An Empirical Analysis of Security and Privacy in Health and Medical Systems

by

Michael A. Rushanan

A dissertation submitted to The Johns Hopkins University in conformity with the requirements for the degree of Doctor of Philosophy.

Baltimore, Maryland

May, 2016

Abstract

Healthcare reform, regulation, and adoption of technology such as wearables are substantially changing both the quality of care and how we receive it. For example, health and fitness devices contain sensors that collect data, wireless interfaces to transmit data, and cloud infrastructures to aggregate, analyze, and share data. FDA-defined class III devices such as pacemakers will soon share these capabilities. While technological growth in health care is clearly beneficial, it also brings new security and privacy challenges for systems, users, and regulators.

We group these concepts under health and medical systems to connect and emphasize their importance to healthcare. Challenges include how to keep user health data private, how to limit and protect access to data, and how to securely store and transmit data while maintaining interoperability with other systems. The most critical challenge unique to healthcare is how to balance security and privacy with safety and utility concerns. Specifically, a life-critical medical device must fail-open (i.e., work regardless) in the event of an active threat or attack.

This dissertation examines some of these challenges and introduces new systems

that not only improve security and privacy but also enhance workflow and usability. Usability is important in this context because a secure system that inhibits workflow is often improperly used or circumvented. We present this concern and our solution in its respective chapter. Each chapter of this dissertation presents a unique challenge, or unanswered question, and solution based on empirical analysis.

We present a survey of related work in embedded health and medical systems. The academic and regulatory communities greatly scrutinize the security and privacy of these devices because of their primary function of providing critical care. What we find is that securing embedded health and medical systems is hard, done incorrectly, and is analogous to non-embedded health and medical systems such as hospital servers, terminals, and personally owned mobile devices. A policy called bring your own device (BYOD) allows the use and integration of mobile devices in the workplace. We perform an analysis of Apple iMessage which both implicates BYOD in healthcare and secure messaging protocols used by health and medical systems.

We analyze direct memory access engines, a special-purpose piece of hardware to transfer data into and out of main memory, and show that we can chain together memory transfers to perform arbitrary computation. This result potentially affects all computing systems used for healthcare. We also examine HTML5 web workers as they provide stealthy computation and covert communication. This finding is relevant to web applications such as personal and electronic health record portals.

We design and implement two novel and secure health and medical systems. One is

ABSTRACT

a wearable device that addresses the problem of authenticating a user (e.g., physician) to a terminal in a usable way. The other is a light-weight and low-cost wireless device we call Beacon+. This device extends the design of Apple's iBeacon specification with unspoofable, temporal, and authenticated advertisements; of which, enables secure location sensing applications that could improve numerous healthcare processes.

Primary Reader: Dr. Aviel D. Rubin

Secondary Readers: Dr. Anton Dahbura and Dr. Christoph U. Lehmann

Acknowledgments

I would like to thank foremost Professor Avi Rubin, my advisor, for his mentorship and patience. He inspired my persistence to push forward when I encountered challenges in graduate school. I have grown both personally and professionally because of his guidance and attitude toward computer science.

I am thankful for the guidance and support of many others at Johns Hopkins University. In particular, the computer science office, faculty, and the Distributed Systems, Crypto, and Health Medical Security labs. I enjoyed late night musings, debates, paper writing, and coding with Ayo Akinyele, Christina Garman, Gabriel Kaptchuk, Paul Martin, Ian Miers, Matthew Pagano, David Russell, and Tom Tantillo, who are all top-notch researchers that are positioned to change our field.

Each of these labs has its set of research scientists and professors. I am thankful for the advisement provided by Stephen Checkoway and Matthew Green. I learned more than reverse engineering and cryptography because of them. I learned how to give an academic talk, write a technically sound paper, and deal with pitfalls and troubles of research.

ACKNOWLEDGMENTS

I am also thankful for the opportunity to visit and gain new insight from Kevin Fu's SPQR lab at the University of Michigan. It is here I learned about academic genealogy thanks to Peter Honeyman, pulled apart a pacemaker or two, and explored embedded system design. The time away from Baltimore was enlightening.

Some chapters of this dissertation have appeared in publications or submissions to academic conferences and workshops. I would like to thank my co-authors on each publication. They include Joseph Carrigan,[1] Stephen Checkoway,[2] Denis Foo Kune,[3] Christina Garman,[4] Gabriel Kaptchuk,[4] Matthew D. Green,[4] Paul Martin,[1,5] Ian Miers,[4] Avi Rubin,[3,5,6] David Russel,[6] Colleen Swanson,[3] and Thomas Tantillo.[5] I may have reproduced some or part of these works in my dissertation. These works may also appear in the later works of my co-authors.

The Health and Human Services-funded, "Strategic Health IT Advanced Research Projects on Security" (SHARPS), and the NSF-funded, "Trustworthy Health and Wellness" (THaW), grants both supported my research. I am thankful to these organizations for their financial support and open access to inter-university collaboration.

I would like to thank my brother, Brandon Hill, parents, Michael and Kimberly Hill, my paternal grandparents, Jackie Wilson and Myrtle Hill, and maternal grandparents, Michael and Helen Rushanan. My brother gave up a lot of hangout time for me to finish this. My grandfather Michael Rushanan provided an infinite source of inspiration through his various stories. Jen Menzer, thanks for keeping me sane.

Dedication

This thesis is dedicated to my loving family and friends. To those of you who stuck around, thanks for putting up with me. You are the reason I finished this journey.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The recent Apple Watch is a health and fitness device that contains a heart rate sensor and a pedometer to collect, disseminate, and display health data to its user. HealthKit is the application that operates on this data and provides extensions to third-parties to access and aggregate data over a large number of users and tertiary sensor plugins (e.g., a digital sphygmomanometer). Cloud platforms are used to perform data aggregation, and thus, private health data is distributed and fragmented across geographical regions. Further, regulatory bodies must decide how to classify health and fitness devices as they integrate into health information exchanges and become comparable to FDA-defined class I, II, and III medical devices (e.g., pacemakers).

While technological growth in health care is beneficial, it also brings new security and privacy challenges for systems, users, and regulators. The Apple Watch, for

example, collects private health information from its wearer, performs some local computation on it, and then transmits it to the cloud or a third-party. This health information is confidential and private. Thus, it needs to be encrypted at rest and transmitted over a secure channel such as SSL/TLS. Moreover, cloud providers must enforce access control on all collected health information, and third-parties (e.g., a physician at a local clinic) must authenticate before accessing and manipulating aggregate health information.

There exist many security and privacy challenges to the Apple Watch as a health and fitness device. These challenges extend to all healthcare-related devices. As such, we group these concepts under the term of health and medical systems to connect and emphasize their importance to healthcare. This grouping also includes the most critical challenge unique to healthcare, balancing security and privacy with safety and utility.

This dissertation examines some of these challenges in health and medical systems and introduces new systems that improve security, privacy, and usability. We present a survey of related work in implantable medical devices (IMDs) and body area networks (BANs), otherwise referred to as embedded health and medical systems. The publications reviewed in this survey aim at improving security and privacy, but as we find in our analysis of common themes and trends, systems are hard to design. Numerous vulnerabilities in deployed systems and proposed protocols support this assertion.

This finding is analogous to non-embedded health and medical systems such as hospital servers, terminals, and BYOD. For example, Apple iMessage is one of the most widely deployed end-to-end encryption messaging protocols. It is used by BYOD devices to send confidential messages between nurses, and other practitioners.[7] We perform an analysis of Apple iMessage and find that it has significant vulnerabilities that can be exploited by a sophisticated attacker. This finding exemplifies the challenges of BYOD and protocol development in health and medical systems.

Modern computer systems contain a variety of special purpose processors designed to offload specific tasks from the CPU such as graphics rendering for oncology imaging.[8] Direct memory access engines perform the copying of data from main memory to the other processor. We show that the ability to chain together memory copying is sufficient to perform arbitrary computation. This means, in the context of healthcare, that an attacker can perform any function on a health and medical system (e.g., ultrasound equipment) with the capabilities we describe in Chapter 4.

Epic, a healthcare software company, and other related businesses implement a suite of software for patient engagement, clinicals, portals, and third-party extensions such as billing. This software is useful as it provides access to electronic health records (EHR) and health information exchanges (HIE). Typically, these softwares interact with users via an HTML-driven user interface (UI). There exists an entire community dedicated to expanding the knowledge of web application vulnerabilities, which would affect these UIs, with a particular focus on the new HTML5 APIs. We

examine the Web Workers API and find that it provides stealthy computation and covert communication.

Stealthy computation can affect the computer systems of patients and physicians visiting a compromised web application. In particular, we can use their computational resources to perform a distributed operation such as password cracking. We can mount a denial-of-service (DoS) attack that causes OS X systems to halt, and we can perform a resource depletion on mobile devices, again affecting BYOD. Also, health and medical systems that expose a full operating system, such as the Baxa ExactaMix 2400 pharmaceutical compounder,[9] are susceptible to these stealthy computations.

The most common method for a user to gain access to a system is to authenticate (i.e., verify her identity), with a password. It's important that systems in a healthcare environment, for example, a workstation-on-wheels (WOW), require authentication. However, complex password requirements and policies can be perceived as inhibiting care and thus users may try to circumvent them. Examples include never logging out of systems and sharing user credentials with others. Moreover, the invention of systems to improve security while not hindering usability is often circumvented if not properly implemented. Sinclair and Smith[10] describe a proximity-based deauthentication system that uses cameras to determine when if a system is in use. In a hospital setting, users were frustrated with bugs where the system misinterpreted movements and logged off users, so the users simply covered the cameras with cups.

We introduce an authentication system that addresses the problem of complex

passwords and poor usability by prompting the user to enter a password as infrequently as once a day. We design a wearable bracelet that stores authentication information for the user upon logging in and then transmitting that information to each system upon use. The bracelet is not a component in multi-factor authentication, rather a mechanism for enhancing usability and workflow while maintaining stringent access control policies.

Tracking and managing assets in real-time are critical for hospitals as they impact patient care. In particular, tracking needs to be secure against both active and passive attacks that misappropriate assets. We implement a real-time tracking system using a Bluetooth low-energy (BLE) device we call Beacon+. This device enables other secure location sensing applications such as location-based access restrictions, whereby a physician or nurse can only access the medical records of nearby patients. Location in this application is one factor in a multi-factor access control scheme. For example, physicians who step away from their personal computer system (assuming she can access all records here) take a hospital-issued tablet with them, log in to the tablet, and be within close physical proximity of a patient to access her records.

1.1 Our Approach

This dissertation examines the aforementioned security and privacy challenges in health and medical systems and presents new systems that are usable and secure.

These systems also enable new applications that could improve numerous healthcare processes. For brevity, we describe each challenge and system independently and summarize our approach in the below.

1.1.1 SoK: Security and Privacy in Implantable Medical Devices and Body Area Networks

Balancing security, privacy, safety, and utility is a necessity in the health care domain, in which implantable medical devices (IMDs) and body area networks (BANs) have made it possible to continuously and automatically manage and treat a number of health conditions. In this work, we survey publications aimed at improving security and privacy in IMDs and health-related BANs, providing clear definitions and a comprehensive overview of the problem space. We analyze common themes, categorize relevant results, and identify trends and directions for future research. We present a visual illustration of this analysis that shows the progression of IMD/BAN research and highlights emerging threats. We identify three broad research categories aimed at ensuring the security and privacy of the telemetry interface, software, and sensor interface layers and discuss challenges researchers face with respect to ensuring reproducibility of results. We find that while the security of the telemetry interface has received much attention in academia, the threat of software exploitation and the sensor interface layer deserve further attention. In addition, we observe that while

the use of physiological values as a source of entropy for cryptographic keys holds some promise, a more rigorous assessment of the security and practicality of these schemes is required.

1.1.2 Dancing on the Lip of the Volcano: Chosen Ciphertext Attacks on Apple iMessage

Apple's iMessage is one of the most widely-deployed end-to-end encrypted messaging protocols. Despite its broad deployment, the encryption protocols used by iMessage have never been subjected to rigorous cryptanalysis. In this paper, we conduct a thorough analysis of iMessage to determine the security of the protocol against a variety of attacks. Our analysis shows that iMessage has significant vulnerabilities that can be exploited by a sophisticated attacker. In particular, we outline a novel chosen ciphertext attack on Huffman compressed data, which allows *retrospective* decryption of some iMessage payloads in less than 2^{18} queries. The practical implication of these attacks is that any party who gains access to iMessage ciphertexts may potentially decrypt them remotely and after the fact. We additionally describe mitigations that will prevent these attacks on the protocol, without breaking backwards compatibility. Apple has deployed our mitigations in the latest iOS and OS X releases.

1.1.3 Run-DMA

Copying data from devices into main memory is a computationally-trivial, yet time-intensive, task. In order to free the CPU to perform more interesting work, computers use direct memory access (DMA) engines — a special-purpose piece of hardware — to transfer data into and out of main memory. We show that the ability to chain together such memory transfers, as provided by commodity hardware, is sufficient to perform arbitrary computation. Further, when hardware peripherals can be accessed via memory-mapped I/O, they are accessible to "DMA programs." To demonstrate malicious behavior, we build a proof-of-concept DMA rootkit that modifies kernel objects in memory to perform privilege escalation for target processes.

1.1.4 MalloryWorker: Stealthy Computation and Covert Channels using Web Workers

JavaScript execution and UI rendering are typically single-threaded. Consequently, the execution of some scripts can block the display of requested content to the browser screen. Web Workers is an API that enables web applications to spawn background workers in parallel to the main page. Workers support long-lived and computationally expensive operations that might otherwise block the UI. Despite the usefulness of concurrency, users are unaware of worker execution, intent, and impact on system resources. We show that workers can be used to abuse system resources by imple-

menting a unique denial-of-service attack on OS X and resource depletion attack on Android. Further, we show that workers can be used to perform stealthy computation by developing a distributed password cracker, and covert channels exist by exploiting controllable CPU and memory fluctuations. We discuss potential mitigations (i.e., fine-grained control) and implement a lightweight browser extension to increase awareness of worker execution.

1.1.5 KBID: Kerberos Bracelet Identification

The most common method for a user to gain access to a system, service, or resource is to provide a secret, often a password, that verifies her identity and thus authenticates her. Password-based authentication is considered strong only when the password meets certain length and complexity requirements, or when it is combined with other methods in multi-factor authentication. Unfortunately, many authentication systems do not enforce strong passwords due to a number of limitations; for example, the time taken to enter complex passwords. We present an authentication system that addresses these limitations by prompting a user for credentials once and then storing an authentication ticket in a wearable device that we call *Kerberos Bracelet Identification* (KBID).

1.1.6 Applications of Secure Location Sensing in Healthcare

Secure location sensing has the potential to improve healthcare processes regarding security, efficiency, and safety. For example, enforcing close physical proximity to a patient when using a barcode medication administration system (BCMA) can mitigate the consequences of unsafe barcode scanning workarounds. We present Beacon+, a Bluetooth Low Energy (BLE) device that extends the design of Apple's popular iBeacon specification with unspoofable, temporal, and authenticated advertisements. Our prototype Beacon+ design enables secure location sensing applications such as real-time tracking of hospital assets (e.g., infusion pumps). We implement this exact real-time tracking system and use it as a foundation for a novel application that applies location-based restrictions on access control.

1.2 Outline of This Work

We group the concepts of security and privacy challenges for systems, users, and regulators spurred by technological growth in health care under the term health and medical systems. Each chapter of this dissertation presents a unique challenge, or unanswered question, related to health and medical systems. Challenges presented in Chapters 3 through 7 are motivated by concepts and challenges in embedded health and medical systems described in Chapter 2.

Chapter 2 surveys research publications aimed at improving security and privacy in implantable medical devices and health-related body area networks. It also categorizes themes, results, and trends that motivate challenges in subsequent chapters.

Chapter 3 conducts a thorough analysis of iMessage to determine the security of the protocol against a variety of attacks. The result of which implies the difficulty of implementing protocols securely, and exposing security and privacy implications for BYOD and messaging protocols in health and medical systems.

Chapter 4 shows the ability to chain together direct memory transfers, as provided by commodity hardware, is sufficient to perform arbitrary computation. All health and medical systems could be potentially vulnerable to this class of attack.

Chapter 5 uses HTML5 web workers to abuse system resources, perform stealthy computation, and create a covert channel for unauthorized communication. Health and medical systems that allow web browsing and EHR portals could be potentially vulnerable to this type of attack.

Chapter 6 presents an authentication system that addresses the limitation of complex passwords by using a wearable bracelet and authentication module. This system's intent is to be usable and easily integrated into healthcare workflow as to avoid poor security workarounds.

Chapter 7 presents a device called Beacon+ that extends the design of Apple's

popular iBeacon specification with unspoofable, temporal, and authenticated advertisements. Beacon+ enables secure location sensing applications that could improve numerous healthcare processes.

Chapter 2

SoK: Security and Privacy in Implantable Medical Devices and Body Area Networks

The integration of computing devices and health care has changed the landscape of modern medicine. *Implantable medical devices (IMDs)*, or medical devices embedded inside the human body, have made it possible to continuously and automatically manage a number of health conditions, ranging from cardiac arrhythmia to Parkinson's disease. *Body area networks (BANs)*, wireless networks of wearable computing devices, enable remote monitoring of a patient's health status.

In 2001, the estimated number of patients in the United States with an IMD exceeded 25 million;[11] reports from 2005 estimate the number of patients with in-

sulin pumps at 245000.[12,13] IMDs have become pervasive, spurred by the increased energy efficiency and low cost of embedded systems, making it possible to provide real-time monitoring and treatment of patients.[14] Low power system optimizations,[15] ultra-low-power wireless connectivity,[16] and the development of numerous lightweight communication protocols (e.g., on-demand MAC)[17–19] have helped make small-scale sense-actuate systems like IMDs and BANs a reality. Through sensors, these systems can collect a range of physiological values (e.g., heart rate, blood pressure, oxygen saturation, temperature, or neural activity) and can provide appropriate actuation or treatment (e.g., regulate heart rate or halt tremors). On-board radios enable wireless data transfer (or wireless medical telemetry[20]) for monitoring and configuration without sacrificing patient mobility or requiring surgical procedures to physically access the devices.

The need for security and privacy of medical devices has received increasing attention in both the media and the academic community over the last few years—a perhaps telling example is the recent revelation that Vice President Dick Cheney had the wireless telemetry interface on his implanted pacemaker disabled.[21] In the academic community, the seminal work by Halperin et al.,[22] which introduces a class of wireless threats against a commercial *implantable cardiac defibrillator (ICD)*, has been followed by numerous papers researching techniques to improve the security and privacy of medical devices.

