [18], led by Travis Goodspeed, began as an open-source JTAG adapter and offers a slimmed-down framework (as compared to something like TinyOS or Contiki) for interacting with MSP430 family microprocessors. Using the GoodFET CCSPI "application," GoodFET can also interact with the CC2420 radio chip on the device.

The GoodFET firmware compiled with CCSPI support can be loaded from the precompiled version in KillerBee's svn repository, which is highly suggested to avoid dependency requirements. From within `killerbee/firmware`, run `./goodfet.bsl --telosb -e -p gf-telosb-001.hex`. The firmware can also be compiled from the GoodFET repository if updates are needed, as seen in Figure 2.7. This requires that the GoodFET SVN project and the msp430-gcc compiler are installed.

From within `goodfet/trunk/firmware`, run:

```
export platform=telosb
export mcu=msp430x1611
export config='monitor ccspi spi'
make clean install
../client/goodfet.bsl --speed=38400 -e -p goodfet.hex
```

Figure 2.7: Compiling TelosB Firmware from GoodFET Repository

Mr. Goodspeed implemented basic frameworks in the `goodfet.ccspi` client for sniffing and injection, and the author has contributed to testing and improving these capabilities. The author has ported the classic TinyOS example application `RadioCountToLeds` to `goodfet.ccspi` and verified compatibility [18, revision 954]. The author also refined the

GoodFET injection code to correctly handle lengths and checksum calculation. The advantage of this device over the Atmel RZUSBSTICK is that it can be reflashed with `goodfet.bsl` over USB with no need for a JTAG programmer, making it easier and less-expensive for a user to start using. Due to these features, the author provided a KillerBee driver for the Tmote Sky or TelosB running the GoodFET firmware (see 2.3.1). This driver, `dev_telosb.py`, implements the `SNIFF`, `SET_CHAN`, `INJECT`, and `PHYJAM_REFLEX` capabilities.

2.3 Software Tools

2.3.1 KillerBee Expansion

As discussed in 2.1, Joshua Wright's KillerBee 1.0 code offered a good base for 802.15.4/KillerBee software tools. The author's work in this section expanded upon the codebase to add several additional features useful for security assessments.

The first addition was supporting more devices. Both the Tmote Sky/TelosB (2.2.3) and Dartmouth Freakduino (2.2.2) previously described were supported by the author in a driver model. The driver model was designed to remove device-specific code from KillerBee's `__init__.py` file and move it to files specific to that device. Those files contain classes imported by KillerBee after the type of device being used as an interface is identified. Much work needed to be done to support serial USB devices (such as FTDI UART chips) which mount like `/dev/ttyUSB0` in addition to the standard USB bus supported by Wright's KillerBee 1.0.

Support for logging packets to a database schema has also been added by `dblog.py` and is discussed in 2.4. A tool called `zbdblog` provides packet capture to a database similar to `zbdump` for saving to Pcap.

In addition to the driver model, design changes were made to move a number of functions out of the KillerBee class. These functions, like `dev_list()`, do not need any of the information from the KillerBee class or its initialization. Thus, we are able to run these functions without the overhead of initializing the KillerBee class.

With additional support for capture interfaces and tools such as `zbwardrive` (see 2.3.2) that use KillerBee to monitor multiple frequencies, the data capture framework needs to support these. For large amounts of data, the added database logging functions (see 2.4) provide the ability to store metadata, but many users may not need or desire the overhead required to configure and run a MySQL database. The industry standard PCAP file is a good capture format, and was supported by the original KillerBee tool to store 802.15.4 packets. The author added support for the dBm (signal strength) and frequency (channel) data using the CACE Per-Packet Information [12] header format. Specifically, the 80211-Common field (which is modeled after the common Radiotap header) was used, as it is recognized by the standard and implemented by Wireshark and Scapy, among other tools. After discussions with Joshua Wright and Gerald Combs, the author has determined that future work should entail registering a IEEE 802.15.4 Common field with the specification, consisting of at least RSSI, capture interface, dBm, and channel or MHz fields. This type will have to be supported both by the KillerBee PCAP writer, but also should be implemented in Wireshark and Scapy. Finally, the author recommends that future work include implementation of the CACE PPI GEOLOCATION header, to support capture of latitude, longitude, and altitude to the PCAP format integrated with the packets.

2.3.2 `zbwardrive`: Automated Wardriving Tool

In addition to strengthening and expanding the KillerBee framework, the author also created some new tools to take advantage of the increased flexibility and tackle specific problems posed by practical security assessments. The `zbwardrive` tool is optimized for more active "wardriving" of ZigBee networks. Currently, of the many network discovery tools available for WiFi networks, only Kismet offers a plug-in framework with very rudimentary 802.15.4 support [1]. It was a difficult decision to develop a new tool instead of building upon the Kismet framework, but a tool was needed that addresses the specific challenges of wardriving 802.15.4/ZigBee networks, such as the need for injection to stimulate a quiet network to respond. Furthermore, in the initial stages of a security assessment, an auditor will not typically know the channels a target network is operating on, and if the auditor is trying to cover large geographic areas for analysis, they will need an automated way of finding these channels and then capturing data from them.

The author expanded beyond Mr. Wright's active scanning tools (`zbstumbler` and `zbfind`) to create a truly automated system that is feasible for wardriving. The resulting program utilizes multiple capture interfaces (that have been supported with KillerBee drivers) in an intelligent way between active network scanning (by injecting beacon request frames onto every ZigBee channel) and listening for a beacon response. If the zbwardrive tool receives a frame that decodes to an 802.15.4 beacon response when listening, this indicates that a network is currently operating on this frequency (and beaconing is enabled), and a free capture interface will be assigned to capture any traffic. If the packets on a channel subside for a period of time, the capturing device will be freed so it is available to capture other discovered networks. The desire to capture all active 802.15.4 based networks is not new, however previous implementations (such as Kevin Finisterre's work [16, 15]) are based on using one capture interface for each channel that a network *might* be on. The author's approach aims to more efficiently allocate physical capture device resources, hoping to have both less power and hardware demands as well as a lower visual profile. The wardriving software will be flexible in order to support other capture devices and schemes, including a directional antenna interface for use in capturing packets for direction finding. If the wardriving user has network connectivity, data collected from this tool will be transferred to the centralized database schema (see 2.4) for use by other tools, such as the author's packet generation tool.