Even though the likelihood of *targeted* adversarial attacks on IMDs and BANs

may be debatable, the consequences of an insecure system can be severe. Indeed, Fu and Blum[23] observe that while the hacking of medical devices is a "red herring", poor security design can result in real vulnerabilities. For example, the existence of malware on networked medical devices can result in unreliable data or actuation, impacting both the *integrity* and *availability* of the systems in question. Any private data on the system may be exposed, leading to a breach of *confidentiality*.

Although traditionally there has been little incentive for medical device manufacturers to incorporate security and privacy mechanisms for fear of inhibiting regulatory approval,[24] the FDA has recently called for manufacturers to address cybersecurity issues relevant to medical devices for *the entire life cycle* of the device, from the initial design phase through deployment and end-of-life.[25] Although these calls are in the form of draft guidelines for ensuring appropriate medical device security, there is evidence that the FDA means to use these guidelines as grounds for rejection of premarket medical device submissions.[26]

Ensuring security and privacy in the context of safety-critical systems like IMDs, however, is more nuanced than in the traditional computer science setting. As Halperin et al.[27] observe, the security and privacy goals of IMDs may at times conflict with the safety and utility of these devices. For example, eavesdropping on communications between an IMD and its programmer may reveal a sensitive medical condition, or querying an IMD with an unauthenticated programmer may allow clandestine tracking, both of which compromise the *privacy* of the affected patient.

Unauthenticated communication can lead to denial of service attacks, in which legitimate communication is prevented from reaching the device or the device's battery is needlessly depleted,[22] as well as replay and injection attacks, in which potentially dangerous commands sent to the device can alter the patient's therapy.[22,28,29] On the other hand, using traditional cryptographic mechanisms to ensure secure communication and storage of data can compromise the safety of the patient. If the patient needs treatment outside of his normal health care context (e.g., at the emergency room), it is necessary for health care professionals to have the ability to identify and access the IMD in order to diagnose and treat the patient.

Balancing security, privacy, safety, and utility is a necessity in the health care domain.[24] Multiple academic disciplines (e.g., embedded systems, computer security, and medicine) have independently explored the IMD/BAN problem space. We go beyond related work[27,29,30] by providing a comprehensive overview of security and privacy trends and emerging threats, in order to facilitate uptake by research groups and industry.

Moreover, we provide a more formal adversarial model and classification of threats than the work of Halperin et al.[27] and Zhang et al.[30] By identifying and analyzing popular research trends in this space, we observe that current work may be roughly subdivided into three classes: the security of the wireless telemetry, detection and prevention of software vulnerabilities, and the security of the hardware architecture and sensor interface. Our categorization allows us to easily trace the evolution of

IMD/BAN research, connect current work to related notions from the field of RFID security and privacy, and identify emerging threats in this space.

We identify challenges computer science researchers face in examining the security and privacy of medical devices, including the lack of reproducibility of research results. Access to medical devices is a common problem that limits researchers' ability to validate prior results; food-grade meat as a phantom also complicates reproducibility due to its inaccurate approximation of a human body.[18,31] In addition, we provide clear definitions of IMDs and BANs and describe the relevant communications standards, including clarifying the term *medical device*, which is strictly defined by the FDA. The distinction between a medical device and a device used in the context of health (e.g., FitBit, a popular tool to track physical activity) is a common source of confusion.

In the IMD/BAN space, we need to achieve trustworthy communication, trustworthy software, and trustworthy hardware and sensor interfaces. While the security of the wireless telemetry interface has received much attention in academia, both the threat of software exploits in medical devices and the security and privacy of the sensor interface are areas of research that deserve further attention. Subtle eavesdropping and injection attacks on sensor inputs, such as the work by Foo Kune et al.[32] on *cardiac implantable electrical devices (CIEDs)*, which include pacemakers and defibrillators, and Bagade et al.[33] on compromising the privacy of physiological inputs to key generation mechanisms, are a promising avenue of future work.

2.2 Background and Definitions

Advances in embedded systems[34] and *wireless sensor networks (WSNs)*[35] have made modern IMDs and BANs possible. Current embedded systems trade computing performance and memory resources for energy efficiency and lower costs. Wireless sensor networks link both homogeneous and heterogeneous autonomous devices. WSNs have been used for health care monitoring via the introduction of both wearable and implanted sensor networks,[15,36] giving rise to modern healthcare-related BANs.

2.2.1 Implantable Medical Devices and Body Area Networks

The U.S. FDA has a broad, albeit relatively strict, definition of *medical devices*, which range from tongue depressors to MRI machines. The U.S. Federal Food Drug & Cosmetic Act [37, Section 201(h)] defines a medical device as an instrument, apparatus, machine, or other similar article which is *a)* officially recognized by national registries; *b)* intended for use in the diagnosis, cure, or prevention of a disease; and *c)* intended to affect the structure or function of the body. We emphasize that in order for a device to qualify as a medical device, it must undergo substantial review by the FDA before being released on the commercial market; we use this definition of medical device in this chapter. The FDA also has significant global influence through arrangements with numerous foreign government organizations;[38] therefore devices,

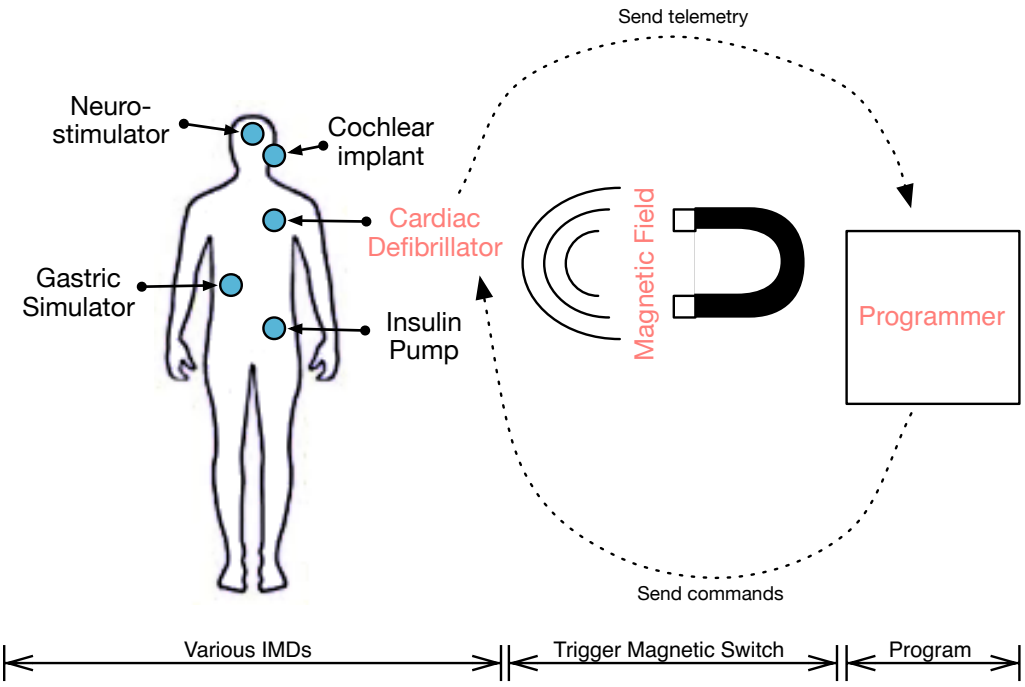


Figure 2.1: Example IMDs and ICD/Programmer communication.

standards, and protocols used in the U.S. are likely to be of interest to other countries as well.

The U.S. Federal Communications Commission (FCC) defines *wireless medical telemetry* in FCC 00-211 [39, Section 3B] and FCC 47 CFR 95.401[20] as the measurement and recording of physiological values via wireless signals. The wireless medical telemetry system is comprised of sensors, radio-based communication, and recording devices. In this chapter, we use the phrase *wireless telemetry*, or simply *telemetry*, to mean radio-based communication, as in the FCC definition; this is distinct from the traditional RFID definition of telemetry, which comprises data collection and transmission.

2.2.1.1 Implantable medical devices

We define an implantable medical device (IMD) as one which is surgically placed inside of a patient's body. Figure 2.1 provides examples of IMDs and *an IMD programmer* (or simply, *programmer*), and shows the high-level communication protocol of an ICD. The programmer in this context is an external device with an interface (usually a *radio frequency (RF)* transceiver) for communicating wirelessly with an IMD and relaying data to a device used by clinicians or other health care providers. An IMD system supports:

- *Analog front end*, the signal conditioning circuitry for application-specific sensing and actuation;
- *Memory and storage*, for storing personal health information and sensed data;
- *Microprocessor*, for executing device-specific software;
- *Telemetry interface*, often radio-based, for transmitting data between the device and a programmer or other sensor/actuator on the patient; and
- *Power management*, for monitoring and managing battery use for increased longevity.

IMDs are resource-constrained, requiring reduced size, weight, low peak power and low duty cycle. Past research uses resource-constrained hardware platforms such as an 8-bit Atmel-AVR and a 16-bit TI MSP430[40] to model IMD configurations. The

TI MSP430F1611 consumes energy at approximately 0.72nJ per clock cycle. Typical IMDs are designed to last 90 months on a single battery with 0.5Ah to 2Ah of battery life.[41] These requirements minimize the impact of invasive surgeries to replace depleted implants. Furthermore, modern IMDs rely on low-power radio communication and network connectivity to provide a remote-monitoring system.[24] The FCC has allocated the 401MHz to 406MHz band for Medical Devices (MedRadio),[42] sometimes called the *Medical Implant Communication Service (MICS)* band. This band is currently used for IMD wireless telemetry.

The MICS band allows for reasonable signal propagation through the human body without interfering with other devices. Additionally, it allows for a greater distance between the patient and external transceiver, unlike previous IMDs (e.g., a pacemaker transmitting at 175kHz, which required a proximity within 5cm[19]).

2.2.1.2 Body area networks

We define a *body area network (BAN)* as a wireless network of heterogeneous computing devices that are wearable. This network enables continuous remote monitoring of patient physiological values in the medical setting. In this work, we are mainly concerned with BANs as they relate to IMDs.

BANs typically include three types of devices: sensors, actuators, and a sink. In Figure 2.2, sensors are placed at various locations on the body, support multiple network topologies, and forward sensed data to a more computationally powerful device

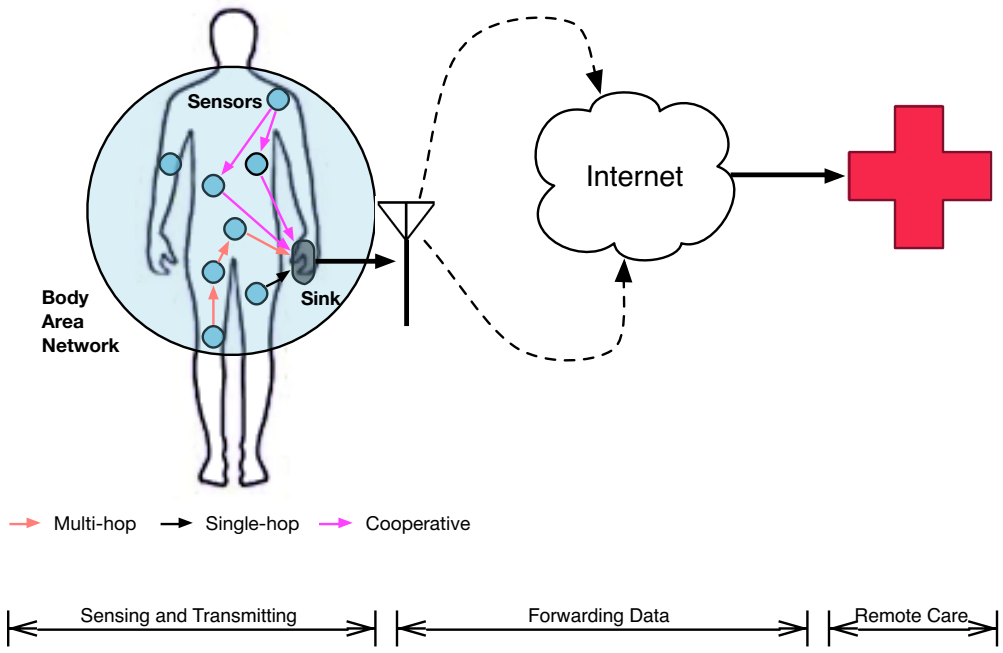


Figure 2.2: Body area network architecture.

(e.g., a smartphone). Although related to wireless sensor networks, BANs exhibit some notable differences[43] with respect to wearability (e.g., size and power), battery availability, and transmission (i.e., the human body is a lossy medium). Moreover, reliability requirements may be stricter than in a typical wireless sensor network, depending on how safety-critical the application.

As we are most interested in BANs as they relate to IMDs, we only give a brief overview of the communication standards for clinical environments.[44] The ISO/IEEE 11073[45] standard spans the entire BAN communication stack, while Health Level 7 (HL7),[46] Integrating the Health Enterprise (IHE)[47] and the recent ASTM F2761 (MDPnP)[48] standard only describe the application layer. While at least some security mechanisms

are mentioned in these standards, most are optional, presumably to ensure interoperability. Foo Kune et al.[44] find that by enabling these security mechanisms in combination with known security protocols, a vast majority of security requirements could be satisfied. The Association for the Advancement of Medical Instrumentation (AAMI) is working on TIR-57, a draft guidance document to start standardizing secure Information Technology (IT) practices for clinical environments; at the time of this writing, a draft was not yet available.

2.3 Security and Privacy in IMDs and BANs

In this section, we first review security and privacy goals for IMDs and BANs. We then present our adversarial model and discuss security threats.

2.3.1 Security and Privacy Goals

We recognize the following security goals for IMDs and BANs, building on the models provided by Halperin et al.,[27] Burleson et al.,[24] and Zhang et al.[30] These properties should hold throughout the entire life cycle of the IMD/BAN devices, including appropriate disposal of explanted devices.

- *Confidentiality*: Data, device information, and device systems should be acces-

sible only to *authorized entities* (i.e., appropriate entities) and these entities should be *authenticated* (i.e., the identity of entities communicating with devices should be verifiable). In particular, data should be kept confidential both in storage and while in transmission.

- *Integrity*: Data, device information, and device systems should not be modifiable by unauthorized entities. The system should also satisfy *data origin authentication*; the source of any received data should be verifiable.

- *Availability*: Data, device information, and device systems should be accessible when requested by authorized entities.

IMDs and BANs should also satisfy the following *privacy* goals; we include criteria from Halperin et al.,[27] Denning et al.,[49] and Kumar et al.[50] for completeness. Although these goals bear some overlap with confidentiality, we include the full list in order to allow for a more comprehensive treatment of privacy (apart from security) in the context of IMDs and BANs. We refer the reader to the work of Avancha et al[51] for a policy-oriented treatment of privacy issues in health-related mobile technology.

- *Device-existence privacy*: Unauthorized entities should not be able to determine that a patient has an IMD/BAN.

- *Device-type privacy*: If device-existence privacy is not possible, unauthorized entities should not be able to determine what type of IMD/BAN is in use.

- *Specific-device ID privacy*: Unauthorized entities should not be able to determine the unique ID of an IMD/BAN sensor.

- *Measurement and log privacy*: Unauthorized entities should not be able to determine private telemetry or access stored data about the patient. The system design phase should include a privacy assessment to determine appropriate policies with respect to data access.

- *Bearer privacy*: Unauthorized entities should not be able to exploit IMD/BAN properties to identify the patient.

- *Tracking*: Unauthorized entities should not be able to leverage the physical layer (e.g., by monitoring analog sensors or matching a radio fingerprint[52–54]) to track or locate a patient.

2.3.2 Adversarial Model

Following the standard approach in computer security literature, adversaries may be distinguished based on their goals, capabilities, and relationship to the system in question. We have the following classification criteria.

1. An adversary is either *active* or *passive*:

    - Passive adversaries are able to eavesdrop on all communication channels in the network, including *side channels*, or unintentional communication

channels.

- Active adversaries are able to read, modify, and inject data over the communication channel.

2. An adversary is either an *external* or *internal* entity with respect to the system. That is, an adversary may either be an *outsider* or an *insider* with a legitimate system role (e.g., manufacturer employees, patient, physician, or hospital administrator).

3. An adversary may be either a *single entity* or a member of a *coordinated group* of entities.

4. An adversary may be *sophisticated*, relying on specialized, custom equipment, or *unsophisticated*, relying only on readily available commercial equipment.

All system components of IMDs and BANs may be used as *attack surfaces*, or points of potential weakness, by an adversary (e.g., any existing sensors, actuators, communication networks, or external programming devices). In addition, the adversary may have the following targets and goals with respect to the specified target.

1. The *patient*: The adversary may wish to obtain private information concerning the patient (e.g., whereabouts, diagnosis, or blackmail-worthy material), or cause physical or psychological harm to the patient.

2. The *device or system manufacturer*: The adversary may wish to engage in corporate espionage or fraud.

3. *System resources*: The adversary may wish to utilize system resources and may be unaware of the type of device or network compromised. That is, the adversary does not knowingly target an IMD/BAN.

2.3.3 Threats

We classify IMD and BAN security and privacy threats found in the literature into the following categories:

- The *telemetry interface*, which is typically wireless. Threats include a passive adversary who eavesdrops on wireless communications and an active adversary who attempts to jam, replay, modify, forge, or drop wireless communications.

- *Software threats*, which consider an adversary that can alter the logic of the system (e.g., through software vulnerabilities) to affect expected operation.

- *Hardware and sensor interface threats.* An adversary may have knowledge of the internal hardware architecture or analog sensors and may use that knowledge to attack the system. Specifically, sensor threats stem from the implicit trust that the system places on those sensor inputs, under the assumption that physical contact with the sensor is necessary to alter the signal. An active at-

tacker, however, may introduce remote interference to sensing in order to affect actuation.

These categories inform our analysis of security and privacy research trends in Section 2.4.

2.4 Medical Device Security and Privacy Trends

We follow the broad categorization of IMD and BAN security and privacy threats given in Section 2.3.3 in order to analyze research trends in the literature. That is, we group research according to the relevant attack surface: the telemetry interface, software, and hardware/sensor inputs. We give an explicit categorization of relevant research with respect to security threats and goals in Table 2.1. Due to the large amount of work on the wireless telemetry threats, we separate the wireless threats into subclasses. An overview of current research, grouped thematically and by publication year, is given in Figure 2.3.

As Figure 2.3 indicates, the vast majority of results in the literature focus on threats to the telemetry interface, while a limited number of papers consider software threats. Since very few papers deal with threats to the sensor interface, we defer discussion of this emerging threat to Section 2.5.3.

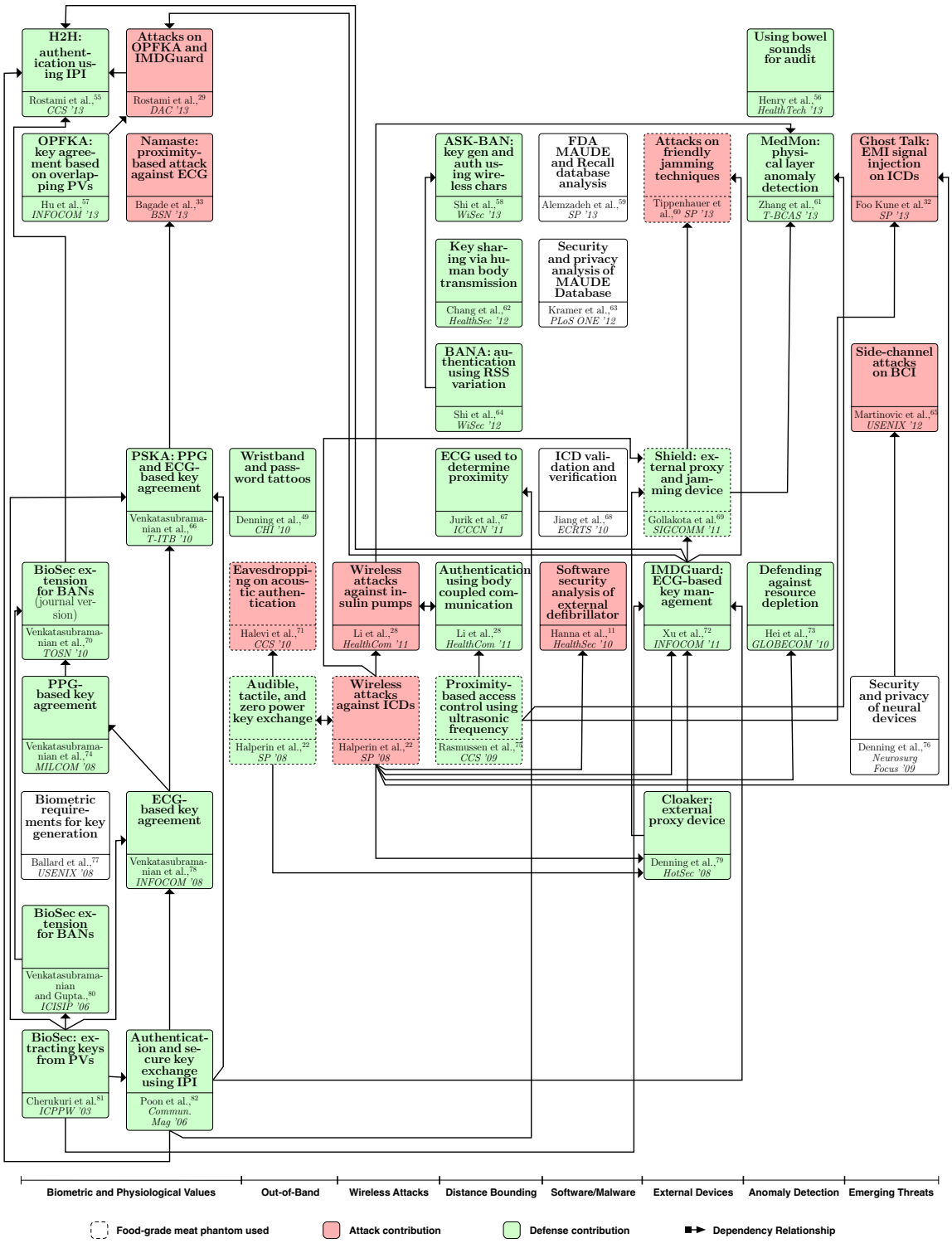


Figure 2.3: Trends in Security and Privacy Research on IMDs/BANs.

		Goal Compromised by Indicated Threat					
Threat	**Attacks**	**Confident.**	**Integrity**	**Avail.**	**Privacy**	**Safety**	**Defenses**
Wireless eavesdropping	[22,28,60]	✓			✓		[22,28,55,64,79–81,58,66,69,70,72,75,83,61,74,78,82,84,57,62,85,86]
Wireless modification	[22,28,29]		✓	✓		✓	[22,28,55,64,80,81,58,69,70,72,75,79,57,61,66,74,78,82–85,62,86]
Wireless replay	[22,28]		✓	✓		✓	[22,28,55,64,80,81,58,69,70,72,75,79,61,66,78,82–84,57,62,74,85,86]
Wireless jamming				✓		✓	[72,79]
Analog sensor injection	[32]		✓			✓	[32]
Battery depletion	[22]			✓		✓	[22,69,73,79]
Protocol Design Flaws	[22,28,29,33,60,71]	✓	✓	✓	✓	✓	Not Applicable
Software Flaws	[87]	✓	✓	✓	✓	✓	[68,87]
Side channels	[33,65,71]	✓	✓	✓	✓	✓	[65]

Table 2.1: IMD and BAN security and privacy threats and defenses

2.4.1 Securing the Wireless Telemetry Interface

Halperin et al.[22] introduce a class of wireless threats against a commercial ICD; since then, attacks on the telemetry interface of IMDs have received a large amount of attention.[28,88,89] At the physical layer, Halperin et al.,[22] targeting an ICD, and Li et al.,[28] targeting an insulin pump system, develop passive and active attacks against their respective device using an off-the-shelf software defined radio (SDR) platform. In the devices and programmers analyzed, the communication links do not use an authenticated channel and transmit unencrypted data without freshness checks, thereby allowing eavesdropping, replay,[22] and injection attacks.[28]

Unsurprisingly, many authentication techniques have been proposed to secure the wireless telemetry of IMDs and BANS, including the use of biometrics, distance-bounding authentication, out-of-band authentication, external devices, and anomaly detection. We explore each of these areas individually below.

2.4.1.1 Biometrics

Popular techniques for key generation and key agreement in IMDs/BANs include the use of biometrics, or *physiological values (PVs)*.[55,57,66,67,70,74,78,80–82,84] *Electrocardiograms (ECGs)* are a common choice as a source of key material in these protocols, although other PVs such as heart rate, blood glucose, blood pressure, and temperature have been proposed.[81]

The choice to use ECGs is motivated by a well-cited paper by Poon et al.,[82] which asserts that the time between heartbeats, or *interpulse interval (IPI)*, has a high level of randomness. IPI has the additional benefit that it can be measured anywhere on the body and many IMDs in use today can measure IPI without modification.

A typical approach to PV-based key agreement between an IMD and programmer, for example, involves both devices taking a measurement of the chosen PV. This measured PV is used to generate a cryptographic key that is agreed upon by both devices, which is then used to establish an authenticated channel. The basic assumption is that physical contact (or at least physical proximity) with the patient is required in order to precisely measure the chosen PV.

Security analyses of these protocols have been mostly ad hoc in nature, however, and in general more comprehensive assessments are required. For example, Rostami et al.[29] demonstrate simple, but damaging attacks against OPFKA[57] and IMDGuard,[72] which we discuss in Section 2.4.1.4.

Chang et al.[62] also explore the use of IPI, drawing attention to the issue of noise in real-world measurements. Later work by Rostami et al.[55] presents a more robust IPI-based authentication protocol, which unlike previous work, takes into account both the impact of measurement noise and provide a more rigorous security analysis. We discuss the subtleties and potential difficulties of using IPI as part of a key agreement protocol in more detail in Section 2.5.2 and Section 2.5.3.

2.4.1.2 Distance-Bounding Protocols

Distance bounding[90] is a technique that establishes physical distance between two entities by timing the delay of sent and received transmissions. This distance bound can be computed over various signals such as RF or ultrasonic sound (which is an acoustic signal above 20kHz). A number of IMD/BAN access control and authentication protocols use distance bounding.[28,58,62,64,75] However, distance bounding by itself provides for only weak authentication, in which physical proximity between devices is established but identity and authorization are not, thereby requiring the use of additional authentication techniques.

A typical distance-bounding protocol between a programmer and IMD, for ex-

ample, involves the programmer proving to the IMD that it is physically close (e.g., within 3cm). Rasmussen et al.[75] use ultrasonic sound signals to compute the distance bound of a programmer and IMD, since it is impossible for an attacker to send audio data that propagates faster than the speed of sound. Shi et al.[58,64] use *received signal strength (RSS)* variation to differentiate BAN devices on the same body from external signals (i.e., attacker transmissions). This technique relies on the observation that the RSS variation between two BAN devices on the same body is more stable than the RSS between an on-body device and an external device. Jurik et al.[67] make use of ECG signals to establish the continued proximity of an authenticated mobile device to a user.

Distance bounds are also computed over *body-coupled communication (BCC)*. BCC uses the human body as a transmission medium, requiring physical proximity to the patient in order to communicate. Li et al.[28] introduce wireless attacks against BCC and find that both passive and active attacks are mitigated for distances greater than 0.5m. Chang et al.[62] inject artificial signals through the patient's body to authenticate BAN devices on the same body. These signals, however, only achieve an estimated 0.469 to 5.429 bits per hour, making this technique impractical.

In the related field of RFID, system implementations have inaccurately assumed distance-bounding guarantees as a result of short read ranges (e.g., 10cm). Kfir et al.[91] introduce a relay attack in which two coordinated adversaries fool an RFID reader into believing that the RFID tag is nearby. Relay attacks can be mitigated with context-

aware communication,[92] a method which requires the user to perform an uncommon, but easily repeatable movement in order to be authenticated. The applicability of this defense to IMDs is debatable, however, because a patient may not be able to authenticate in the event of a medical emergency.

Cremers et al.[93] provide a classification of distance-bounding attacks that assumes weak authentication, suggesting additional evaluation is required before such protocols are used in the medical setting; the adversarial capabilities necessary to launch these attacks are included in our model. Cremers et al. use the terminology *verifier* and *prover* to describe the participants in distance-bounding protocols; the verifier establishes physical proximity to the prover. The attacks consider various adversarial capabilities for falsifying physical proximity to the prover. Specifically, the adversary may modify transmissions between a verifier and prover. He may introduce his own dishonest prover, or he may collude with other dishonest entities. Lastly, he may also exploit honest provers (e.g., by first allowing the prover to establish physical proximity, then jamming subsequent prover transmissions and authenticating in the prover's stead).

2.4.1.3 Out-of-Band (OOB) Authentication

OOB techniques make use of auxiliary channels, such as audio, visual, and tactile, that are outside the established data communication channel.[22,49,83,94] Using auxiliary channels for authentication obviates the need for trusted third parties and key

pre-distribution schemes. A common assumption in these schemes is that the chosen out-of-band channel is resistant to eavesdropping attacks.

Halperin et al.[22] propose an OOB authentication scheme that uses a low-frequency audio channel. The basic idea is that the IMD uses a zero-power RFID device to generate a random key and transmit it over the audio channel. The patient is alerted when a key exchange occurs through vibrations produced by a piezo element connected to the RFID device. The programmer, at a distance of no more than 0.6m to 0.9m,[71] listens for the key and then establishes a secure authenticated channel with the IMD.

Halevi et al.[71] examine a passive adversary with the ability to deploy (or otherwise make use of) a general-purpose microphone (e.g., PC microphone) in the vicinity of the IMD/programmer communication. Halevi et al. show that although the measured piezo sound accuracy varies with distance, the average key retrieval correctness at 0.9m, computed for multiple supervised methods, is as high as 99.88%. This contradicts Halperin et al.'s[22] earlier experimental result, which indicates the audio channel is resistant to eavesdropping.

Alternatively, Denning et al.[49] and Li et al.[83] opt for visual OOB authentication. Denning et al. propose the use of ultra-violet or visible tattoos to record permanent IMD keys. This mechanism allows emergency authentication, but does not allow for key revocation and may suffer from usability concerns.[49] Li et al.[83] require the users to visually inspect simultaneous LED blinking patterns in order to achieve

authentication in BANs. The usability of this scheme is unclear and it is unlikely to be appropriate for emergency scenarios, so its applicability to IMDs is limited.

2.4.1.4 External Wearable Devices

A unique approach to securing IMD/BAN telemetry makes use of external devices worn by the patient. The basic idea is that this external device mediates communication with the IMD, thereby providing both confidentiality for transmitted data and protection against unauthenticated communication. One concern with the use of such devices is their acceptability to the patient, however. Denning et al.[49] treat this issue in some detail and study the usability of several possible authentication methods, including external devices and password tattoos.

Denning et al.[79] propose an external device, called the *cloaker*, that proxies authorized communication to the IMD. If the cloaker is absent, the IMD communicates openly (e.g., in case of a medical emergency, the cloaker *fails open*). A malicious programmer can exploit this fail-open behavior by selectively jamming the cloaker or otherwise convincing the IMD of the cloaker's absence, so Denning et al. suggest additional mitigation techniques to prevent such an attacker from communicating with the IMD.

Gollakota et al.[69] and Xu et al.[72] use *friendly jamming* to protect IMD communication, which uses jamming constructively to prevent unauthorized communication. IMDGuard[72] employs an external wearable device, called the Guardian, to enable

access control and confidential data transmissions. The Guardian first authenticates the programmer and then uses an ECG-based key agreement mechanism to authenticate itself to the IMD. Temporary keys can then be issued to allow a secure channel between the programmer and the IMD. In the event that an attacker jams the messages from the Guardian device to the IMD, the Guardian initiates an active defense by jamming all IMD transmissions. However, IMDGuard has the disadvantage of requiring modifications to the IMD itself (which is difficult in practice with respect to already-deployed devices) and the suggested ECG-based key agreement scheme suffers from security flaws. Rostami et al.[29] show a simple man-in-the-middle attack that reduces the effective key length from 129 bits to 86 bits. This attack takes advantage of a protocol flaw in the second round of reconciliation (in which the two parties verify they know the same key), which can be spoofed to reveal one bit per block.

The *shield*[69] works by listening for and jamming all IMD transmissions and unauthorized commands. Given the shield's proximity and jamming power, the assumption is that only the shield can cancel out its own jamming signal and decode IMD transmissions. This design mitigates both passive and active wireless attacks, but the security of the system relies on the assumption that an attacker whose distance from the IMD is greater than the distance between the IMD and the shield will be unable to recover IMD transmissions, even if the attacker is equipped with *multiple input and multiple output (MIMO)*-systems and directional antennas. Tippenhauer et al.[60]

challenge this assumption, however, and show that MIMO-based attacks are possible in the presence of an adversary with two receiving antennas from distances of up to 3m.

2.4.1.5 Anomaly Detection

Anomaly detection attempts to automatically identify resource depletion and malicious communication, as well as distinguish between safety and security events.[56,61,73] This is generally achieved by observing patterns over time, such as physiological changes or IMD access patterns (e.g., programmer commands, date, or location).

Hei et al.[73] obtain and use normal IMD access patterns as training data for their supervised learning-based scheme. The resultant classification is used to identify anomalous IMD access in real time. That is, Hei et al.'s method tries to detect abnormal access attempts and block such authentication from proceeding, *before* any expensive computations take place. In this way, the IMD is protected against denial of service attacks that deplete the system's resources. This scheme is designed for non-emergency settings, however, and Hei et al. recommend that either the IMD automatically detect emergency conditions and fail open, or that hospitals have access to a master device key. The feasibility and security provided by these two approaches is not considered.

Another anomaly detection approach makes use of audits; Henry et al.'s scheme[56] observes correlated physiological changes when an insulin bolus is administered by

tracking acoustic bowel sounds. These observations are recorded as an audit log for retroactive verifiability of intended system execution. While useful, a limitation of passive anomaly detection is that such schemes do not provide medical device integrity, and so need to be used in conjunction with another mechanism that protects communications.

At the physical layer, wireless transmissions from an attacker are likely to deviate in physical characteristics from legitimate programmer transmissions. Zhang et al.[61] propose a medical security monitor, MedMon, which is an external device that detects anomalous transmissions by examining physical characteristics of the transmitted signal; such characteristics include received signal strength, time of arrival, differential time of arrival, and angle of arrival. When an anomalous transmission is detected, MedMon can initiate either a passive defense (e.g., by alerting the patient) or an active defense (e.g., by blocking the transmissions from reaching the medical device).

The characteristics of the device used for anomaly detection (and any associated audit logs) have important implications for the overall security of the system. Suggested anomaly detection implementations make use of dedicated devices, such as analog sensor systems,[56] or extend the functionality of personal devices, such as smartphones.[61,73] Offloading heavy computation to another device like a smartphone might improve the IMD's battery life, but significantly increases the attack surface, as malware on mobile devices is common.[95] Moreover, regulatory barriers for medical devices may make this approach difficult. Additional challenges related to the use of

mobile devices and health-related BANs are surveyed by Avancha et al.[51]

2.4.2 Software Threats

Software running on medical devices spans a wide range of complexity. An increasing number of medical devices are reliant on digital circuits controlled by software, rather than analog circuits. Faris[96] notes that in 2006, a major milestone was crossed when over half of deployed medical devices contained software. So far there has been a lack of detailed analysis of IMD software. However, there have been efforts to verify proper functionality by simulating an artificial heart to interface with cardiac pacemakers.[68,97] Although these testing methods are not directly tailored to security, the tests reduce software bugs and may therefore reduce possible software vulnerabilities.

Devices communicating over a BAN, in addition to their application code, have to include a telemetry interface that increases both the amount of code and the number of possible bugs. It is not surprising, then, that software is one of the main reasons for FDA recalls of computer-related issues.[59] Sandler et al.[98] report that in 2010, the FDA issued 23 recalls of defective devices, six of which were likely caused by software defects. Alemzadeh et al.[59] report that the percentage of computer-related recalls between 2006 and 2011 was between 30% to 40%. In this study, software defects are found to be the cause of 33% of computer-related class I recalls (reasonable chance of patient harm), 66% of class II recalls (temporary or reversible adverse effects), and 75% of class III recalls (non-compliant, but unlikely to cause harm).

Bugs in medical devices have been a cause of over 500 recalls recorded between 2009 and 2011 by the FDA.[63] While there exists no method to extrapolate from the reported bugs to those existing in deployed devices, the number reported is most likely only a lower bound. Fu reports that failures in medical device software often result from a failure to apply known system engineering techniques,[99] indicating that the problem is partially solvable today.

Moreover, the presence of a telemetry interface on the device may expose software bugs to a remote attacker. Evidence of the brittleness of software implementations is apparent when investigating security vulnerabilities, including those in proprietary firmware. Hanna et al.[87] perform the first public software security analysis of an *automatic external defibrillator (AED)*. By reverse engineering the device, the authors successfully target three software packages responsible for programming device parameters, collecting post-cardiac device data, and updating the AED. The authors locate four vulnerabilities, one of which enables arbitrary code execution on the device.

The need for secure coding practices for safety-critical devices is clear. However, closed source for medical devices make it challenging to run a static analyzer on the source code, let alone obtain the firmware. With proprietary protocols and the special MICS band used on the wireless telemetry interface, traditional fuzzing tools such as Peach Fuzzer[100] have not developed modules appropriate for testing medical devices.

A related security vulnerability is the existence of malware on medical devices.