2.4 Data Aggregation Schema and Database

2.4.1 Introduction

Typical network reconnaissance involves generating `pcap` files with file names describing where and when they were taken. However, flat-file data logging on individual devices is not a sustainable way to carry out a large scale assessment of a ZigBee network, due to both the multiple possible ways of obtaining the data, and the metadata which we wish to associate. For example, both the "zbplant" hardware (see 2.2.2) and the wardriving setup (see 2.3.2) may gather information, including GPS coordinates and RSSI indicators, in addition to actual packet captures. Other tools, such as the intelligent packet generation and network visualization software can access this information in order to build inferences upon it. For these reasons, the author presents a database schema which will include the ability to store parsed 802.15.4 packets, store cached inference data such as network flows and relationships between devices, and store GPS and received signal strength indicator (RSSI) metadata for the packets captured. This schema – and its MySQL implementation – is a research enabler for other tools described in this work, and can be adopted as a platform for future work in this subject area.

2.4.2 Prior Work

Although network tools typically log to a pcap or Daintree file format, there is precedent for database logging of network captures due to the advantages in searching and correlating large amounts of data, especially when it is gathered from multiple sensors. SNORT is the most widely deployed intrusion detection and prevention (IDS/IPS) system worldwide, and it supports a database capture module [33, 117] for SQL logging. The author is not aware of any tools intended for network analysis on wireless networks.

2.4.3 Implementation

To create this database, the author first designed the schema and then implemented it in MySQL data-definition language. The central table in the schema holds packets, which are time-stamped, labeled by channel, and dissected. Foreign key relations link this to a capture type table (i.e. if this was imported from a Pcap file or sent live to the database by a connected sniffer), as well as tables for location and devices. The device table stores devices by PAN identifier and long/short device addresses, and entries in this are linked by ID to source or destination fields in captured packets.

The data manipulation language for basic packet logging is handled by the KillerBee `dblog.py` extension. Adding a packet will call the author's Scapy `dot15d4` dissection, if it is not already dissected, and extract any available source or destination. If a database query finds that the device identified by the packet already exists in the devices table, it will reference that entry; otherwise it will add the entry first and then reference it. The same methodology is used for capturing location. If the KillerBee reports a location to the database logging module, it is searched to see if the location is already entered in the database (as determined equivalent by longitude, latitude, and elevation). If this is the case, the pre-existing entry is linked to the new packet; otherwise a new entry is created and linked. This system ensures that when the database is capturing data from the same device, or if it is capturing in a stationary position, that correlations are maintained and extra data is not stored when it contributes no additional information.

In order to quickly retrieve data for intelligent packet crafting, a number of MySQL views were created. Currently `devstatus` is used to retrieve the most up-to-date status for a given device, and `linkstatus` retrieves the current status between two devices. These are computed through intermediate views and depending on the implementation choice can be cached to provide performance improvements.

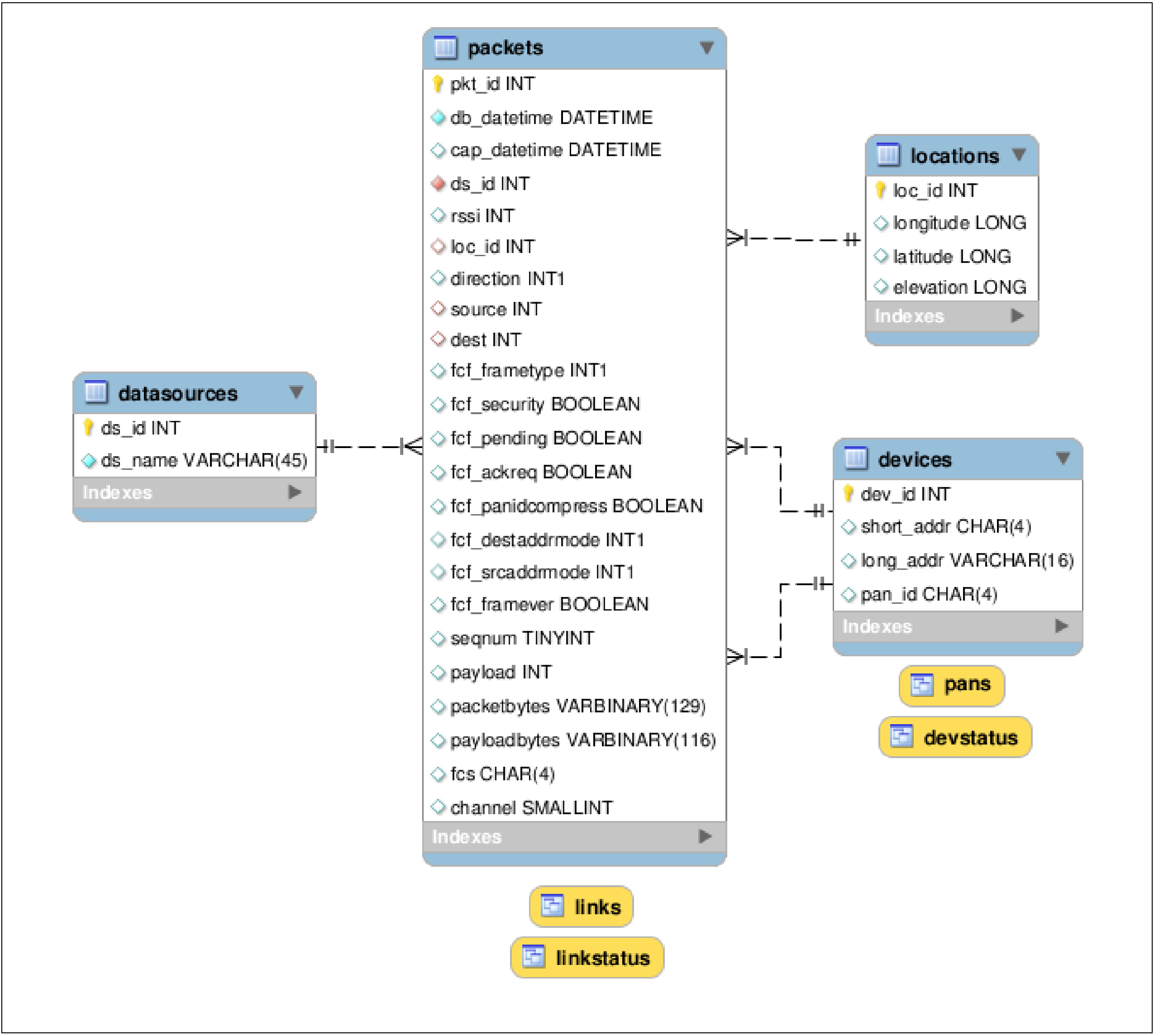

Figure 2.8: Entity-Relationship Diagram

This schema operates separately from Community Resource for Archiving Wireless Data At Dartmouth (CRAWDAD) [4] for logistical reasons, however the schema and sample captures have been made available for inclusion in the CRAWDAD project.