Regardless of whether the intent of the attacker is to compromise a medical device, malware can significantly impact the performance and reliability of safety-critical devices such as IMDs.[23]

2.5 Research Challenges and Emerging Threats

In this section, we identify and address challenges computer science researchers face in examining the security and privacy of medical devices and discuss promising areas for future work. In particular, we discuss common problems, identifying partial solutions and highlighting areas where further work is needed. A particularly difficult issue is the lack of reproducibility of research results in this field; given the safety-critical nature of IMDs and some BANs, it is critical that proposed attacks and defenses be thoroughly and independently evaluated in order to accurately assess risk of the attack and efficacy of the defense. A second area of concern, which we discussed briefly in Section 2.4.1, is the use of physiological values to secure IMDs/BANs. The evaluations in the literature are limited in scope, partially because of the lack of availability of appropriate data sets for use by researchers and partially because the focus has been on protocol design rather than on a rigorous assessment of the use of biometrics for cryptographic key establishment.

We first address issues related to reproducibility in Section 2.5.1, before moving

to a discussion of the use of physiological values in Section 2.5.2.

2.5.1 Reproducibility challenges

Lack of access to devices is a common problem; access to medical devices is either non-existent or limited to older, end-of-life models that have been received from patients, relatives, or physicians. The ICD that Halperin et al.[22] study, for example, is a model introduced to the market five years earlier. Without access to the devices themselves, researchers are necessarily limited in their ability to analyze potential attacks and defenses; often device hardware configurations are not public knowledge. Research results from groups that have managed to acquire and study particular IMDs are not likely to be validated by others, if only because of lack of equipment. While there have been some efforts to provide access to medical devices,[101] direct access to devices from manufacturers by the security research community appears to be limited at present.

A second issue in computer security and privacy experiments on medical devices is the use of food-grade meat as a *phantom*, or human tissue simulator.[22,60,69] As Clark and Fu[31] observe, this method does not lead to reproducible experiments, possibly due to the introduction of uncontrolled variables that can affect the impedance of the tissue or propagation of signals in the phantom. Instead, researchers should use a calibrated saline solution at 1.8g/L at 21°C [102, Table 10, p. 30] with electrodes to inject the appropriate simulated physiological signals. The complete design is de-

scribed in the ANSI/AAMI PC69:2007 standard [102, Annex G]; this is the accepted standard for electromagnetic compatibility of medical devices by researchers, device manufacturers, and regulators.

2.5.2 Physiological values as an entropy source

As mentioned in Section 2.4.1.1, the use of physiological values as a building block for security and privacy mechanisms is widespread in the literature. In particular, much research relies on the use of ECGs for security and privacy mechanisms. ECG measurements have been suggested for use in authentication,[55] key establishment,[66,72,82] and proximity detection[67] protocols (i.e., determining if one or more devices are in physical contact with the same body). Several systems have devices generate a shared secret key by reading the ECG signal through physical contact with the same person.[33,57,66,70,72,78,85]

Most of these ECG-based mechanisms rely on the reported randomness of the IPI, or the amount of time between individual heartbeats;[55,72] Rostami et al.[29,55] suggest that sufficient entropy may be extracted from the least significant bits of properly quantized IPIs. There are some inconsistencies in the literature with respect to the quality of randomness it is possible to extract,[75,77,81] however, and in studying this issue, researchers have been limited by a lack of sufficient real-world data. In particular, it is important to understand the impact of confounding factors such as health and age on the amount of entropy in IPI, in order to ensure that appropriate

protocol parameters are chosen for entropy extraction.

In addition, Chang et al.[62] draw attention to the fact that the feasibility of these schemes relies on the ability of two devices to measure (and agree on) IPI in the presence of noise. Therefore, realizing such schemes may be more difficult using real-world data, rather than data collected in controlled environments (as measured by physicians with advanced medical equipment). Chang et al.'s results are indicative that measurement noise must be taken into account; later work by Rostami et al.[55] address this concern by taking into account and optimizing for these error rates.

Most evaluations have relied on an aggregation of heart rate databases from the MIT PhysioNet portal,[103] which provides access to a large number of waveforms (collected by clinicians) ranging from healthy sinus rhythms to irregular heartbeat rhythms, or *arrhythmias*. Many suggested protocols are evaluated using either unspecified databases[33,57,66,72,78,85] or arrhythmia databases.[55,70,86,104] To extract random bits for a given record, the mean and standard deviation of the record are used to first quantize the bits, with a subset of the least significant bits treated as random. For example, Rostami et al.[55] quantize the IPI data into 8-bit representations and take the four least significant bits as random; the amount of entropy is estimated empirically using the classical definition of Shannon entropy (i.e., average entropy). A statistical battery of tests is then applied to the extracted bits—typically the (basic) subset of the NIST test suite[105] appropriate for the amount of data available.

Following the state of the art,[106,107] the assessment of a *true random number*

generator (TRNG) for cryptographic purposes requires

a) an assessment of the quality of the entropy source itself (and a justification that the physical process being measured is random); *b)* an analysis of the efficiency and robustness of the extraction method (and the impact of the extraction method on the statistical properties of the TRNG); and *c)* cryptanalysis in the suggested use case (e.g., if an adversary can observe the entropy source or has an advantage in guessing future bits, this is not good for cryptographic use).

In particular, statistical analysis of the output of a TRNG, such as testing the output using the NIST test suites, is not sufficient to determine suitability for use in key agreement. The statistical properties of the physical phenomena need to be well-understood; properly quantizing the data and extracting bits that are close to uniform requires an accurate characterization of the distribution. For example, in the case of IPI, if the suggested methods for bit extraction do not ensure that the distribution characteristics used at time of authentication are accurate, the resulting bits may exhibit bias. We discuss the issue of observability of the IPI entropy source in more detail in the next section.

2.5.3 Emerging threats: sensors, remote attacks, and privacy

The traditional assumption with respect to IMDs and BANs is that many physiological signals stay within a patient's body, limiting the exfiltration of data and the possibility for signal injection attacks. Recent studies, however, show that both are possible.

To date, the design constraints of IMDs have carefully dealt with the possibility of *accidental* electromagnetic interference, but do not consider the possibility of an active attacker. Recent work by Foo Kune et al.[32] shows that intentional interference at a CIED sensor interface is possible. By injecting a signal that mimics a cardiac waveform, Foo Kune et al. show that it is theoretically possible to alter the therapy delivered by the CIED, although the current range of this attack is very limited (on the order of a few centimeters). Reliance on sensor readings to achieve accurate and timely actuation, combined with increasingly sophisticated attacks, highlights the need to carefully consider adversarial capabilities and how best to achieve trustworthy systems.

Similarly, if the assumption that certain physiological signals stay within the human body is incorrect, both the security and privacy of schemes may be affected. For example, the use of physiological values as a source of entropy in key agreement schemes relies heavily on the assumption that it is not feasible for an adversary to

observe the given biometric. A standard assumption in current literature is that the adversary cannot make physical contact with the target patient. In this sense, protocols that make use of physiological values to generate a shared key can be viewed as body-coupled communication protocols, whereby the key is transmitted via the human body. Although the assumption that an adversary does not have physical contact has merit in practice, we remark that this adversarial model neglects subtle classes of attacks by people known to the victim; ideally, new technologies should not enable "perfect crime" scenarios, even for the most sophisticated of attackers. As more and more people become active participants in (potentially insecure) BANs, moreover, it may be possible for a person close to the victim (i.e., with physical contact) to inadvertently aid a remote attacker (e.g., by leaking patient biometrics or performing signal injection attacks on sensors/wireless telemetry).

Remote attackers are also a concern today, especially with respect to observing physiological values assumed to be secret. Rostami et al.[55] and Chang et al.[62] both recognize the need to consider remote sensing of IPI. Rostami et al. attempt to extract IPI from video footage of the target, following work by Poh et al.[108] on the correlation between color fluctuations and IPI. Although Rostami et al. fail to replicate these results, other recent work in this area[109,110] indicates that such attacks deserve further attention.

As a final remark, recent results in Bagade et al.[33] show that the ECG data of one person may be observable from another person's physiological signals, if the two

are in physical contact. That is, if two individuals touch, the ECG of one person is coupled to the EEG of the other person. We conclude that while the use of ECG (and other physiological values) as a security mechanism appears to hold some promise, cryptanalysis and entropy assessments need to be undertaken more rigorously.

A related area of research is the study of *neurostimulators*, which are IMDs designed to send electrical pulses to the nervous system, including the brain. These devices are used to treat conditions such as epilepsy, Parkinson's, and obsessive compulsive disorder, with ongoing human trials exploring their efficacy in treating severe depression. Very little computer security and privacy research has been completed on these devices, and as the technology progresses, the need for further work in this area becomes more pressing. Denning et al.[76] give a brief overview of potential security and privacy implications with respect to neurostimulators, but concrete results in this area are lacking. A related question is explored by Martinovic et al.:[65] the authors' side channel attacks in the context of *brain-computer interfaces (BCIs)*, which measure and respond dynamically to a user's brain activities, thereby allowing communication without words or gestures. Although the study is preliminary in nature, Martinovic et al.'s results support the hypothesis that personal information, such as passwords and whether or not a particular person is known to the target, may unintentionally leak through BCI use.

2.6 Concluding Remarks

In this chapter, we have given a cohesive narrative of security and privacy research in IMDs and BANs, analyzing current and emerging research trends: namely the security of the IMD/BAN telemetry and sensor interfaces and the need for trustworthy software. Our analysis in Section 2.4.1 shows that much attention has been paid to securing the telemetry interface and many useful approaches have been developed.

We have identified several areas for future work, such as the need for a more rigorous assessment of the use of physiological values as a source of entropy for cryptographic keys. As mentioned in Section 2.4.2, the increasing complexity of software in IMDs and the history of FDA software-related recalls highlights the need for future work ensuring the trustworthiness of IMD and BAN software.

Finally, as discussed in Section 2.5.3, the possibility of EMI attacks on the sensor interface and eavesdropping on physiological signals formerly thought to be private is indicative of the need for a more nuanced approach to security and privacy research for medical devices. Computing devices that interface with the brain are becoming more advanced and more popular, both in the entertainment (in the form of BCI-integrated gaming) and health care industries (in the form of neurostimulators). The ability to record and analyze brainwaves in real time using implanted computing devices that alter the brain's functionality has far-reaching implications for security and privacy, moving well beyond the traditional treatment of these topics in computer security.

Chapter 3

Dancing on the Lip of the Volcano: Chosen Ciphertext Attacks on Apple iMessage

The past several years have seen widespread adoption of end-to-end encrypted text messaging protocols. In this work we focus on one of the most popular such protocols: Apple's iMessage. Introduced in 2011, iMessage is an end-to-end encrypted text messaging system that supports both iOS and OS X devices. While Apple does not provide up-to-date statistics on iMessage usage, in February 2016 an Apple executive noted that the system had a peak transmission rate of more then 200,000 messages per second, across 1 billion deployed devices.[111]

The broad adoption of iMessage has been controversial, particularly within the law

enforcement and national security communities. In 2013, the U.S. Drug Enforcement Agency deemed iMessage "a challenge for DEA intercept",[112] while in 2015 the U.S. Department of Justice accused Apple of thwarting an investigation by refusing to turn over iMessage plaintext.[113] iMessage has been at the center of a months-long debate initiated by U.S. and overseas officials over the implementation of "exceptional access" mechanisms in end-to-end encrypted communication systems,[114–116] and some national ISPs have temporarily blocked the protocol.[117] Throughout this controversy, Apple has consistently maintained that iMessage encryption is end-to-end and that even Apple cannot recover the plaintext for messages transmitted through its servers.[118]

Given iMessage's large installed base and the high stakes riding on its confidentiality, one might expect iMessage to have received critical attention from the research community. Surprisingly, there has been very little analysis of the system, in large part due to the fact that Apple has declined to publish the details of iMessage's encryption protocol. In this chapter we aim to remedy this situation. Specifically, we attempt to answer the following question: how secure is Apple iMessage?

Our contributions. In this work we analyze the iMessage protocol and identify several weaknesses that an attacker may use to decrypt iMessages and attachments. While these flaws do not render iMessage completely insecure, some flaws reduce the level of security to that of the TLS encryption used to secure communications between end-user devices and Apple's servers. This finding is surprising given the protection

claims advertised by Apple.[118] Moreover, we determine that the flaws we detect in iMessage may have implications for other aspects of Apple's ecosystem, as we discuss below.

To perform our analysis, we derived a specification for iMessage by conducting a partial black-box reverse engineering of the protocol as implemented on multiple iOS and OS X devices. Our efforts extend a high-level protocol overview published by Apple[119] and two existing partial reverse-engineering efforts.[120,121] Armed with a protocol specification, we conducted manual cryptanalysis of the system. Specifically, we tried to determine the system's resilience to both back-end infrastructure attacks and more restricted attacks that subvert only client-local networks.

Our analysis uncovered several previously unreported vulnerabilities in the iMessage protocol. Most significantly, we identified a *practical* adaptive chosen-ciphertext attack on the iMessage encryption mechanism that allows us to retrospectively decrypt certain iMessage payloads and attachments, provided that a single Sender or Recipient device is online. To validate this finding, we implemented a proof of concept exploit against our own test devices and show that the attack can be conducted remotely (and silently) against any party with an online device. This exploit is non-trivial, and required us to develop novel exploit techniques, including a new chosen ciphertext attack that operates against ciphertexts containing `gzip` compressed data. We refer to this technique as a ***`gzip`*** *format oracle* attack, and we believe it may have applications to other encryption protocols. We discuss the details of this attack in

§3.5.

We also demonstrate weaknesses in the device registration and key distribution mechanisms of iMessage. One weakness we exploit has been identified by the reverse engineering efforts in ,[120] while another is novel. As they are not the main result of this work, we include them in Appendix 3.10 for completeness.

Overall, our determination is that while iMessage's end-to-end encryption protocol is an improvement over systems that use encryption on network traffic only (*e.g.,* Google Hangouts), messages sent through iMessage may not be secure against sophisticated adversaries. Our results show that an attacker who obtains iMessage ciphertexts can, at least for some types of messages, *retrospectively* decrypt traffic. Because Apple stores encrypted, undelivered messages on its servers and retains them for up to 30 days, such messages are vulnerable to any party who can obtain access to this infrastructure, *e.g.,* via court order,[113] or by compromising Apple's globally-distributed server infrastructure.[122] Similarly, an attacker who can intercept TLS using a stolen certificate may be able to intercept iMessages on certain versions of iOS and Mac OS X that do not employ certificate pinning on Apple Push Network Services (APNs) connections.

Given the wide deployment of iMessage, and the attention paid to iMessage by national governments, these threats do not seem unrealistic. Fortunately, the vulnerabilities we discovered in iMessage are relatively straightforward to repair. In the final section of this chapter, we offer a set of mitigations that will restore strong cryp-

tographic security to the iMessage protocol. Some of these are included in iOS 9.3 and Mac OS X 10.11.4, which shipped in March 2016.

Other uses of the iMessage encryption protocol. While our work primarily considers the iMessage instant messaging system, we note that the vulnerabilities identified here go beyond iMessage. Apple documentation notes that Apple's "Handoff" service, which transmits personal data between Apple devices over Bluetooth Low Energy, encrypts messages "in a similar fashion to iMessage".[119] This raises the possibility that our attacks on iMessage encryption may also affect intra-device communication channels used between Apple devices. Attacks on this channel are particularly concerning because these functions are turned on by default in many new Apple devices. We did not investigate these attack vectors in this work but subsequent discussions with Apple have confirmed that Apple uses the same encryption implementation to secure both iMessage and intra-device communications. Thus, securing these channels is one side effect of the mitigations we propose in §3.7.

3.1.1 Responsible disclosure

In November 2015 we delivered to Apple a summary of the results in this chapter. Apple acknowledged the vulnerability in §3.5 and has initiated substantial repairs to the iMessage system. These repairs include: enforcing certificate pinning across all channels used by iMessage,[1] removing compression from the iMessage composition

[1]This feature was added to OS X 10.11 in December, as a result of our notification.

(for attachment messages), and developing a fix based on our proposed "duplicate ciphertext detection" mitigation (see §3.7). Apple has also made changes to the use of iMessage in inter-device communications such as Handoff, although the company has declined to share the details with us. The repairs are included in iOS 9.3 and OS X 10.11.4, which shipped in March 2016.

3.1.2 Attack Model

Our attacks in §3.5 require the ability to obtain iMessage ciphertexts sent to or received by a client. Because Apple Push Network Services (APNs) uses TLS to transmit encrypted messages to Apple's back-end servers, exploiting iMessage requires either access to data from Apple's servers or a forged TLS certificate. We stress that while this is a strong assumption, it is the appropriate threat model for considering end-to-end encrypted protocols.

A more interesting objection to this threat model is the perception that iMessage might be too weak to satisfy it. For example, in 2013 Raynal *et al.* pointed out a simple attack on Apple's key distribution that enables a TLS MITM attacker to replace the public key of a recipient with an attacker-chosen key.[120] One finding of this work is that as of December 2015 such attacks have been entirely mitigated by Apple through the addition of certificate pinning on key server connections (see Appendix 3.10). More fundamentally, however, such attacks are *prospective* – in the sense that they require the attacker to target a particular individual before the indi-

vidual begins communicating. By contrast, the attacks we describe in this chapter are *retrospective*. They can be run against any stored message content, at any point subsequent to communication, provided that one target device remains online. Moreover, unlike previous attacks which require access to the target's local network, our attacks may be run remotely through Apple's infrastructure.

3.2 The iMessage Protocol

To obtain the full iMessage specification, we began with the security overview provided by Apple, as well as a detailed previous software reverse-engineering efforts conducted by Raynal[120] and others.[121] While these previous results provide some details of the protocol, they omit key details of the encryption mechanism, as well as the complete key registration and notification mechanisms. We conducted additional black-box reverse engineering efforts to recover these elements. Specifically, we analyzed and modified protocol exchanges to and from several jailbroken and non-jailbroken Apple devices.[2] In conformity to Apple's terms of service, we did not perform any software decompilation.

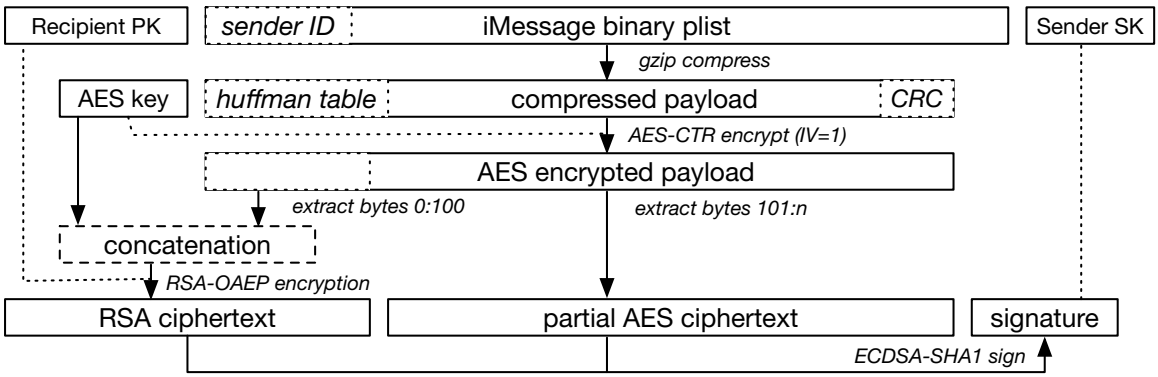


Figure 3.1: The iMessage encryption mechanism. From the top, each iMessage is encoded in a binary `plist` key/value structure. The structure encodes a list of Sender and Recipient account identifiers, as well as the message contents. This payload is subsequently `gzip` compressed, and encrypted under a freshly-generated 128-bit message key using AES in CTR-mode. The AES key and the first 101 bytes of the AES ciphertext are concatenated and are encrypted to each Recipient's public key using RSA-OAEP. The remaining bytes of the AES ciphertext are concatenated to the RSA ciphertext and the result is signed using ECDSA under the Sender's registered signing key.

3.2.1 System overview

iMessage clients. iMessage clients comprise several pieces of software running on end-user devices. On iOS and OS X devices, the primary user-facing component is the Messages application. On OS X computers, this application interacts with at least three daemons: `apsd`, the daemon responsible for pushing and pulling application traffic over the Apple Push Notification Service (APNs) channel; `imagent`, a daemon that pulls notifications even if Messages is closed; and `identityservicesd`, a daemon which maintains a cache of other users' keys. iOS devices also contain an `apsd` daemon, while other daemons handle the task of managing identities.

[2]In this analysis we considered iOS 6, 8, and 9 devices, as well as Mac clients running OS X 10.10.3, 10.10.5, and 10.11.1.

Apple services. iMessage clients interact with multiple back-end services operated by Apple and its partners. We focus on the two most relevant to our attack. The Apple directory service (IDS, also known as ESS) maintains a mapping between user identities and public keys and is responsible for distributing user public keys on request. iMessage content is transmitted via the Apple Push Notification Service (APNs). Long iMessages and attachments are transmitted by uploading them to the iCloud service, which is operated by Apple using both their own servers and virtual servers provisioned on Amazon AWS, Microsoft Azure, and Google's Cloud Platform.

3.2.1.0.1 Identity and registration

The basic unit of identity in iMessage is the iCloud account name, which typically consists of an email address or phone number controlled by the user. End-user devices are registered to the iCloud service by associating them with an account. The mapping between client devices and accounts is not one-to-one: a single account may be used across multiple devices, and similarly, multiple accounts can be associated with a single device. We give further information about the registration process in Appendix 3.10.

3.2.1.0.2 Message encryption and decryption

To transmit a message to some list of Recipient IDs, the Sender's iMessage client first contacts the IDS to obtain the public key(s) $PK_1, \ldots, PK_D$ and a list of APNs

push tokens associated with the Sender and Recipient identities.[3] It then encodes the Sender and Recipient addresses and plaintext message into a binary `plist` key-value data structure and compresses this structure using the `gzip` compression format. The client next generates a 128-bit AES session key K and encrypts the resulting compressed message using AES-CTR with $IV = 1$. This produces a ciphertext c, which is next partitioned as $c = (c_1 \| c_2)$ where c_1 represents the first 101 bytes of c. The Sender parses each PK_i to obtain the public encryption key $pk_{E,i}$ and for $i = 1$ to D, calculates $C_i = \mathsf{RSA\text{-}OAEP}(pk_{E,i}, K \| c_1)$ and a signature $\sigma_i = \mathsf{ECDSASign}(sk_S, C_i \| c_2)$. For each distinct push token received from IDS, the Sender transmits (C_i, c_2, σ_i) to the APNs server. This process is illustrated in Figure 3.1.

For each ciphertext, the APNs service delivers the tuple $(ID_{sender}, ID_{recipient}, C_i, c_2, \sigma_i)$ to the intended destination. The receiving device contacts IDS to obtain the Sender's public key PK, parses for the signature verification key vk_S, then verifies the signature σ. If verification succeeds, it decrypts C_i to obtain $K \| c_1$, reconstructs $c = (c_1 \| c_2)$ and decrypts the resulting AES-CTR ciphertext using K. It decompresses the resulting `gzip` ciphertext, parses the resulting `plist` to obtain the list of Recipient IDs, and verifies that each of ID_{sender} and $ID_{recipient}$ are present in this list. If any of the preceding checks fail, or if the Recipient is unable to parse or decompress the resulting message, the receiving device silently aborts processing.

[3]This list includes one entry for each device registered to each Sender and Recipient ID. The Messages client encrypts the message with each Sender public key to ensure that message transcripts can be read across all of the Sender's devices.

3.2.1.0.3 Attachments and long messages

For long messages and messages containing file attachments (*e.g.,* images or video), iMessage delivers the encrypted data using a separate mechanism. First, the client generates a 256-bit AES key K' and encrypts the attached data using AES in CTR mode. It next uploads the resulting encrypted document to Apple's iCloud service and obtains a unique `icloud.com` URL and an access token for the attachment. In the course of this process, the iCloud service may redirect the client to upload the encrypted file to a third-party storage server operated by an outside provider such as Amazon, Microsoft or Google. Having uploaded the attachment, the client now constructs a standard iMessage `plist` containing the URL and access token, the key K' and a SHA1 hash of the encrypted document. This `plist`, which may also include normal message text, is encrypted and transmitted to the Recipient using the standard message encryption mechanism. Upon receiving and decrypting the message, the Recipient downloads the attachment using the provided URL and access token, verifies that the provided hash matches the received attachment, and decrypts the attachment using K'.

3.3 Security goals & Threat model

Apple has stated that iMessage is an end-to-end encryption protocol that should be secure against all attackers that do not have control of Apple's network. We base

our threat model on a recent survey on secure messaging by Unger *et al.*[123] This threat model includes the following attackers:

Local Adversary. This includes an attacker with control over local networks, either on the Sender or Recipient side of the connection.

Global Adversary. An attacker controlling large segments of the Internet, such as powerful nation states or large Internet service providers.

Network operator. Apple operates centralized infrastructure for both public key distribution and message transmission/storage. Potential adversaries include Apple, a government, or a malicious party with access to Apple's servers.

Each of these attackers may be active or passive. A passive attacker simply observes traffic and does not seek to alter or inject its own messages. An active attacker may issue arbitrary messages to any party. In many cases, these adversary classes may interact. As in[123] we assume that adversaries also have access to the messaging system, and can use the system to register accounts and transmit messages as normal participants. We also assume that the endpoints in the conversation are secure, although in some cases we allow for the possibility that an attacker might briefly take physical control of a device and/or convince a user to modify device configurations.

3.4 High-level Protocol Analysis

An initial analysis of the iMessage specification shows that the protocol suffers from a number of defects. In this section we briefly detail several of these limitations. In the following sections we focus on specific, exploitable flaws in the encryption mechanism.

3.4.0.0.1 Key server and registration

iMessage key management uses a centralized directory server (IDS) which is operated by Apple. This server represents a single point of compromise for the iMessage system. Apple, and any attacker capable of compromising the server, can use this server to perform a man-in-the-middle attack and obtain complete decryption of iMessages. The current generation of iMessage clients do not provide any means for users to compare or verify the authenticity of keys received from the server.

Of more concern, Apple's "new device registration" mechanism does not include a robust mechanism for notifying users when new devices are registered on their account. This mechanism is triggered by an Apple push message, which in turn triggers a query to an Apple-operated server. Our analysis shows that these protections are fragile; in Appendix 3.10 we implement attacks against both the key server and the new device registration process.

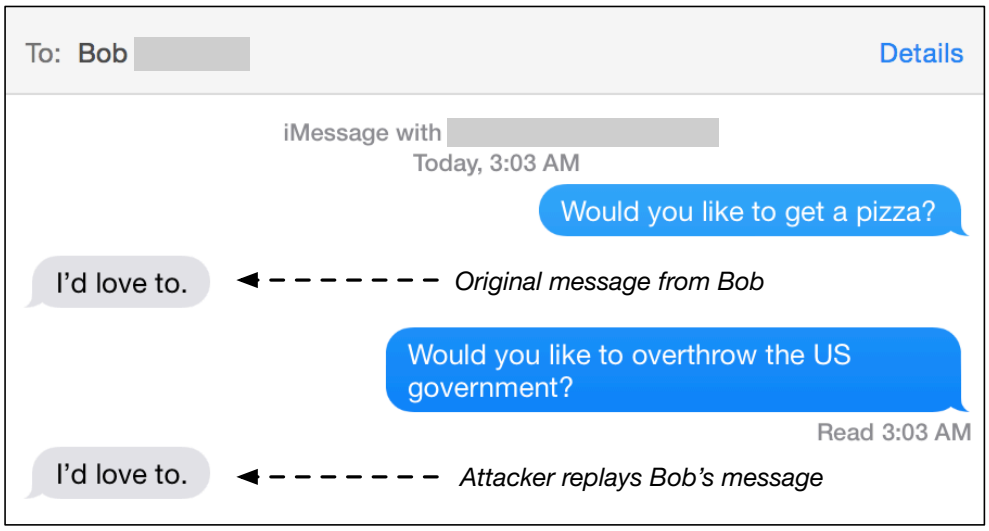


Figure 3.2: Example of a simple ciphertext replay.

3.4.0.0.2 Lack of forward secrecy

iMessage does not provide any forward secrecy mechanism for transmitted messages. This is due to the fact that iMessage encryption keys are long-lived, and are not replaced automatically through any form of automated process. This exposes users to the risk that a stolen device may be used to decrypt captured past traffic.

Moreover, the use of long term keys for encryption can increase the impact of other vulnerabilities in the system. For example, in §3.5, we demonstrate an active attack on iMessage encryption that exposes current iMessage users to decryption of past traffic. The risk of such attacks would be greatly mitigated if iMessage clients periodically generated fresh encryption keys. See §3.7 for proposed mitigations.

3.4.0.0.3 Replay and reflection attacks

The iMessage encryption protocol does not incorporate any mechanism to prevent replay or reflection of captured ciphertexts, leading to the possibility that an attacker can falsify conversation transcripts as illustrated in Figure 3.2. A more serious concern is the possibility that an attacker, upon physically capturing a device, may replay previously captured traffic to the device and thus obtain the plaintext.

3.4.0.0.4 Lack of certificate pinning on older iOS versions

iMessage clients interact with many Apple servers. As of December 2015, Apple has activated certificate pinning on both APNs and ESS/IDS connections in iOS 9 and OS X 10.11. This eliminates a serious attack noted by Raynal *et al.*[120] in which an MITM attacker who controls the Sender's local network connection and possesses an Apple certificate can intercept calls to the ESS/IDS key server and substitute chosen encryption keys for any Recipient (see Appendix 3.10 for further details). We note that devices running iOS 8 (and earlier) or versions of OS X released prior to December 2015 may still be vulnerable to such attacks. For example, at the time of our initial disclosure in November 2015 to Apple, pinning was not present in OS X 10.11.

3.4.0.0.5 Non-standard encryption

iMessage encryption does not conform to best cryptographic practices and generally seems *ad hoc*. The protocol (see Figure 3.1) insecurely composes a collection of secure primitives, including RSA, AES and ECDSA. Most critically, iMessage does not use a proper authenticated symmetric encryption algorithm and instead relies on a digital signature to prevent tampering. Unfortunately it is well known that in the multi-user setting this approach may not be sound.[124] In the following sections, we show that an on-path attacker can replace the signature on a given message with that of another party. This vulnerability gives rise to a *practical* chosen ciphertext attack that recovers the full contents of some messages.

3.5 Attacks on the Encryption Mechanism

In this section we describe a practical attack on the iMessage encryption mechanism (Figure 3.1) that allows an attacker to completely decrypt certain messages.

3.5.1 Attack setting

Our attack assumes that an adversary can recover encrypted iMessage payloads, and subsequently access the iMessage infrastructure in the manner of a normal user. The first requirement implies one of two conditions: in condition (1) the attacker is on-path and capable of intercepting encrypted iMessage payloads sent from a client to

Apple's Push Notification Service (APNs) servers. Since the APNs protocol employs TLS to secure connections between the client and APNs server, this attacker must possess some means to bypass the TLS encryption layer; we discuss TLS interception in more detail in Appendix 3.11. In condition (2) the attacker can recover iMessage ciphertexts from within Apple's network. This requires either a compromise of Apple's infrastructure, a rogue employee, or legal compulsion. Figure 3.3 describes the network flow of a single iMessage, along with potential attacker locations.

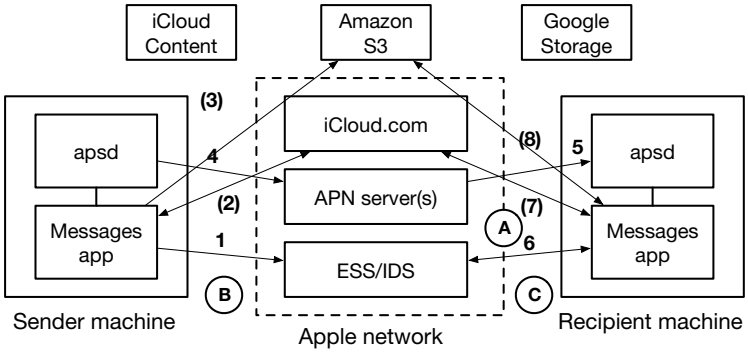


Figure 3.3: The process of sending an iMessage through the APNS network. The steps are as follows: (1) The Sender contacts ESS/IDS to obtain the public keys for each Recipient; (2) *(optional)* the Sender contacts iCloud to upload an attachment; (3) *(optional)* the Sender uploads the encrypted attachment to an outside storage provider as directed by iCloud; (4) the Sender's `apsd` instance transmits the encrypted iMessage payload to Apple's APNs server; (5) Apple delivers the payload to a Recipient; (6) the Recipient contacts ESS/IDS to obtain the Sender's public key; (7) *(optional)* the Recipient contacts iCloud if an attachment is present; (8) *(optional)* the Recipient downloads the encrypted attachment from an outside storage provider. Potential attacker locations are labeled A, B and C.

3.5.2 Attack overview

There are two stages of the attack. The first exploits a weakness in the design of the iMessage encryption composition: namely, that iMessage does not properly authenticate the symmetrically encrypted portion of the message payload. In a properly-designed composition, this section of the ciphertext would be authenticated using a MAC in generic composition[125] or via an AEAD mode of operation. Apple, instead, relies on an ECDSA signature to guarantee the authenticity of this ciphertext. In practice, a signature is insufficient to prevent an attacker from mauling the ciphertext since an on-path attacker can simply replace the existing signature with an new signature using a signing key from an account controlled by the attacker. In practice, the actual attack is slightly more complex; the first phase includes additional operations to defeat a countermeasure in the decryption mechanism, which we discuss below.

The second stage of the attack leverages the ability to modify the AES ciphertext (specifically, the section not contained within the RSA ciphertext). This phase consists of an adaptive chosen ciphertext attack exploiting the structure of the underlying plaintexts. The attack repeatedly modifies the ciphertext and sends it to either the Sender or a Recipient for decryption. If the attacker can determine if decryption and parsing were successful on the target device, she can gradually recover the underlying iMessage payload.

The attack specifics are reminiscent of Vaudenay's padding oracle attack,[126] but relies on the usage of *compression* within the iMessage protocol. Specifically, our at-

tack takes advantage of the 32-bit CRC checksum, computed over the pre-compressed message, incorporated into `gzip` compressed ciphertexts. Since CRCs are linear under XOR we can verify guesses about message content by editing the compressed, encrypted message and testing if the corresponding correction to the CRC results in a valid message.

3.5.3 A format oracle attack for `gzip` compression

The `gzip` format[127] is a variant of DEFLATE compression that combines LZ77[128] and Huffman coding to efficiently compress common data types. The format supports both static and dynamically-generated Huffman tables, though most encoders use dynamic tables for all but the shortest messages. To compress a message, a CRC32 C is calculated over the uncompressed input. Next, the encoder identifies repeated strings and replaces each repeated instance with a tuple of the form $\langle \textit{length}, \textit{backwards distance} \rangle$, where distance indicates the relative position of the previous instance of the string. The input is encoded using an alphabet of 286 symbols, comprising the 256 byte literals, an end-of-block (EOB) symbol, and 29 string replacement length values.[4] If dynamic generation is selected, a Huffman table T is calculated using the resulting text as a basis (for static tables, $T = \varepsilon$), and the text is Huffman coded into a string of variable-length symbols $S = (s_1, \ldots, s_N)$ where string replacement symbols are internally partitioned into a pair $\langle \textit{length}, \textit{distance} \rangle$. The re-

[4] A separate Huffman table is used to encode backwards distances.

sulting compressed message consists of (T, S, C). On decompression the process is reversed and the CRC of the resulting string is compared to C. If any step fails, the decompressor outputs $\perp$.

Attack intuition. Our attack assumes that the attacker has intercepted a `gzip` compressed message encrypted using an unauthenticated stream cipher and that we have access to a decryption oracle that returns 1 if and only if the message decrypts and successfully decompresses. Our goal is to recover a substantial fraction of the plaintext message.

For clarity, we assume the attacker knows the Huffman table T and the length in bits L of the uncompressed input. We further assume the attacker knows the exact location in the ciphertext corresponding to some (unknown) ℓ-bit Huffman symbol s that she wishes to recover, as well as the position of the corresponding decoded literal in the uncompressed text. These are simplifying assumptions and we will remove them as we proceed.

Given a ciphertext c, our attack works by first selecting a mask $M \in \{0,1\}^\ell, M \neq 0^\ell$ and perturbing the ciphertext such that the underlying symbol s will decrypt to $s' = s \oplus M$. This is done by *xor*ing M into the ciphertext at the appropriate location. Let $\mathsf{decode}(T, s)$ and $\mathsf{decode}(T, s')$ represent the Huffman decoding of s and s' respectively, and let $\mathsf{repeats}$ be a boolean variable that is true if and only if s (resp. s') is repeated subsequently via a DEFLATE string replacement reference. The potential values of these three variables can be categorized into the following

seven cases:

Case	$\mathsf{decode}(T, s)$	$\mathsf{decode}(T, s \oplus M)$	repeats
1	$[0, 255]$	$[0, 255]$	False
2	$[0, 255]$	$[0, 255]$	True
3	$[0, 255]$	$[256, 285]$	(either)
4	$[0, 255]$	$\perp$	(either)
5	$[256, 285]$	$[0, 255]$	(either)
6	$[256, 285]$	$[256, 285]$	(either)
7	$[256, 285]$	$\perp$	(either)

In the following paragraphs, we consider the outcome of our experiment for each of the cases above.