Chapter 3

Intelligent Crafting of Packets

3.1 Prior Work

Packet injection capability is necessary for an attacker to launch anything more than an attack on the confidentiality of a network, and in the case of ZigBee's encryption capabilities, even most confidentiality attacks may require injection capabilities. [9] points to "the logical next step after observing the system or network [is] testing its behaviors by manipulating the systems internal data or injecting custom-formatted messages into the network." As this statement alludes to, being able to inject packets into a network is critical for conducting a full security assessment, but this is only part of the puzzle. An effective attack requires packets tailored to the specific network environment in regards to the devices, the network topology, and even the current state of each device, such as its current sequence number. Thus, many attacks are not successful simply with a replay of packets or a pre-constructed packet. This leaves a security auditor needing to provide appropriate bytes of data for packets, in real time, while auditing, and possibly in the field under time pressure.

The 802.11 WiFi space has seen a number of tools emerge to help inject frames, and even provide some level of automation for crafting frames. These tools, such as airjack [6], file2air [42], libwlan [38], fakeap [8], and void11 [3], however, are often driver-specific for WiFi cards and do not understand the 802.15.4 or ZigBee networks or even the frame structure. Known packet injection tools which are currently available for ZigBee and 802.15.4 require the user to provide the bytes they wish to inject. This is either in the form of a libpcap or Daintree packet capture file being supplied to a utility such as KillerBee's `zbreplay`, or writing code that uses the KillerBee library to inject bytes that the user provides [43, tools/zbreplay, killerbee/__init__.py]. A BlackHat EU 2010 paper [17] suggests a tool that collects data and creates attack frames for wireless sensor networks, but the specific protocols targeted are unclear and the author cannot find any proof of a tool being produced. Some implementation of 802.15.4 packet generation was done in Metasploit as part of a project targeting the 6loWPAN protocol, however this work is based on a number of other modifications to the "firmware and processing daemon" which this work avoids in order to keep the tools portable and easier to use [10, 22, 17].

In summary, a tool to allow auditors to specify minimal data, and the type of frame to generate, and have the program determine the additional variables, craft a valid frame, and inject the frame is needed to enable serious security auditing for 802.15.4 and ZigBee networks. [9] points out that releases of tools and frameworks that provide such capabilities both provide the groundwork for future tools, but also provide a major educational opportunity, as studying the source and methods of such tools “benefits both attackers and defenders”.

3.2 Scapy `dot15d4` Extension

The author wrote a layer extension for Scapy, called `dot15d4`, which implements the most useful subset of the IEEE 802.15.4 protocol. The definition of the protocol was taken from a variety of sources, including the IEEE 802.15.4 Specification [24, 25], ZigBee Specification [45], and Daintree Network’s Getting Started with ZigBee and IEEE 802.15.4 primer [13]. The current status of the implementation is shown in Table 3.1.

Table 3.1: Implementation Status of Frame Types in Scapy `dot15d4`

Class Name	Notes
Dot15d4	
Dot15d4FCS	Calculates and appends FCS checksum when built
Dot15d4Ack	
Dot15d4Data	
Dot15d4Beacon	GTS and Pending addresses not fully implemented
Dot15d4Cmd	
Dot15d4CmdAssocReq	Not yet implemented
Dot15d4CmdAssocResp	Not yet implemented
Dot15d4CmdDataReq	Not yet implemented
Dot15d4CmdPANIDConflictNotify	Not yet implemented
Dot15d4CmdOrphanNotify	Not yet implemented
Dot15d4CmdBeaconReq	Not yet implemented
Dot15d4CmdCoordRealign	
Dot15d4AuxSecurityHeader	

In Figure 3.1, the complexity that occurs within beacon frames, as an example, is shown. The 802.15.4 Beacon specification alone, not even including the ZigBee beacon specification, has a number of variable length list fields, fields whose values are dependent on other fields, and fields whose existence is dependent on other fields. The Scapy dot15d4 layer handles the relation between these interconnected fields for the subset of the 802.15.4 which is implemented. For example, one simple example is that the frame control field source addressing mode is used to determine the existence and length of the frame’s source address field.

Although Scapy provides ConditionalField interfaces that can be adapted to work well for most questions of fields existing based on other fields values, it is more difficult to force the value of a field based on the value of another field. In many cases, we wish to leave this untouched by Scapy, so as to allow security researchers to craft frames that do not adhere strictly to the 802.15.4 standard for applications such as intelligent fuzzing or fingerprinting.

However, the author implements one such check to ensure that a basic constraint for acknowledgment frames is satisfied. This implementation demonstrates how other such constraints could be added, if needed. The constraint ensures that the frame control field addressing modes are both set to 0 (“None”) when an acknowledgment frame is set, ensuring that the default is “x02x00” for the frame control field bytes. This implementation is currently done in the `post_build` method of the `Dot15d4` class, which means that it is always satisfied after a build (triggered by `str()` or `show2()`), but it may not be reflected otherwise. This could also be implemented by creating a special field class for the field that the constraint is based upon, which has an `any2i()` method defined which performs the constraint checking and adjusting the dependent field’s value if needed. However, this requires both a special field class being defined and it would not enforce the constraint if the dependent field was changed after the constraining field was set.

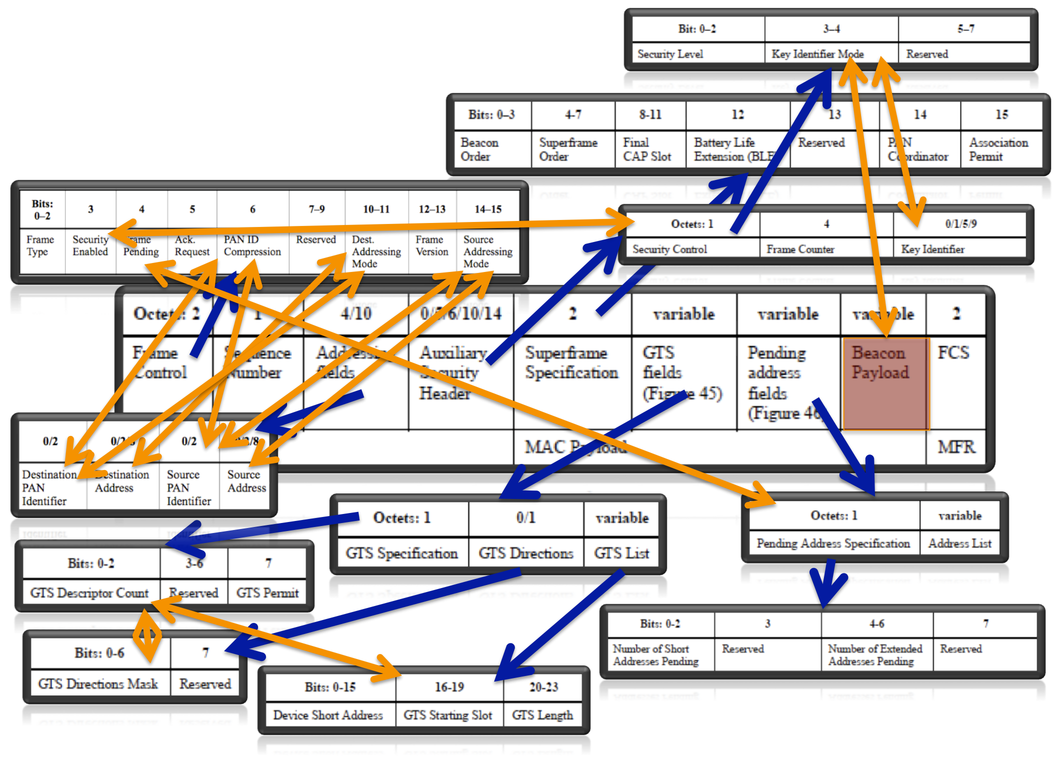

Figure 3.1: Relationships and dependencies between fields in 802.15.4 Beacon Frames

Additionally, the author’s Scapy `dot15d4` layer has been adopted as the base layer for the Scapy 6loWPAN layer written by Cesar Bernardini.