Case 1: In this case, when the attacker submits the mauled ciphertext to the decryption oracle, the oracle will internally decode a result that differs from the original input string in exactly one byte position: the position corresponding to symbol s'. However, with overwhelming probability, the CRC C' of the decompressed string will not match C and cause the oracle to output 0.

Because CRC is linear under XOR, the attacker may correct the encrypted value C by further mauling the ciphertext. Let d indicate the bit position of the symbol associated with s (resp. s') in the decoded message. For each $i \in \{0, 1\}^8$ the attacker *xor*s the string $\bar{C} = CRC(0^d||i||0^{L-d}) \oplus CRC(0^L)$ with the ciphertext at the known location of C and submits each of the resulting ciphertexts for decryption. Since we have that $\mathsf{decode}(T, s') \in [0, 255]$, one of these tests will always result in a successful CRC comparison.

Upon receiving a successful result from the decryption oracle, the attacker now examines the Huffman table T to identify candidate symbols s for which relation $\mathsf{decode}(T, s \oplus M) = \mathsf{decode}(T, s) \oplus i$ holds. If the attacker cannot identify a unique solution for s, she may select a new $M' \neq M \neq 0^\ell$ and repeat the procedure described above until she has uniquely identified s. The attacker can now increment her position in the ciphertext by ℓ bits and repeat this process to obtain the next plaintext symbol.

If this experiment is unsuccessful, it indicates that the ciphertext is not in Case 1 afrom the above table. To determine which case applies, the attacker must conduct additional experiments as described below. Sometimes recovery of the symbol s will not be feasible at all; when this occurs, the attacker must simply continue to the next symbol in S. Occasionally, the adversary may still be able to recover s at some additional cost.

Cases 3-4: In these cases, the original decoding of s was a byte literal, but the decoding of s' is either an invalid symbol or a special symbol (EOB or string replacement symbol). The former case always results in decompressor failure, while the latter will typically cause the decoded string to differ from the original input at multiple locations, resulting (with high probability) in a CRC comparison failure that will not be corrected by the procedure described above.

To address these cases, the attacker may select a new mask $M' \neq M \neq 0^\ell$ and repeat the complete experiment described above. Depending on the structure of the Huffman table T, and provided that $s \in [0, 255]$, the new result $s \oplus M'$ may produce

an outcome that satisfies the conditions of cases (1) or (2).[5]

CASE 2: In this case, the symbol represented by s (resp s') is referenced by one or more subsequent instances of DEFLATE string repetition. The practical impact is that modifying s will produce an identical alteration at two or more positions in the decoded string, and with high probability none of the experiments indicated for Case 1 will succeed.

In some circumstances, it may be cost effective for the attacker to skip s and simply move on to the next symbol in S. Alternatively, the attacker can experimentally modify the CRC to indicate the same alteration at *all* positions that could be affected by modifying s. Since the attacker does not know the locations at which s is repeated or the number of such locations, this requires the attacker to submit many candidate ciphertexts to the oracle, one for each possible set of locations where s may repeat. In the event that s (resp s') is repeated only once, this requires the attacker to issue $2^8 \cdot (L-d)/8$ queries to the oracle (one for each value of i and for each possible location for the repeated value of s'). This may be feasible for reasonably short strings.

CASES 5-7: These cases occur when the original symbol represented by $\mathsf{decode}(T, s)$ is a string replacement or EOB symbol. In most instances, replacing s with $(s \oplus M)$ produces a decoded string that differs from the original in many positions, making it challenging for the attacker to repair the CRC. If s decodes to a string replacement

[5]In principle, this approach might require as many as $2^8 \cdot 2^{|M|} = 2^{8+\ell}$ decryption queries to obtain a successful result, or rule out these cases. In practice, however, the number of candidate mask values M' is likely to be much more limited.

token, and the replacement reference points to a location that the attacker has already recovered, it may be possible for the attacker to detect the alteration using the technique described under Case 2. Otherwise the attacker must skip s and move on to the next symbol in S.

Recovering the unknowns. The procedure described so far requires the attacker to know the Huffman table T, the length of the uncompressed message L, the location and length of the symbol s, and the byte index of the corresponding decompressed literal. In practice many of these quantities may be determined experimentally by iterating through candidate values for L, ℓ, k and the symbol position. This requires the attacker to issue many candidate decryption requests until one succeeds. In the case of iMessage attachment messages, the length L is fixed and an attacker can generate a representative corpus of messages offline and easily estimate the other parameters *without* oracle queries.

Recovering the Huffman table is more challenging. If the message is encoded using a static table, then the table is known to the attacker. However, if T is dynamically generated, then the attacker learns only the relation $\mathsf{decode}(T, s \oplus M) = \mathsf{decode}(T, s) \oplus i$, but has no clear way of learning s or $\mathsf{decode}(T, s)$. Nonetheless, it might still be possible to recover enough information from these relations to recover the value of the underlying literals.

However, in iMessage this proves unnecessary as we take advantage of iMessage's structure to recover a large fraction of the dynamic table T. iMessage payloads con-

taining attachments embed a URL within the encrypted message. Requests to which can be monitored (described below). In this way, we learn the file path and/or hostname indicated by the plaintext URL within each ciphertext. Given this information, and by mauling individual symbols s contained within the URL string, the attacker can recover the value $\mathsf{decode}(T, s \oplus M)$ for many different values of M. This allows the attacker to identify a relative-distance map of a portion of the Huffman tree. This proves sufficient to recover much of the Huffman table T.

Detecting successful decryption. Our attack assumes that the attacker can detect successful decryption of a modified ciphertext. To simplify this assumption, we focused on messages containing attachments, such as images and videos. These messages include a URL for downloading the attachment payload, as well as a 256-bit AES key to be used in decrypting the attachment. When an iMessage client correctly decrypts such a message, it automatically initiates an HTTPS POST request to the provided URL. A local network attacker can view (and intercept) this request to determine whether decryption has occurred. Moreover, if the attacker blocks the connection, the device will retry several times and then silently abort. Since the client provides no indication to the user that a message has been received, this admits silent decryption of ciphertexts.

This technique can be also extended to situations where the attacker is not on the target device's local network. By mauling the URL field to change the requested hostname (*e.g.,* from `icloud.com` to a domain that the attacker controls), the at-

tacker can simply direct the target device to issues HTTPS to a machine that the attacker controls. This allows the attacker to conduct the attack remotely by transmitting ciphertexts through Apple's APNs network, at which point she obtains the full HTTPS POST request from the target device. Since the attacker controls the request domain, there is no need to MITM the TLS connection.[6]

3.5.4 An Attack on Attachment Messages

Having provided an overview of the attack components, we will describe each individual step of the complete attack. This attack scenario assumes that a target Sender has transmitted an attachment-bearing message to one or more online receivers, and the attacker has the ability to monitor the local network connection (and intercept TLS connections) on one of the Sender or Recipient devices.

Step 1. *Removing and replacing the iMessage signature.*

Each iMessage is authenticated using an ECDSA signature, formulated using the private key of the iMessage Sender. This signature prevents the attacker from directly tampering with the message. However, a limitation of using signatures for authenticity is that they do not prevent ciphertext mauling when an attacker controls another account in the system. An attacker who intercepts a signed iMessage may simply remove the existing signature from the message and re-sign the message

[6]The current versions of Apple's Messages client do not enforce that this URL contains `icloud.com`, and will connect to any hostname provided in the URL. Similarly, the Messages client does not pin certificates for the HTTPS connection.

using a different key, corresponding to a separate account that the attacker controls.[7] The attacker now transmits the resulting encrypted payload, signed and delivered as though from a different Sender address. The signature replacement process is illustrated in Figure 3.4.

In practice, simply replacing the signature on a message proves insufficient. In iMessage, a full list of Sender and Recipient addresses is specified both in the unencrypted metadata for the message, *and* in the encrypted message payload. Upon decrypting each message, iMessage clients verify that the message was received from one of the accounts listed in the Sender/Recipient list, and silently abort processing if this condition does not hold.[8]. While it is trivial to replace the unencrypted Sender field, replacing encrypted envelope information is more challenging. Fortunately, in most cases this field of the iMessage `plist` is contained within the malleable AES-CTR ciphertext, and we are able to alter the contents of the Sender/Recipient list so that it contains the identity of the replacement Sender account.

Step 2. *Altering the Sender identity.*

To alter the Sender identity, the attacker must selectively maul the AES-CTR ciphertext to change specific bytes of the Sender/Recipient `plist` field to incorporate the new Sender identity she is using to transmit the mauled ciphertext. This is challenging for several reasons.

[7] On Mac OS X, iMessage signing keys are readily accessible from the Apple Keychain.

[8] Based on our experiments, the participant list does not appear to be ordered, or to distinguish between Sender and Recipients. It is sufficient that the Sender identity appears somewhere in this list.

First, the initial 101 bytes of the AES ciphertext are stored within the RSA-OAEP ciphertext, which is strongly non-malleable. Thus we are restricted to altering the subsequent bytes of the ciphertext. Fortunately, the binary `plist` key-value data structure is *top heavy*, in that it stores a list of all key values in the data structure prior to listing the values associated with each key. In practice, this ensures that the relevant Sender identity appears some distance into the data structure. Moreover, the application of `gzip` compression produces additional header information, including (in many cases) a dynamic Huffman table. In all of the cases we observed, the symbols encoding the Sender identity are located subsequent to the first 101 bytes, and are therefore not included within the OAEP ciphertext.

The use of `gzip` compression somewhat complicates the attack. Rather than mauling uncompressed ASCII bytes, the attacker must alter a set of compressed Huffman symbols which have been encoded using a (dynamically-generated) table T that the attacker does not know. Fortunately, the attacker knows the original identity of the Sender, as this value is transmitted in the unencrypted `apsd` metadata. Moreover, in all iMessage clients that we examined, the Sender identity is transmitted as the first string in the Sender/Recipient list, which – due to iMessage's predictable format – appears in a relatively restricted range of positions within the ciphertext. Even with this knowledge, altering the Sender ID involves a large component of guessing. The attacker first estimates the location of the start of the Sender/Recipient list, then selectively mauls the appropriate portions of the AES ciphertext, while

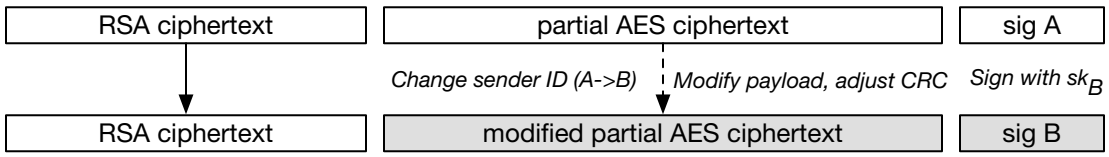


Figure 3.4: Modifying the partial AES ciphertext, including the Sender ID and CRC, and replacing the signature with a new signature corresponding to an account (and signing key) we control.

simultaneously updating the CRC to contain a guess for the modified (decoded) symbol. This is a time consuming process, since the attacker must simultaneously identify (1) the appropriate location in the ciphertext for the symbol she wishes to modify, (2) a modification that causes the symbol to change to the required symbol. The target device will silently ignores any incorrect guesses, and will proceed with attachment download only when the mauled Sender ID in the `plist` is equal to the Sender ID from which the the attacker is transmitting.

To simplify the attack, the attacker may restrict her attention to addresses that differ from the original Sender ID in at most one symbol position. This is accomplished by registering new iCloud addresses that are "one off" from the target Sender identity. To increase the likelihood that we will succeed in altering the Sender account to match one that we have selected, we register multiple new Sender identities that are near matches to the original identity. For each attempt at mauling the ciphertext, we must also "repair" the CRC by guessing the effect of our changes on the decompressed message.

In our experiments, we found that an email address of the form `abcdef@icloud.com` could be efficiently modified to a new account of the form `abcdef@i8loud.com` in ap-

proximately 2^{17} decryption queries to a target device.[9] Since Huffman tables vary between messages, we cannot mutate every message to the same domain, and thus we need to control several variants of `icloud.com` for this strategy to be successful in all cases. Fortunately, the edits are predictable and our simulations indicate that we require only one domain to recover most messages.

A side effect of this modification is that, due to string replacement in `gzip`, the attachment URL is simultaneously altered to point to `i8loud.com`, which means that attachment HTTPS POST requests are sent to a computer under our control. This makes it possible to conduct the attack remotely.

Step 3. *Recovering the Huffman table.* Given the ability to intercept the attachment request POST URL to `icloud.com`, we now recover information about the dynamic Huffman tree T used in the message. The attachment path consists of a string of alphanumeric digits, which in most instances are encoded as Huffman symbols of length $\ell \in [4, 8]$.

By intercepting the HTTPS connection to `icloud.com`, the attacker can view the decoded the URL path and systematically maul each Huffman symbol in turn, repairing the CRC using the technique described in the previous subsection. This allows the attacker to gradually recover a portion of the Huffman tree (Figure 3.5). In practice, the attacker is able to recover only a subset of the tree, however, because the iMessage client will silently fail on any URL that contains characters outside the allowed URL

[9] These email addresses are examples, and not the real email addresses we used in our experiments.

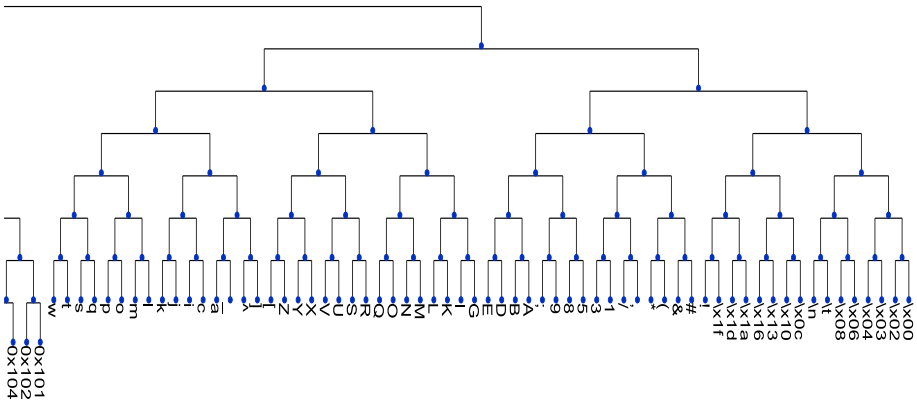


Figure 3.5: Fragment of a Huffman tree from an attachment iMessage.

character set.[10] Fortunately this set includes most printable alphanumeric characters.

Our implementation recovers a portion of the Huffman tree that is sufficient to identify the characters in the set $0-9$, $A-F$. Our experiments indicate that this phase of the process requires an average 2^{17} decryption requests and a maximum of 2^{19}

Step 4. *Recovering the attachment encryption key.* When an iMessage contains an attachment, the message embeds a 256-bit AES key that can be used to decrypt the attachment contents. This key is encoded as 64 ASCII hexadecimal characters and is contained within a field named `decryption-key`. An attacker with oracle access to a target device, and information on the Huffman table T, can now systematically recover bytes from this key. Upon recovering the key, they can use the intercepted HTTPS request information to download the encrypted attachment and decrypt it using the recovered key.

[10]iMessage does not perform URL coding on disallowed characters.

The approach used in recovering the attachment key is an extension of the general format oracle attack described above. The attacker first searches the ciphertext to identify the first position of the decryption key field. The attacker identifies a mask M (typically a single or double-bit change to the ciphertext) that produces a change in the decoded message at the first position of the encryption key, which is known due to the predictable structure of attachment messages. To identify this change, the attacker "fixes" the CRC to test for each possible result from the decryption key, then learns whether the decryption/decompression process succeeds. To obtain the full key, the attacker repeats this process for each of the 64 hexadecimal symbols of the encryption key.

This process does not reliably produce every bit of the key, due to some complications described in the general attack description above. Principal among these is the fact that some Huffman symbols represent string replacement tokens rather than byte literals. While it seems counterintuitive to expect repeated strings within a random key, this occurrence is surprisingly common due to the fact `gzip` will substitute even short (3 digit) strings. Indeed, on average we encounter 1.9 three-digit repetitions within each key. In this case, we attempt to identify subsequent appearances of the symbol by guessing later replacement locations. If this approach fails, our approach is to simply ignore the symbol and experimentally move forward until we reach the next symbol.

While it is possible to recover a larger fraction of the symbols in the message

by issuing more decryption queries (see §3.6 for a discussion of the tradeoffs), in many cases it is sufficient to simply to guess the missing bits of the key offline after recovering an encrypted attachment. In practice, the entropy of the missing sections is usually much lower than would be indicated by the number of missing bits, since in most cases the replacement string is drawn from either the URL field or earlier sections of the key, both of which are known to the attacker.

Step 5. *Recovering the message contents.* Each attachment message may also contain message text. This text can be read in a manner similar to the way the key is recovered in the previous step, by mauling the message portion of the text and editing the CRC appropriately. This approach takes slightly more effort than the hexadecimal key recovery step, due to the higher number of potential values for each Huffman symbol in the message text.

3.6 Implementation and Evaluation

3.6.1 Estimating attack duration

To validate the feasibility of the attack described in §3.5.4, we implemented a prototype of the gzip format oracle attack in Python and executed it against the Messages client on OS X 10.10.3. Our attack successfully recovered 232 out of 256 key bits after 2^{18} decryption queries to the target device. The main challenge in running the attack was to determine the correct timeout period after which we can

be confident that a message has not been successfully decrypted. This timeout period has a substantial impact on the duration of the attack, as we describe below.

3.6.1.0.1 Experimental Setup

To deliver iMessage payloads to the device, we customized an open-source Python project called `pushproxy` (hereinafter called the proxy) and used it to intercept connections from the device to Apple's APNs server.[129] This approach models an attacker who can either impersonate or control Apple's APNs servers. While our attack assumed local network interception and did not send messages through Apple's servers, we note that if an attacker is able to capture messages in transit (by bypassing TLS) or by compromising Apple's servers, the remainder of the attack can in principle be conducted remotely (see the end of §3.5.3 for details). For ethical and legal reasons, we explicitly chose not to test attacks that relayed messages via Apple's production servers. Thus all of our attacks were conducted via a local network.

To address the use of TLS on `apsd` connections, we configured our modified proxy with a forged Apple certificate based on a CA root certificate we created, and change `/etc/hosts` to redirect APNs connections intended for Apple towards our local proxy. We generate the forged certificate by installing our root CA on the target system.[11]

To monitor and intercept attachment download requests, we configured an instance of a TLS MITM proxy (`mitmproxy`) using our self-signed root certificate to

[11]Since OS X 10.10.3 does not include certificate pinning for APNs connections, this allowed us to intercept and inject iMessage ciphertexts.

intercept all outbound requests from the device made via HTTP/HTTPS. When the target device receives an attachment message, it makes two HTTPS POST requests to $\{0, \ldots, 255\}$`-content.icloud.com`. Based on the result of these requests, the device issues a second HTTP GET request to download the actual attachment. In our experiments we block both of the POST requests, ensuring that no indication of the message processing is displayed by the Messages client. For each oracle query, the attack code waits for mitmproxy to report an attachment POST request as defined above or, after a set time out, assumes the oracle query resulted in a failed message.

Finally, we created an iMessage account for the attacker that is a single-character edit of the sender's address (e.g. if the sender is alice@example.com, the attacker might be clice@example.com). We only generate one such account for the edit we expect to be successful, although a real attacker might register a large corpus of iMessage accounts and thus increase the success probability of this phase of the attack.

3.6.1.0.2 Verifying the existence of the oracle

To ensure that iMessage behavior is as expected, we conducted a series of tests using hand-generated messages to determine if we were able to detect decryption success or failure on these messages. Our results were sufficient to confirm the vulnerability of §3.5, and verify iMessage's behavior sufficiently well that we could construct a simulated oracle for our experiments of §3.6.2.

3.6.1.0.3 Estimating the timeout for failed queries

The main goal of our experiment was to determine the maximum timeout period after which we can determine that the device has been unable to successfully decrypt and process a message. To determine this, our attack queries the gzip format oracle by sending a candidate message and waiting until it either sees a resulting attachment download (in which case the message decrypted) or some timeout passes. Too long of a timeout results in unreasonable runtimes and too short of a timeout produces false negatives, which lead to incorrect key recovery.

Small scale experiments proved unable to reliably estimate the maximum timeout: the observed wait time distribution seemingly has a long tail and may be dependent on load not encountered in small experiments (e.g. due to failed decryptions). Using the full attack code to find the max timeout, on the other hand, is impractical, since we must run 2^{18} queries, each lasting as long as the timeout. This would take between 18 hours and 3 days depending on the timeout duration we wish to test.

In order to estimate the correct timeout, we ran our attack on the device in tandem with a local instance of the format oracle which, using the recipient's private key, also decrypts the message and emulates iMessage's behavior. If the candidate message fails to decrypt against the local oracle, we use a short (400ms) timeout period. If the candidate message decrypts successfully on this local oracle, then we wait an unbounded amount of time for the oracle query, and record the necessary delay. We stress that this local-oracle approach was used only to speed up the process of finding

the maximum delay; the full attack can be conducted *without* knowledge of the private key.

3.6.1.0.4 Results

We ran our main experiment on a real message intercepted using the proxy. It recovers 232 out of 256 key bits in 2^{18} queries and took 35 hours to run. The maximum observed delay between a query and the resulting download request was 903ms, while the average was 390ms with a standard deviation of 100ms. Based on this data, and without considering further optimizations, we estimate that the full attack would require approximately 73 hours to run if we naively used 1 second as the timeout.

3.6.1.0.5 Optimizing runtime

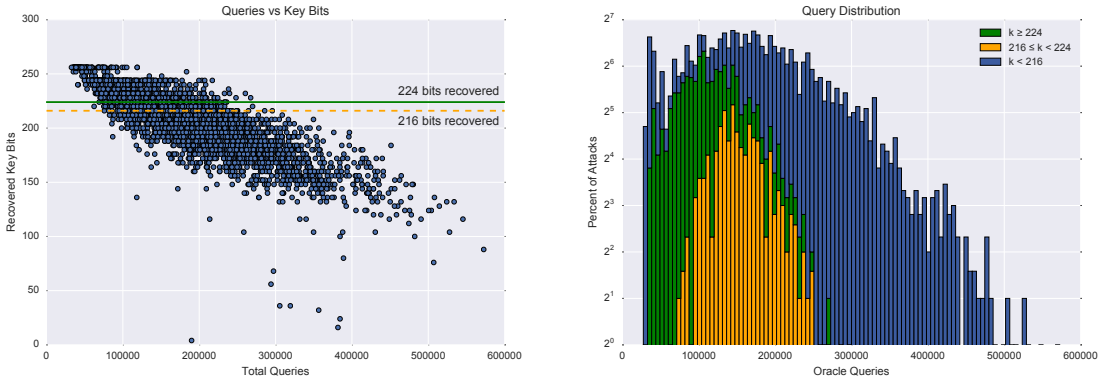
The obvious approach to optimizing our attack is to reduce the timeout period to the minimum period that iMessage requires to successfully process and queue a message. Through experiment, we determined this to be approximately 400ms. Thus one avenue to optimizing the experiment is to reduce the timeout period for all messages to 400ms, using the assumption that a successful experiment may result in a "late" download. Since we would not be able to neatly determine the specific message query that occasioned the download, we would need to temporarily increase the delay period and "backtrack" by repeating the most recent *e.g.,* 10 queries to determine which one caused the download. We are in the process of implementing

this optimization and will present the results in the full version of this work. Because successful queries are quite sparse,[12] this does not meaningfully affect the number of queries needed for the attack. In our estimation, these techniques will reduce the cost of the full attack down to 35 hours and requires only straightforward modifications to our proof of concept code.

A second optimization is to run the attack against multiple devices with attack queries split and conducted in parallel against them. For n devices, the attack time is reduced by approximately a factor of n. As many users may have 2 or 3 devices, this can offer substantial reductions.

Finally, we can reduce the raw number of queries needed to mount the attack by refining the gzip-oracle attack techniques. In particular, we can reduce the number of queries needed to recover the Huffman table by inferring the structure of the tree from the partial information we have, and from the observation that the Huffman trees fall within a fairly limited range of distributions. In particular we note that for the Huffman trees used in `gzip`, recovering the symbol lengths alone is sufficient to recover the tree. An approach drawing from techniques in machine learning to recover the Huffman table given only a few queries, the distribution of such tables, and known partial information could offer substantial improvements. We leave a full exploration of these optimizations to future work.

[12] Out of the 2^{18}, only 418 were successful.



(a) Number of queries vs number of recovered key bits. The orange dashed line represents 216 bits recovered, the solid green line 224.

(b) Distribution of attack length, measured in queries. The high concentration of attacks near zero is due to a rapid failure when it fails to edit the sender email.

Figure 3.6: Simulation results for the attachment recovery attack.

3.6.2 Simulation results

Although we have conducted our attack on iMessage, we have not explored its effectiveness with a large range of messages. Given the time it takes to run an experiment, doing so is prohibitive. We opt instead to simulate our results.

3.6.2.0.1 Simulation

To evaluate the overall effectiveness of our format oracle attack, we constructed a simulated message generator and decryption oracle. Messages produced by our generator are distributed identically to real attachment-bearing messages, but contain randomly-generated strings in place of the filename, URL path, Sender and Recipient addresses, decryption key, and "signature" (hash) fields. The decryption oracle emulates the iMessage client's parsing of the inner binary plist. For performance,

it skips encryption and decryption.[13] Decompression is done using Python's `gzip` module, which is a wrapper around on `zlib`. We experimentally validate the oracle's correctness against the transcript of a real attack and against separate messages.

3.6.2.0.2 Results

We ran our simulated attack on a corpus of 10,000 generated messages and show the results in Figure 3.6. In all cases, our experiments completed in at most 2^{19} queries, with an average of approximately 2^{17} queries. For 34% of the experiments we ran, our attack was able to recover ≥ 216 bits of the attachment AES key. For 23% of the messages we experimented with, we recovered ≥ 224 bits of the key, enabling rapid brute-force of the remaining bits on commodity hardware.[14]

3.6.2.0.3 Optimizing success rate

Many of the failures we experience in key recovery are caused by issues with string repetition. Recall that repeated substrings in a message are compressed in `gzip` by replacing all subsequent repetitions of the substrings with a backwards-pointing reference. As a result, editing the canonical location of a substring in the compressed message may cause similar changes to future instances of the same substring in the decompressed message. Our CRC correction for a given location fails to compensate

[13] Our implementation prevents the attacker from modifying the first 101 bytes of the message, as those are normally contained within the RSA ciphertext. Additionally, the oracle enforces that the alleged Sender identity is included within the `plist`, which is a condition enforced by iMessage.

[14] Experiments on an inexpensive Intel Core i7 show that we can recover 32 missing key bits in approximately 7 minutes using an AES-NI implementation. Therefore recovering 40 missing key bits should take approximately 28 hours on a single commodity desktop.

for these later changes because we simply do not know where in the uncompressed message the second instance of the substring appears. As a result, our current attack simply skips these bits.

However, we can address this weakness with only a modest increase in the number of oracle queries. By scanning through the remaining bytes and applying the same CRC correction at each subsequent location in the uncompressed message, we can identify the location of the subsequent instances of the substring. This is efficient mainly for strings that are repeated twice, but our experiments indicate this is the most common case. Note that we do not need to scan through the entire message. As a result of the particular format of the messages, there are only a few points where we can get duplicates: most of the message is in lowercase letters or non-printable characters, whereas the `decryption-key` and `mmcs-url` field (i.e. the locations where repeats cause the most serious issues) are upper case alpha-numeric and hence will not contain repeats from the majority of the other fields. For the experiments described above, this would result in a 14% increase in the number of messages for which we can recover 224 bits.

3.7 Mitigations

Our main recommendation is that Apple should replace the entirety of iMessage with a messaging system that has been properly designed and formally verified.

However, we recognize this may not be immediately feasible given the large number of deployed iMessage clients. Thus we divide our recommendations into short-term "patches" that preserve compatibility with existing iMessage clients and long-term recommendations that require breaking changes to the iMessage protocol.

3.7.1 Immediate mitigations

Duplicate RSA ciphertext detection. The attacks we described in §3.5 are possible because the unauthenticated AES encryption used by iMessage is malleable and does not provide security under adaptive chosen ciphertext attack, unlike RSA-OAEP encryption.[130] Maintaining a list of all previously-received RSA ciphertexts should prevent these replay and CCA attacks without the need for breaking changes in the protocol. Upon receiving a stale RSA ciphertext, the Recipient would immediately abort decryption. This fix does not prevent all possible replays, given that iMessage accounts may be shared across multiple distinct devices. However, it would substantially reduce the impact of our attacks until a more permanent fix can be implemented. *Note: This modification has been incorporated into iOS 9.3 and Mac OS X 10.11.4.*

Force re-generation of all iMessage keys and destroy message logs. iMessage uses long-term decryption keys, and offers no mechanism to provide forward secrecy. If possible, Apple should force all devices to re-generate their iMessage key pairs and

destroy previously-held secret keys. In addition, Apple should destroy any archives of encrypted iMessage traffic currently held by the company.

Pin APSD/ESS certificates or sign ESS responses. The current iMessage protocol relies heavily on the security of TLS, both for communications with the key server and as an additional layer of protection for iMessage push traffic. Apple should enhance this security by employing certificate (or public key) pinning within the Messages application and `apsd` to prevent compromise of these connections. Alternatively, Apple could extend their proprietary signing mechanisms to authenticate key server responses as well as requests.

Reorganize message layout. The current layout of encrypted messages includes approximately 101 bytes of the CTR message within the RSA-OAEP ciphertext, which is resilient to ciphertext malleability attacks. Modifying sender-side code to re-organize the layout of the underlying `plist` data structure to incorporate the sender and receiver fields within this section of the message would immediately block our attack. Implementing this change requires two significant modifications: (1) Apple would need to disable dynamic construction of Huffman tables within the gzip compression, and (2) restructure the binary plist serialization code to place the sender address first. We stress that this is a fragile patch: if any portion of the sender ID is left outside of the RSA ciphertext, the ciphertext again becomes vulnerable to mauling. Moreover, this fix will not protect group messages where the list of Recipients is longer than 100 bytes.

3.7.2 Long term recommendations

Replace the iMessage encryption mechanism. Apple should deprecate the existing iMessage protocol and replace it with a well-studied construction incorporating modern cryptographic primitives, forward secrecy and message authentication (*e.g.,* OTR[131] or the TextSecure/Axolotl protocol[132]). At minimum, Apple should use a modern authenticated cipher mode such as AES-GCM for symmetric encryption. This change alone would eliminate our active attack on iMessage encryption, though it would still not address any weaknesses in the key distribution mechanism. In addition, iMessage should place the protocol versioning information within the public key block and the authenticated portions of the ciphertext, in order to prevent downgrade attacks.

Implement key transparency. While many of the protocol-level attacks described in this chapter can be mitigated with protocol changes, iMessage's dependence on a centralized key server represents an architectural weakness. Apple should take steps to harden iMessage against compromise of the ESS/IDS service, either through the use of key transparency,[133] or by exposing key fingerprints to the user for manual verification.

3.8 Related Work

There are a three lines of research related to our work: secure message protocols, attacks on symmetric encryption, and decryptions attacks using compression schemes.

Instant messaging has received a great deal of attention from the research community. Borisov et al. introduced OTR,[131] and proposed strong properties for messaging, such as per-message forward secrecy and deniability. Frosh et al. analyze a descendant protocols such as TextSecure.[134] More recent work has focused on multi-party messaging[135] and improved key exchange deniability.[136] In a related area, Chen *et al.* analyzed push messaging integrations, including Apple push networking.[137] For a survey of secure messaging technologies, see.[123]

A number of works have developed attacks on unauthenticated, or poorly authenticated encryption protocols. In addition to the padding oracle of Vaudenay[126] and later applications,[138] padding oracle attacks have been extended to use alternative side channels such as timing.[139,140] Some more recent works have proposed attacks on more complex data formats such as XML.[141,142]

Some work has addressed the combination of compression and encryption. Some attacks use knowledge of a relatively small number of bytes in the plaintext to learn information about the compression algorithm and eventually recover an encryption key.[143,144] Kelsey[145] and others[146,147] used compression in the (partially) chosen plaintext setting to recover information about plaintexts.

3.9 Conclusion

In this work we analyzed the security of a popular end-to-end encrypted messaging protocol. Our results help to shed light on the security of deployed messaging systems, and more generally, provide insight into the state of the art in security mechanisms currently deployed by industry. This insight raises questions about the way research results are disseminated an applied in industry and how our community should ensure that widely-used protocols employ best cryptographic practices.

This work leaves several open questions. First, the `gzip` format oracle attack we describe against iMessage may apply to other protocols as well. For example, OpenPGP encryption (as implemented by GnuPG)[148] also employs `gzip` and may be vulnerable to similar attacks when it is used for online applications such as instant messaging.[149] Moreover, our attack requires that the adversary have some access to a portion of the decrypted information. We leave to future work the development of a pure "blind" attack on gzip encryption, one that does not require this additional information.

3.10 Attacks on Key Registration

While this work focuses on the retrospective decryption of iMessage payloads, in the course of our reverse engineering we were able to implement attacks on Apple's key registration infrastructure. The first attack is an implementation of attacks previously

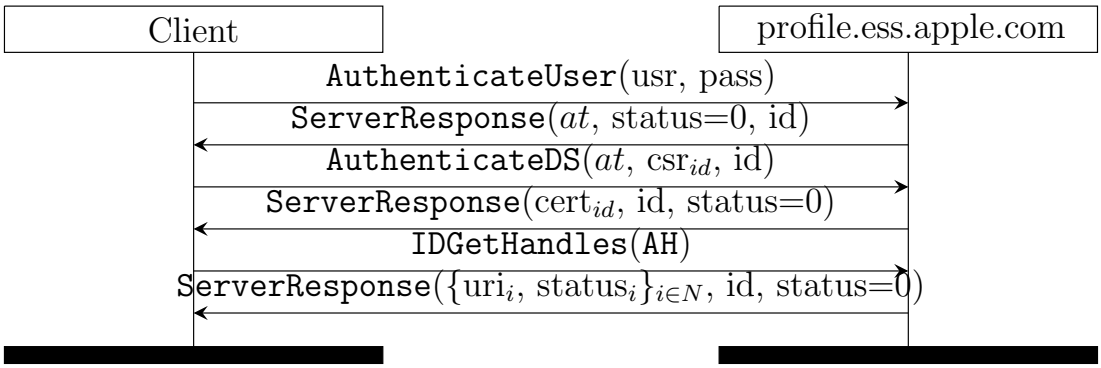


Figure 3.7: Profile conversation. usr = username, pass = password, at = authentication token pt = push token, pk_{client} = client's public key, st = session token. `AH` is an authentication header with the following fields: $cert_{device}$ = signed by the Apple Fairplay Certificate, $cert_{id}$ = a certificate associated with the client id, id, pt, $\text{nonce}_{\text{device}}$, nonce_{id}, σ_{device}, and σ_{id}.

noted by Raynal *et al.*[120] In these attacks, which work only against versions of iOS prior to iOS 9 and Mac devices prior to OS X 10.11.4 (*i.e.,* devices without key pinning), an attacker with a forged Apple TLS certificate can intercept the connection to the Apple key server in order to substitute chosen public keys. Additionally, we find a novel attack against the device registration process that allows an attack with stolen credentials to circumvent existing protection mechanisms.

The protocol for registering a device is shown in Figure 3.8. The user first establishes a TLS connection to Apple's IDS server and authenticates using their iCloud credentials. The client generates two separate key pairs: a 1280-bit RSA public key pair (pk_E, sk_E) for use in encrypting and decrypting messages, and an ECDSA keypair (vk_S, sk_S) for authenticating messages. The client transmits the public portion of these keys $PK = (pk_E, vk_E)$ to the IDS, which registers it to the user's iCloud account name. We diagram the full login and registration protocols in Figures 3.7

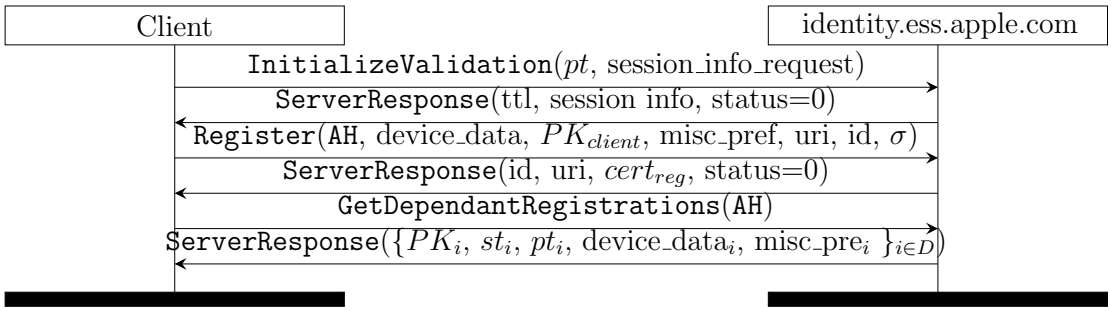


Figure 3.8: Identity conversation. pt = push token, pk_{client} = client's public key, st = session token. `AH` is an authentication header with the following fields: $cert_{device}$ = signed by the Apple Fairplay Certificate, $cert_{id}$ = a certificate associated with the client id, id, pt, $\text{nonce}_{\text{device}}$, nonce_{id}, σ_{device}, and σ_{id}.

and 3.8. To support multiple devices on a single account, the IDS will store and return all public keys associated with a given account.