3.3 Inferenced Packet Generation

As discussed in 2.3.2, the author constructed techniques for gathering data about networks from a combination of active and passive listening. This information is loaded into the central database schema (2.4) over a “learning period” and data from multiple frames captured can be merged to inference data which may not be available from any single frame. Using this, tools can craft packets with the user having to define only minimal data. For example, given a type of frame and a target (destination) short address, the tool (a number of functions in `zbForge.py`) can pick an appropriate source address that has been observed having communication with the target, and use that address in the packet being constructed. The tool will also use a sequence number that follows from the most recent number observed, with some randomization added. The goal of this tool is to require minimal user interaction yet produce realistic looking packets that are more likely to be accepted by target devices, and which are more difficult to detect in packet analysis than packets generated without inferencing.

The database is programmed with a number of views to quickly return the most recent “inferred” status between devices. The tool retrieves this data based on the parameters supplied to it in the `makeLinkData` function, which can take None for one of its arguments. This function returns data, which along with the desired type of frame (as defined by the IEEE 802.15.4 standard), and uses the dot15d4 scapy module to produce a scapy object which can be injected. Figure 3.2 provides an example use of these functions to send a data frame.

```
kb = killerbee.getKillerBee(channel)
link = makeLinkData(srcTarget, destTarget)
byteframe, scapy = create(link, FRAME_802_DATA)
print "Sending forged frame:"
scapy.show2()
kb.inject(byteframe)
```

Figure 3.2: Example Usage of zbForge Functions

If a user specifies that they would like a beacon frame to be sent, they need only specify the sending device, and the remaining information (such as the sequence number, network PAN ID, long address) will be inferred. Many other types of packets similarly need only minimal information (usually sender and receiver) specified by the user, and the remainder of fields are calculated.

Chapter 4

Proof-of-Concept Attacks

The creation of attacks is a logical extension of being able to craft packets. After many types of packets can be generated on demand in an intelligent way to adapt to the current network state, a security auditor must be able to use these packets to implement a proof-of-concept attack.

4.1 Prior Work

Such work has been done extensively in the 802.11 WiFi realm. AirJack's `monkey_jack` does a man-in-the-middle to move a client off of its legitimate access point and associate it with an attacker-controlled access point [39, 174]. Other tools that implement disassociation attacks include `wlan_jack` and `fata_jack` [39, 175]. Authentication and association frames are sent in floods to the network by Void11 [3] to try to de-authenticate many hosts or overwhelm an access point [39, 175].The routing protocols can also be targeted, such as how the Ettercap plugin lamia tries to make an attacker into a root bridge (so most traffic flows through it) [39, 177].

There is literature discussing possible attacks on 802.15.4 and ZigBee, but these are theoretical in nature, and many of them discuss attacks that may be applicable more generally to wireless sensor networks (of which some 802.15.4 networks are a subtype). A review of current literature finds many interesting ideas about different attacks that may work on "wireless sensor networks" (WSNs), however the actual ability to implement these depends heavily on the routing protocols employed and the amount of security enabled in the network. Giannetsos, Dimitriou and Prasad propose an attack tool [37] for WSNs that will implement data replay, sinkhole, selective forwarding, flooding, and device reprogramming/code injection [17, 2]. This tool, however, currently focuses on the simple MultiHopLQI and MintRoute protocols. Hu proposes a defense against wormhole attacks [11], a concern which is echoed by Karlof and Wagner [26, 6], however such an attack may only be effective on specific geographic routing algorithms. A wormhole attack captures packets or individual bits at one location in the network and injects them, possibly after selectively discarding some packets, at another location. One outcome of a wormhole attack is to make nodes that are not neighbors believe that they may be within range of each other. Karlof and Wagner also discuss the possibility of spoofing, altering, or replaying routing information, as well as performing selective forwarding, a sybil (multiple identity) attack, a HELLO flood attack, or acknowledgment spoofing [26]. It is known that acknowledgment spoofing is possible on 802.15.4 networks, as acknowledgment (ACK) frames are not protected [35, 18]. However, implementing an effective attack with ACK spoofing requires implementing selective jamming, which is a difficult problem due to the actual radio interfaces and processing time. Krontiris, Dimitriou, and Giannetsos describe how the sinkhole attack might apply to two WSN routing protocols, MintRoute and MultiHopLQI, but do not offer an implementation or provide information relevant to ZigBee routing [28]. Wood discusses denial of service attacks such as jamming, tampering, collision, exhaustion, unfairness, neglect and greed, homing, misdirection, black holes, flooding, and desynchronization, but there is again no indication which of these may be relevant to ZigBee networks [40].

A few publications have focused more specifically on attacks on 802.15.4 and ZigBee networks. Zheng, Lee and Anshel discussed PHY and MAC layer attacks on 802.15.4 including jamming, capture and tamper, exhaustion, collision, and unfairness. Collision is basically selective jamming, and they point out that "collision with an acknowledgment frame will cause the sender to back off exponentially; collision with an association response frame will force the device to start the multi-step association procedure from the very beginning; and collision with several beacon frames from a beacon enabled coordinator will cause its children to get orphaned" [44, 5]. The paper concludes that "collision attacks are very effective and difficult to detect," but it is important to note that these attacks have a moderate to high implementation complexity as well. They also discussed some ZigBee attacks including route disruption by a compromised device or coordinator, a "void address" attack, and introducing a loop in a cluster tree. However, it is not clear which of the ZigBee attacks proposed are implementable, especially without having full access to a captured node. In analyzing the security protection offered by 802.15.4 and ZigBee, Silva and Nunes suggest several attacks including a compelling idea for instant DoS on networks using AES-CTR security with sequential freshness protection enabled and possible cases for cryptographic nonce reutilization [35, 34].

4.2 Reflexive Jamming

To make several of these attacks possible, specifically selective forwarding and acknowledgement spoofing, we need to be able to reflexively jam some transmissions. Notably, this technique will avoid some jamming defense strategies which try to differentiate jamming (intentional denial-of-service) from other non-malicious issues in the network. Wood claims that "a node can easily distinguish jamming from the failure of its neighbors by determining that constant energy, not lack of response, impedes communication" [40, 50]. However, a reflexive jamming attack, as implemented here, has no constant energy, and most devices – even those listening in a monitor/promiscuous mode – will not see the jamming frame independently of the frame that it jammed. Many authors classify such an attack as a collision attack, to separate it from "traditional" jamming. Wood states that these malicious collisions "create a kind of link-layer jamming, but no completely effective defense is known" [40, 51]. Some mitigation techniques are discussed in 4.4.1.

4.2.1 Prior Work

A brief paper on implementing reflexive jamming against IEEE 802.15.4 frames is [14], scheduled for presentation in two weeks. This paper provides good background on why such reflexive jamming is important, and benchmarking techniques for its effectiveness. However, this paper implements their jammer on a USRP2, calling it a "low-cost" system. The implementation presented here uses only a Tmote Sky (readily available at under $40), whereas the USRP2 system has a base price of $1,400 plus the $275 to $400 radio interface board[1]. [31] accomplishes reflexive jamming with a lower hardware cost (the presentation for the paper cites 80 Euros), specifically what seems to be two Atmel evaluation boards connected together. However, this paper unfortunately does not specify the specific Atmel hardware to use, the wiring between these two boards, and furthermore does not provide any source code to implement the results they discuss.

This thesis contributes an implementation that is publicly available, implementable on only one low-cost device, and the setup requires no physical alterations or wiring of the device.

4.2.2 Implementation on Tmote Sky with GoodFET

The author implemented reflexive jamming on the Tmote Sky device using GoodFET firmware. The author's code is in the GoodFET CCSPI firmware application, with the verb `0xA0`. The `goodfet.ccspi` application first places the radio into promiscuous mode, disables CRC checking, and tunes to the desired channel before placing the firmware into reflexive jamming mode. Reflexive jamming has been tested against communications and seems to have constant effectiveness against packets greater than 12 bytes in length. Corruption seems to typically occur on the 12th byte of the frame. The jam is triggered by the SFD (start of frame deliminator) line on the Chipcon2420 radio going high, as shown in Figure 4.1. When the firmware application observes this condition, it immediately shifts the radio into TX mode, loads a frame into the TXFIFO buffer, and sends the STXON strobe command to start transmission *without* the radio performing a CCA (clear channel assessment) which is designed to prevent collisions.

Figure 4.2 shows a series of beacon frames, with frame 13 and 14 being sent when jamming was not enabled, and frames 15 and 16 being sent once reflexive jamming was enabled. Note that the eighth byte (`0x46`) in the frame is the first byte to be corrupted by the reflexive jam. In the author's tests, this is the earliest byte jammed, due to the turn-around time between modes and the processor's clock cycle. This means that in the current implementation, frames of 7 bytes or shorter cannot be reflexively jammed, however, most frames of interest, including virtually any frame with a payload, will be of sufficient length to jam with this technique. The Chipcon2430 chip, which has the radio and an 8051 MCU on the same die, may enable successful jamming on shorter frames in future work.

If this attack is done without acknowledgment spoofing, the sending device may repeatedly try to transmit its packet as it is not being acknowledged. This means that an exhaustion attack on a sending device can be implemented using this technique, if the sending device is configured to continually attempt retransmission. The sending device would not know where

[1]Prices as of 5.31.2011 from `http://www.ettus.com/order`.

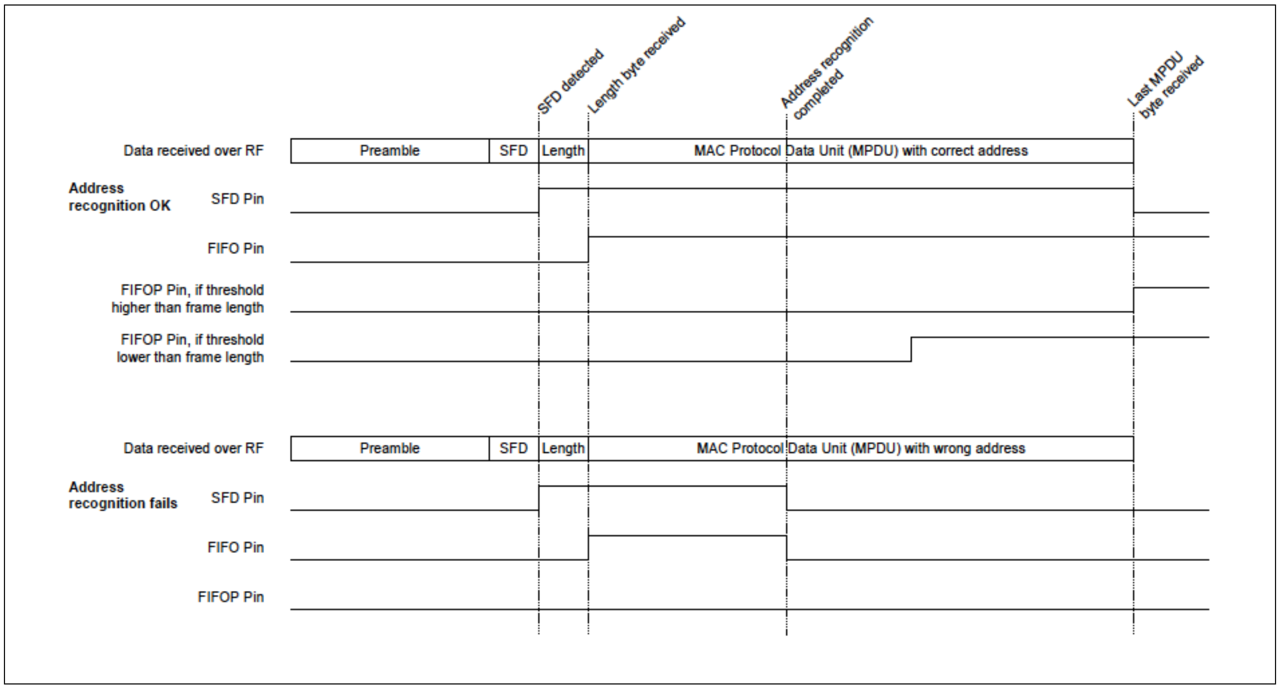
SFD detected
Length byte received
Address recognition completed
Last MPDU byte received
Data received over RF
Preamble
SFD
Length
MAC Protocol Data Unit (MPDU) with correct address
Address recognition OK
SFD Pin
FIFO Pin
FIFOP Pin, if threshold higher than frame length
FIFOP Pin, if threshold lower than frame length
Data received over RF
Preamble
SFD
Length
MAC Protocol Data Unit (MPDU) with wrong address
Address recognition fails
SFD Pin
FIFO Pin
FIFOP Pin

Figure 4.1: Chipcon 2420 Receive Operation Pin Values [36, 34]

in the frame the collision occurred in the frame, as it cannot monitor while it is transmitting due to the basic properties of radio media. An intrusion detection system may be able to monitor frames if it knew the expected frames to determine if collisions were occurring later in the frame, which may be unusual in non-malicious interference.

4.2.3 Acknowledgment Spoofing

In an acknowledgement spoofing attack, we want to prevent some frames from being received at their destination, while making the sender believe that these have been received. In the IEEE 802.15.4 specification, the acknowledgment bit is set in the frame control field if a sender desires their transmission to be acknowledged by the receiver. The acknowledgment frame type simply consists of a 2 byte frame control field (FCF), 1 byte sequence number (matching the sequence number of the frame being acknowledged), and a 2 byte frame check sequence (FCS). The FCF is known in advance for the acknowledgement field, and the FCS is a non-cryptographic checksum that can be calculated prior to sending the packet. The one piece of data which needs to be captured from the original packet is the sequence number. The sequence number is always the third byte of a 802.15.4 frame.

Most collision attacks need a frame to be generated and transmitted after another one is jammed, and these depend on the specific style of the attack. Implementation can be done using Scapy dot15d4 and KillerBee, even using the zbForge library to assist in packet generation, and then the frame can be transmitted either by the reflexive jamming device or another radio interface. Choosing the proper frames to jam, and generating the proper preceding or follow-up frames is key to making an effective, as well as targeted, attack.

4.3 Attack on AES-CTR with Replay Protection Enabled

One attack suggested by Silva and Nunes [35, 34] is an instant denial-of-service attack on networks using AES-CTR security with sequential freshness (replay) protection enabled. The 802.15.4 standard has several security suites. First, it could use no security. The second suite is AES-CTR, which provides data confidentiality. The standard also has AES-CBC-MAC (with 32, 64, or 128 bits of MAC) which provides authentication. Finally, the AES-CCM (with 32, 64, or 128 bits of MAC) provides both confidentiality and authentication. Because in AES-CTR the frame is not authenticated, and the payload does not need to be something that decrypts correctly (it is not validated), then a forged frame could be sent by the attacker with crafted fields. If freshness protection is enabled, the attacker can set the frame counter to the maximal value (`0xFFFFFFFF`) and the key sequence counter to maximal value (`0xFF`) and the receiver will check these, see that they are higher than the current counters, and begin to process the frame. It will perform decryption, which could result in a non-sensical decrypted payload, but this payload is not checked at the MAC layer. The MAC layer then updates the internal frame and key sequence counters in the access control list for the source to the last received values (which are the maximum values), and waits for more packets. However, these legitimate incoming frames will have frame and key sequence counter values less than the maximum values, and these will be dropped as soon as the MAC layer begins checking for sequential freshness, before any payload is decrypted and forwarded to the upper layers in the network stack.

The author did not have access to any commercial devices that seem to be functioning in this mode, but was able to write a proof-of-concept implementation using the Scapy dot15d4 layer. Figure 4.3 demonstrates the ease of crafting specialty packets using zbForge and Scapy dot15d4. For such a simple attack, moreover, the zbForge capability is not needed, and the attack could be written in in a single line of code using Scapy and KillerBee. The zbForge functionality, however, is demonstrated as an example.

4.4 Remediation

4.4.1 Mitigation of Frame Collisions

The reflexive jamming attack presents a number of problems for mitigation. At a fundamental level, frames are sometimes corrupted by the natural environment and a protocol must be able to treat this as a normal case. However, detection of malicious (intentional) interference is difficult if reflexive jamming (collisions) are used. Most radio chips will not see the second frame as an anomaly, because they will receive the frame length from the original legitimate frame, and their radios will then listen for that many bytes. The jamming frame begins partly through the legitimate frame and alters multiple bytes by collision. However, any excess bytes in the jamming frame that are sent after the legitimate frame has ended will not be heard by most radios (a software-defined radio could watch for and analyze these). However, even if these bytes could be examined, the attacker could vary the content of the jamming frame to make these bytes appear as natural noise in the environment. Intrusion-detection systems may need to consider the frequency of collisions, developing a model for the percent of frames which are corrupted by the RF environment during normal operation and comparing this model against current conditions to see if a significant increase has occurred in the number of corrupted frames. An attacker may be able to be successful in running their attack undetected, however, if they do selective reflexive jamming. In such a technique, the attacker does not simply jam each time a start-of-frame is heard, but instead examines the first few bytes – for example the 2-byte frame control sequence – to determine if the frame is of interest. For example, only beacon frames could be jammed in order to try and segment a network, but the attacker may not interfere with other types of frames. This may lead to a race between defenders to develop accurate models of normal network error rates versus attackers to craft their attacks such that they fit into the accepted "normal" model. Defense in this way is unlikely to be a viable long-term solution against determined attackers.

Wood proposes using error correcting codes to mitigate the issue of collisions, but notes that these add overhead to the network, and although these may work well against environmental or probabilistic errors, an attacker can corrupt more bytes than the error-correcting code can correct for, and accomplish an effective collision in this way. Also, other protocols could be used, such as [41], which is specifically designed to fight against jamming.

4.4.2 Mitigation of Forged Acknowledgment Frames

Silva and Nunes propose remediation against forged ACK frames by introducing a new frame type to 802.15.4 that is an acknowledgement frame secured by the same security suite as the rest of the messages, which they call a "protected ACK". This frame would be significantly longer in length than the current ACK frame, however, and Silva and Nunes propose that it carries a payload of the Frame Counter and Key Sequence Counter, the External Optional Frame Counter, and the External Optional Key Sequence Counter [35, 23]. Implementation of this will need to be added to the standard, and implemented in chip designs. This implementation cycle would take a long time, and new chips would likely need to be backwards compatible given the long deployment lifetime expected of devices already deployed in the field.

4.4.3 Mitigation of Fast DoS Attacks

Another remediation proposed by Silva and Nunes aims to prevent fast DoS attacks on devices using the AES-CTR security suite as well as replay protection. The proposal is to keep a Trust Reference Value (TRV) that is used to validate the incoming frame counter and key sequence counter. First the frame counter and key sequence counter are validated as they are currently, to ensure they exceed the values stored in the access-control list (ACL) for the given source address. If they pass this test, they are also examined to ensure that their values are not more than the TRV above the values stored in the ACL [35, 24]. If the distance between the new value and the stored value exceeds the amount allowed by the TRV check, Silva and Nunes suggest that the receiver "asks" the sender to verify the given value. This query and response would need to be protected as well. This ensures that large jumps are not possible in these values, such as immediately maximizing the value of these fields. However, this does not prevent an attacker from slowly ramping up the values using jumps slightly smaller than that disallowed by the TRV check. If this technique was combined with intrusion-detection or intrusion-prevention systems, such systems may be able to detect a pattern before the attack increases the value of the counters significantly. However, even raising it by a few iterations would deny service for some number of frames.

Acknowledgements

The author extends his thanks to Professor Sergey Bratus for his mentoring, advising, and encouragement and Professor Sean Smith for the support of the Trust Lab.

The author wishes to thank Ricky Melgares for his collaboration on 802.15.4/ZigBee technology research, as well as Travis Goodspeed, Joshua Wright, and Kevin Finisterre for their correspondence and assistance on various issues.

The research conducted would not have been possible without the lessons and support provided by the hacker research community. Thanks goes out to everyone in this community, and specifically the organizers of the conferences the author was able to attend: Heidi and the ShmooGroup at ShmooCon 2011, FX and Phenoelit at ph-neutral 0x7db, and the organizers of ToorCon San Diego 2010.

At Dartmouth, the author thanks Rebecca Shapiro, Gabe Weaver, Jason Reeves, Joe Cooley, James Oakley, and the other members of the Trust Lab for their brainstorming and support. Further thanks goes to Conor Galligan and Callen Votzke for their assistance with CAD and rapid prototyping. I would also like to thank my many friends in Dartmouth Emergency Medical Services, Kelsey Woerner, and my family for their exceptional support and friendship.

This research and software was supported in part by funding from the Department of Energy under Grant Award Number DE-OE0000097 Subaward 2010-01251-01 as well as a research program at the Institute for Security, Technology, and Society at Dartmouth College, supported by the National Science Foundation under Grant Award Number 1016782. The views and conclusions contained in this document are those of the author and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the National Science Foundation or the Department of Energy.

```
Frame 13 (13 bytes on wire, 13 bytes captured)
    [Time delta from previous captured frame: 0.999994000 seconds]
    Frame Length: 13 bytes
IEEE 802.15.4 Beacon, Src: 0xef01
    Frame Control Field: Beacon (0x8000)
    FCS: 0x9fba (Correct)
0000  00 80 ac 01 39 01 ef 46 cf 00 00 ba 9f            ....9..F.....

Frame 14 (13 bytes on wire, 13 bytes captured)
    [Time delta from previous captured frame: 0.999993000 seconds]
    Frame Length: 13 bytes
IEEE 802.15.4 Beacon, Src: 0xef01
    Frame Control Field: Beacon (0x8000)
    FCS: 0xd247 (Correct)
0000  00 80 ad 01 39 01 ef 46 cf 00 00 47 d2            ....9..F...G.

Frame 15 (13 bytes on wire, 13 bytes captured)
    [Time delta from previous captured frame: 0.000094000 seconds]
    Frame Length: 13 bytes
IEEE 802.15.4 Beacon, Src: 0xef01, Bad FCS
    Frame Control Field: Beacon (0x8000)
    FCS: 0x3b7c (Incorrect, expected FCS=0x215a
    [Expert Info (Warn/Checksum): Bad FCS]
[Malformed Packet: IEEE 802.15.4]
    [Expert Info (Error/Malformed): Malformed Packet (Exception occurred)]
0000  00 80 ae 01 39 01 ef 93 99 99 b9 7c 3b            ....9......|;

Frame 16 (13 bytes on wire, 13 bytes captured)
    [Time delta from previous captured frame: 0.999994000 seconds]
    Frame Length: 13 bytes
IEEE 802.15.4 Beacon, Src: 0xef01, Bad FCS
    Frame Control Field: Beacon (0x8000)
    FCS: 0x3a42 (Incorrect, expected FCS=0xcbea
    [Expert Info (Warn/Checksum): Bad FCS]
[Malformed Packet: IEEE 802.15.4]
    [Expert Info (Error/Malformed): Malformed Packet (Exception occurred)]
0000  00 80 af 01 39 01 ef 33 33 33 33 42 3a            ....9..3333B:
```

Figure 4.2: Jammed 802.15.4 Beacon Frames

```
#Usage: clear; sudo python dos_aesctr_replay.py -c 15 -s '2add' -d '0000' -v
kb = getKillerBee(channel)
link = makeLinkData(srcSearch, destSearch)
_, scapy = create(link, FRAME_802_DATA) # get our basic data frame
# If "force" src/dest/pan provided, change from those that
#  our search automatically filled in to the user specified settings
if srcTarget is not None:
   scapy.src_addr = int(srcTarget, 16)
if destTarget is not None:
   scapy.dest_addr = int(destTarget, 16)
if panTarget is not None:
   scapy.src_panid = scapy.dest_panid = int(panTarget, 16)
# Weaponize this frame for the DoS Attack on AES-CTR
scapy.fcf_security = True
scapy.aux_sec_header.sec_framecounter = 0xFFFFFFFF
scapy.aux_sec_header.sec_sc_keyidmode = "KeyIndex"
scapy.aux_sec_header.sec_keyid_keyindex = 0xFF
scapy.aux_sec_header = scapy.aux_sec_header #needed to update main packet
# Output and send frame
print "Sending forged frame:", toHex(str(scapy))
scapy.show()
kb.inject(str(scapy))
```

Figure 4.3: Proof-of-Concept Implementation of a Single-Frame DoS Attack on AES-CTR Security with Replay Protection Enabled

Bibliography

[1] Kismet-dot15d4. https://www.kismetwireless.net/code/svn/trunk/plugin-dot15d4/README.

[2] *SUP500F Datasheet.* http://www.sparkfun.com/datasheets/GPS/Modules/SUP500F_v3.pdf.

[3] Void11. http://www.wirelessdefence.org/Contents/Void11Main.htm, 2004.

[4] Community resource for archiving wireless data at dartmouth. http://crawdad.cs.dartmouth.edu/index.php, 2010.

[5] Q51 PANalyzer. http://www.exegin.com/hardware/q51app.php, 2010.

[6] abadd0n and xx25. Airjack. http://sourceforge.net/projects/airjack/.

[7] Akiba. Introducing the freakduino-chibi, an arduino-based board for wireless sensor networking. http://freaklabs.org/index.php/Blog/Store/Introducing-the-Freakduino-Chibi-An-Arduino-based-Board-For-Wireless-Sensor-Networking.html, November 2010.

[8] Black Alchemy. Fake ap. http://www.blackalchemy.to/project/fakeap/, March 2005.

[9] Sergey Bratus. Hacker curriculum: How hackers learn networking. *IEEE Distributed Systems Online*, 8(10), 2007. http://ieeexplore.ieee.org/iel5/8968/4384580/04384582.pdf.

[10] Luca Bruno. Security in lossy and low-power networks: Tools for 6lowpan. Technical report, POLITECNICO DI TORINO, 2010.

[11] Yih chun Hu. Packet leashes: A defense against wormhole attacks in wireless ad hoc networks. 2003. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.113.4100.

[12] Gerald Combs, Gianluca Varenni, and Dustin Johnson. *Per-Packet Information Header Specification.* CACE Technologies, 1.0.1 edition, June 2007.

[13] Daintree Networks. *Getting Started with ZigBee and IEEE 802.15.4*, 2008. www.daintree.net/downloads/whitepapers/zigbee_primer.pdf.

[14] Disco Labs, TU Kaiserslautern. *Short Paper: Reactive Jamming in Wireless Networks - How Realistic is the Threat?*, Hamburg, Germany, June 2011. WiSec '11.

[15] Kevin Finisterre. Africanized swarm? wtf killerbees you say!? Joint Direct Attack Munition Smart Weaponry Blog, October 2010.

[16] Kevin Finisterre. Successful zigbee wardrive rig is online! Joint Direct Attack Munition Smart Weaponry Blog, October 2010. `http://www.digitalmunition.com/_/Blog/Entries/2010/10/13_Successful_Zigbee_Wardrives!.html`.

[17] Thanassis Giannetsos, Tassos Dimitriou, and Neeli R. Prasad. Weaponizing wireless networks: An attack tool for launching attacks against sensor networks. BlackHat EU, 2010. `https://media.blackhat.com/bh-eu-10/whitepapers/Giannetsos/BlackHat-EU-2010-Giannetsos-Weaponizing-Wireless-Networks-wp.pdf`.

[18] Travis Goodspeed. Goodfet.

[19] Travis Goodspeed. Extracting keys from second generation zigbee chips. Las Vegas, Nevada, July 2009. BlackHat USA. `http://www.blackhat.com/presentations/bh-usa-09/GOODSPEED/BHUSA09-Goodspeed-ZigbeeChips-PAPER.pdf`.

[20] Travis Goodspeed. Reversing and exploiting wireless sensors. Arlington, VA, February 2009. BlackHat Federal.

[21] Mikal Hart. *NewSoftSerial Library*. Arduiniana Blog, January 2010. `http://arduiniana.org/libraries/newsoftserial/`.

[22] Mikal Hart. *TinyGPS Library*. Arduiniana Blog, January 2011. `http://arduiniana.org/libraries/tinygps/`.

[23] Paul Henry. Expansion of wireless introduces new risks. Optimal Security Blog, April 2010. `http://blog.lumension.com/?p=698`.

[24] IEEE Computer Society, LAN/MAN Standards Committee, New York, NY. *Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low Rate Wireless Personal Area Networks (LR-WPANs)*, ieee 802.15.4-2006 edition, September 2006. `http://standards.ieee.org/getieee802/download/802.15.4-2006.pdf`.

[25] IEEE Computer Society, LAN/MAN Standards Committee, New York, NY. *Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low Rate Wireless Personal Area Networks (LR-WPANs), Amendment 1: Add Alternate PHYs*, ieee 802.15.4a-2007 edition, August 2007. `http://standards.ieee.org/getieee802/download/802.15.4a-2007.pdf`.

[26] Chris Karlof and David Wagner. Secure routing in wireless sensor networks: Attacks and countermeasures. In *In First IEEE International Workshop on Sensor Network Protocols and Applications*, number 113-127. University of California at Berkeley, 2002.

[27] Nathan Keltner and Shawn Moyer. Wardriving the smart grid. In *BlackHat 2010*, 2010. `http://www.securitytube.net/Wardriving-the-Smart-Grid-(Blackhat-2010)-video.aspx`.

[28] Ioannis Krontiris, Thanassis Giannetsos, and Tassos Dimitriou. Launching a sinkhole attack in wireless sensor networks; the intruder side.

[29] LOCOSYS Technology Inc. *LS20031 Datasheet*, 1.2 edition, July 2009. `http://www.sparkfun.com/datasheets/GPS/Modules/LS20030%7E3_datasheet_v1.2.pdf`.

[30] Microchip Technology Incorporated. *1024K I2C CMOS Serial EEPROM Datasheet*, 2011. `http://ww1.microchip.com/downloads/en/DeviceDoc/21941H.pdf`.

[31] Colin P. O'Flynn. Message denial and alteration on ieee 802.15.4 low-power radio networks. *NewAE, IEEE*, 2011.

[32] Adrian Perrig, John Stankovic, David Wagner, and Caren Rosenblatt. Security in wireless sensor networks. *Communications of the ACM*, 47:53–57, 2004.

[33] Snort Project. Snort users manual 2.9.0. `http://www.snort.org/assets/166/snort_manual.pdf`, January 2011.

[34] Rui Silva and Serafim Nunes. Security issues on zigbee, July 2005.

[35] Rui Silva and Serafim Nunes. Security in ieee 802.15.4 standard, January 2006. `www.estig.ipbeja.pt/~rmss/uploads/materaJan06-3.pdf`.

[36] Texas Instruments. *CC2420: 2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver*. Dallas, TX, SWRS041B edition, 2010.

[37] Giannetsos Thanassis and Dimitriou Tassos. Sensys attack tool. `http://www.ait.gr/ait_web_site/Phd/agia/SenSys/sensys.html`, 2011.

[38] tuxfamily.org. libwlan. `http://www.wireless-warrior.org/detail/1062/libwlan.html`, November 2003.

[39] Andrew Vladimirov, Konstantin V. Gravrilenko, and Andrei A. Mikhailovskiy. *Wi-Foo: The Secrets of Wireless Hacking*. Pearson Education, 2004.

[40] Anthony D. Wood, John A. Stankovic, Anthony D, and John A. Denial of service in sensor networks. In *Upper Saddle River*. Prentice–Hall, Inc, 2002. `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.20.6380`.

[41] Anthony D. Wood, John A. Stankovic, and Gang Zhou. Deejam: Defeating energy-efficient jamming in ieee 802.15.4-based wireless networks. Technical report, Department of Computer Science, University of Virginia. `http://www.cs.virginia.edu/~stankovic/psfiles/secon07-deejam.pdf`.

[42] Joshua Wright. File2air - file-based wireless packet injection. `http://www.willhackforsushi.com/File2air.html`, October 2007.

[43] Joshua Wright. Killerbee: Framework and tools for exploiting zigbee and ieee 802.15.4 networks. GoogleCode Repository, 2010. `http://code.google.com/p/killerbee/source/browse/trunk/killerbee/`.

[44] Jianliang Zheng, Myung J. Lee, and Michael Anshel. Towards secure low rate wireless personal area networks. *IEEE TRANSACTIONS ON MOBILE COMPUTING*, 5(10), 2006.

[45] ZigBee Alliance. *ZigBee Specification*, January 2008.