3.10.1 Key Substitution Attack

The Apple key distribution systems are accessed each time a legitimate user wants to send an iMessage to a new Recipient. The Messages client first contacts `query.ess.apple.com` to look up the keys for a given username. In response, the server returns the user's public key(s), status, and push tokens for addressing APNs communications to the user. A fragment of the request and response is shown in Figure 3.9.

The `query.ess.apple.com` response message contains public keys, along with push tokens, for each of the devices registered to an account. Each of the key entries is a 332 character long base64 encoded binary payload. When decoded, they takes the form shown in Figure 3.10.

```
GET /WebObjects/QueryService.woa/wa/query?uri=mailto
%3A[redacted]40icloud.com&weight=light HTTP/1.1
Host: query.ess.apple.com
```

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>identities</key>
  <array>
    <dict>
      <key>client-data</key>
      <dict>
        <key>public-message-identity-key</
key><data>MIH2gUMAQQQzklEBPP0Nu0FHBovCJe+Prn8Rd97qf/j/ER3p2fRSe/
2BaYJnbIfEfQcpooKa3fWayu4+J1DJsIMaIwl52T7agoGuAKwwgakCgaEAoScfeVODb
EMjRrCNMWDQ2E2hWOXn46Mdqx7mLxJMS3LpGQjBoc3PeN1k3yMUqhi0YUYJJIq7dvac
1IJEiQilQDrc18eZ754BBknNmq7wXuDs8rQ2qmiE8/vOnCP4pOwwDQBy/
bdX2J3u2365R2VK6GDuk0zIjCeeAavAXr8kt9SzcvrO9KkYH1JKyKqn6FIYmR8cfeHt
ctJ0Tax8tnlZGQIDAQAB</data>
        <key>public-message-identity-version</key><real>2</real>
      </dict>
      <key>push-token</key><data>CI/
[redacted]=</data>
    </dict>
  </array>
  <key>status</key><integer>0</integer>
</dict>
</plist>
```

Figure 3.9: Excerpts from an ESS/IDS directory lookup request (top) and response (bottom). The request address and a portion of the response Push token have been redacted.

Upon receiving the RSA public key in the above diagram, the Messages client uses this key to encrypt the outgoing iMessage payload. The ECDSA key is not used when sending a message, but is used to verify the integrity of a message when it is received from that user. iMessage clients appear to accept the most recent key delivered by ESS/IDS even if it disagrees with previous entries cached by the device.

Notably, the only security measures embedded in this conversation are authentication fields in the header of the *request*; the server does not sign the response.

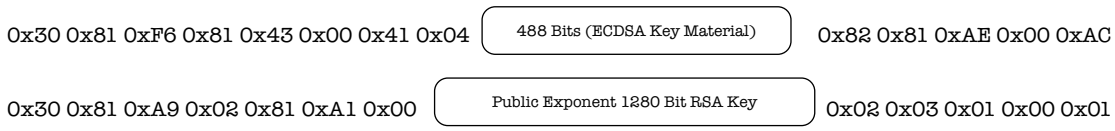


Figure 3.10: Format of public key payload in ESS server response.

Thus the authenticity of the response depends entirely on the security of the TLS connection. This seems like an oversight, given that many other fields in the Apple protocols are explicitly authenticated. Worse, in iOS 8 and versions of OS X 10.11 released prior to December 2015, the Messages client does not use certificate pinning to ensure that the connection terminated by an Apple server. Thus an attacker with a stolen TLS root certificate can intercept key requests and substitute their own key as a response. This degrades the security of iMessage to that of TLS.

We implemented this attack by installing a self-signed X.509 root certificate into the local root certificate store of a Mac device. This allowed us to verify that there were no warning mechanisms that might alert a user to the key substitution. By further intercepting messages transmitted via the APNs network, we were able to respond to all key lookup requests with our own attacker key, and subsequently decrypt any iMessages transmitted via the device.

Our experiments demonstrate that iOS 9 is no longer subject to simple key substitution attacks, due to the addition of certificate pinning on TLS connections. This increases the relative impact of our novel decryption attacks. Surprisingly, our experiments demonstrated that OS X 10.11.1 remained vulnerable as of November 2015. We notified Apple of this oversight, and they have added key pinning as of OS X

10.11.13.

3.10.2 Credential theft

The first message in the registration process, shown in Figure 3.7, passes the user's credentials to the `profile.ess.apple.com` server to be verified. As noted in previous sections, OS X 10.10.5 and iOS 8 devices do not employ certificate pinning on this server, and the credentials are sent in plaintext within the TLS connection.[15] By conducting a TLS MITM attack on this connection, we are able to intercept iCloud login credentials. Using this information we can register new iMessage devices to an account, ensuring that we will be able to receive future messages.

Apple's primary defense against registration of new devices is a notification message that is sent to all previously-registered devices. In order to register a new device to a target account without alerting the victim, we also developed a method to overcome these notification mechanisms. We observed two such mechanisms:

1. Upon registration of a new device, all devices logged into the account receive a push notification over the APNs network. In response, each device initiates the `GetDependantRegistrations` call shown in Figure 3.8.

2. When an iMessage account is registered to a device that has not previously been registered to that account, a notification email is generated and sent to

[15]OS X 10.11 devices do not employ certificate pinning on this connection either, but they do not appear to send the credentials in plaintext.

the account's registered email.

In the first instance, once the APNs push notification signaling that a `GetDependantRegistrations` call should be executed has arrived at a client, the client will continuously send the request until it receives a response. An active attacker on the victim's network can simply block all these requests, but this is not sustainable over long periods of time. We discovered that the client is satisfied when it receives any response — even a poorly formatted unreadable one. Thus, an attacker can edit the server response causing it to decode incorrectly. The client will accept this response and terminate the repeated `GetDependantRegistrations` calls. This blocks notifications that would alert the victim to the fact that a new device has been registered to their account. All subsequent iMessage traffic, both incoming and outgoing, will be forwarded to the attack device. Until a user logs out of their iMessage client, logs into a new iMessage client, or manually checks the list of devices associated with their account, they will never notice that their traffic is being forwarded to the attack device.

3.10.3 Updates in OS X 10.11

The ESS messaging protocol changed in a number of ways with the 10.11 update to OS X. The exchange of credentials for an authorization token has moved to point to `gsa.apple.com` and that connection has certificate pinning implemented. Due to this fact, we are unable to MITM this connection, but attempting to login to an account

with bad credentials will result only in a message to that server and an error message displayed on the client. Additionally, there is a message sent to `setup.icloud.com` with a username and password pair in which the password is no longer transmitted in plaintext.

The key substitution attack still worked against OS X 10.11 versions as of November 2015, but the additional certificate pinning of `apsd` made it more difficult to intercept the message. In order to make sure the attack still functioned properly, we recovered the encrypted payload of the message from the `apsd` logs and were able to successfully decrypt the message using our own keys. Although we are not able to easily intercept the messages as we could with 10.10.5, this attack still effectively reduces the security of iMessage to that of TLS.

3.11 Bypassing TLS

To execute the attacks described in this chapter, the attacker must obtain encrypted iMessages from the APNs link. Since iMessage secures the APNs connection using TLS, this requires the attacker to penetrate to the TLS encryption on the link between Apple and the end-device.

We identified three approaches to bypassing the TLS on the APNs connections: (1) Apple, or an attacker with access to Apple's infrastructure, can intercept the contents of push messages as they transit the APNs servers; (2) on certain iOS and

OS X versions that do not include certificate pinning for APNs, an attacker with access to a stolen CA root certificate may be able to conduct an MITM attack on the TLS connection; or (3) on the same versions, an attacker can "sideload" a root certificate on the target device, by briefly taking physical control of it, or convincing a victim to install a root certificate via a malicious email or web page. The latter technique is particularly concerning due to the similarity between Apple's interface for installing root CAs, and other non-critical certificate installation requests that may be presented to the user (see Figure 3.11). Since some Apple operating systems do not use certificate pinning, installation of a root certificate allows arbitrary interception of both APNs and HTTPS connections.

We identified attacks (2) and (3) as infeasible on all iOS 9 versions due to the inclusion of certificate pinning on APNs connections in that operating system. As of November 2015 when we first notified Apple of the results in this chapter, we discovered that the then-current version of OS X 10.11 did not include certificate pinning. In response to our disclosure, Apple added certificate pinning to OS X as of December 2015.

We stress that given the interest in iMessage expressed by nation-states,[115] a compromise of CA infrastructure cannot be ruled out. Even without such attacks, there have been several recent examples of CA-signed root or intermediate certificates being issued for use within corporate middle-boxes, primarily for the purposes of enterprise TLS interception.[150] TLS interception may occur even within Apple OS

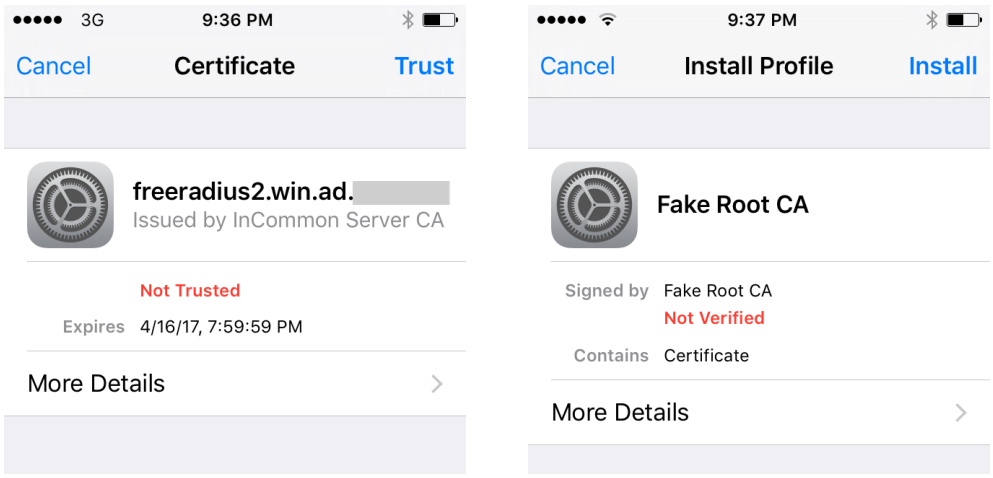


Figure 3.11: On the left is a certificate verification dialog presented on encountering an unknown wireless access point. On the right is a root CA installation dialog.

distributions: a recent incident involving iOS 9 allowed ad-blocking software to install a TLS root certificate.[151]

Chapter 4

Run-DMA

Modern computers contain a variety of special purpose, "auxiliary" processors designed to offload specific tasks from the CPU, freeing the CPU to perform other work. Conceptually, the CPU copies data from main memory to the auxiliary processor and requests that it perform its function. When the auxiliary processor has completed its task, it signals the CPU that it is finished. In reality, if the CPU were responsible for copying the data, it would spend most of its time performing data transfers, for example, copying memory to the GPU or network controller. Instead, computers have specialized hardware called *direct memory access* (DMA) engines that perform the copying to and from the auxiliary processors. The DMA engines perform the data transfers in parallel with the computation performed by the various processors by utilizing otherwise-free memory-bus cycles. In this chapter, we show that DMA engines, despite their limited functionality, are nevertheless capable of performing

Turing-complete computation.

At the same time that computer systems have been gaining additional processors, computer security researchers and practitioners have begun to recognize that the once bright-line separation of code and data is perhaps not so bright. For example, the threat of software exploitation has undergone a paradigm shift from a *malicious code* model (i.e., attacker-delivered payloads), to a *malicious computation* model where the attacker crafts data inputs to induce arbitrary computation on a target system.[152] This style of data-only attack goes by various names including return-oriented programming (ROP)[153–162] and weird machines.[152,163–165]

The ability to induce arbitrary computation from nothing more than copying bytes from one address to another may be surprising to those who are not steeped in the arcana of weird machines.[1] And indeed, it is a surprisingly strong statement: Any function that can be computed by a Turing machine can be computed using DMA.[2] The induced computation of ROP or weird machines generally takes the form of a sequence of "gadgets" which the attacker strings together to perform the desired computation. Each gadget typically performs some discrete action such as "add two numbers together" or "store a value to memory." Once a Turing-complete set of gadgets has been constructed, any desired behavior can be "programmed" in terms of the gadgets.

Turing-complete behavior in unexpected places is not sufficient to write programs

[1] For example, the x86 `mov` instruction is Turing-complete.[166]

[2] As we show in Section 4.5, DMA transfers can perform sequential interactive computation à la Persistent Turing Machines.[167]

that are interesting from a security (as opposed to a computability) perspective. To be useful, a programming language needs to be what Atkinson et al.[168] call "resource complete." That is, the language needs to "be[] able to access all resources of the system [...] from within the language".[168] By design, DMA has direct access to (some) hardware peripherals and RAM, including kernel memory and memory-mapped I/O registers.[3] Thus, a Turing-complete set of DMA gadgets should also be resource-complete.

In order to build DMA gadgets, we require several capabilities of the DMA engine. In particular, the DMA engine (1) must be capable of performing memory-to-memory copies; (2) can be programmed by loading the address of DMA *control blocks* or *descriptors* into memory-mapped registers; and (3) supports a *scatter/gather mode* where DMA transfers can be chained together, typically by providing the address of the next control block or descriptor.

Some DMA engines lack capability 1; for example, the Intel Platform Controller Hub EG20T DMA controller only supports transferring data between main memory and PCI memory [169, Chapter 12]. For DMA engines with similar restrictions, capability 1 can be relaxed as long as the restricted source/target memory contains a byte that could be used as a staging area enabling memory-to-memory copies by transferring data first to the restricted space and then back to memory.

For ease of implementation and testing, our work targets the Raspberry Pi 2's

[3] In some systems an IOMMU unit may restrict DMA access to certain regions of memory.

DMA engine (see Section 4.2) and thus we make no claim that our results hold for other systems. That said, we believe that the three required capabilities listed above are satisfied by modern DMA engines. For example, the following appear to meet our requirements: Intel 8237 (e.g., legacy IBM PC/ATs), CoreLink 330[170] (i.e., ARM Advanced Bus Architecture compliant SoCs), Cell multi-core microprocessor[171] (e.g., Sony Playstation 3), and Intel's I/O Acceleration Technology[172] (e.g., Intel Xeon Server).

Our work differs from traditional DMA malware — that is, malware that runs on an auxiliary processor such as a GPU and leverages that processor's DMA access — in that it runs entirely in the DMA engine. An attacker need only access hardware registers to exhibit control. This can be achieved in user space with administrator permissions on the Raspberry Pi 2 by mapping the appropriate region of physical memory [173, Chapter 4].

In this chapter, we are concerned with the art of crafting Turing- and resource-complete gadget sets using a DMA engine. In particular, we do not discuss how an attacker would gain permission to reprogram a DMA engine, which typically requires administrator access, nor do we discuss the full power of so-called DMA malware as both topics are well described in prior work (see Section 4.7). Concretely, we

- describe the theory behind the construction of DMA gadgets (Section 4.3);
- build an interpreter for a known Turing-complete language and demonstrate resource-completeness (Section 4.4); and

- build a proof-of-concept DMA rootkit (Section 4.5).

4.2 Background

Direct memory access (DMA), is a memory bus architectural feature that enables peripheral devices, such as GPUs, drive controller or network controllers, to access physical memory independently of the CPU. In particular, DMA frees the CPU from I/O data transfer by offloading memory operations (i.e., memory-to-memory copying or moving) to the DMA engine.

In general, each DMA engine has several control registers that specify the operation of DMA transfer, including the direction of data transfer, unit size in which to transfer (e.g., a word or a byte), and the total number of bytes to transfer. DMA transfers are typically configured by the operating system but may be initiated by hardware signals.

Our work targets the Raspberry Pi 2 for implementation and testing. Specifically, the Pi is equipped with the BCM2836 ARM processor which contains a 16-channel Broadcom DMA controller [173, Chapter 4]. DMA transfers are initiated by loading the address of a control block data structure into one of the channel's memory-mapped control registers. This causes the DMA engine to load the rest of its control registers from the control block.

The control block is composed of eight 32-bit words that specify not only which

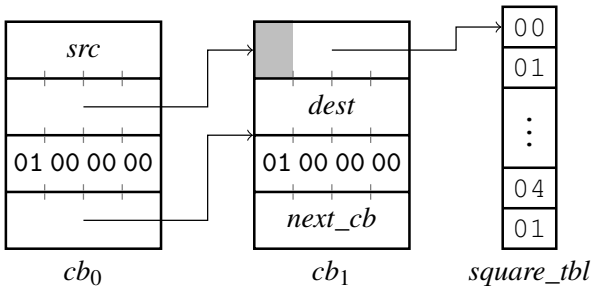


Figure 4.1: Square gadget. This simple gadget loads a byte x from address *src*, computes x^2 mod 256 by using x as an index into the *square_tbl*, and stores the result at address *dest*. The next control block to be loaded into the DMA engine is at address *next_cb*.

Figure 1: Square gadget. This simple gadget loads a byte x from address *src*, computes x^2 mod 256 by using x as an index into the *square_tbl*, and stores the result at address *dest*. The next control block to be loaded into the DMA engine is at address *next_cb*.

operation to perform, but also the address of the control block to be loaded next. The control block forms the basis of our DMA gadget construction.

A single DMA transfer is little more than a glorified, hardware-assisted `memcpy(dest, src, size)`. As described in Section 4.2, on the Raspberry PI 2, DMA transfers are initiated by loading the address of a *control block* into a memory-mapped register. Each control block contains a source address, a target address, a transfer length, and the address of the next control block to load into the engine. In addition, there are fields that control aspects of DMA transfers that are relevant to reading from/writing to DMA-supported peripherals as well as a variety of options such as 2D transfers. However, to make our results more general, we do not make use of any of these features.

Unlike traditional computer programming, constructing a DMA "program" is da

4.3 Constructing DMA gadgets

In the next section, we describe how to build a Turing-complete set of DMA gadgets which we use to build an interpreter for a simple programming language.

4 A Turing-complete gadget set

In 1964, Böhm described the simple programming language $\mathscr{P}''$ and showed that it is Turing-complete. That is, it can compute every Turing-computable function [5, 6]. It holds that a program written in the language can simulate any other computational device or language. In fact, such a program can be written using only six distinct expressions in $\mathscr{P}''$.

The toy programming language Brainfuck (hereafter referred to as BF) consists of six instructions semantically equivalent to the six $\mathscr{P}''$ expressions and two additional instructions used for input and output. To show that we can compute any arbitrary Turing-computable function, we build an interpreter for BF out of DMA gadgets. In order to implement the I/O instructions, we use DMA gadgets which interact directly with memory-mapped registers for a UART, thus demonstrating that DMA gadgets are resource-complete as well.

4.1 BF details

In this section, we give a brief overview of the BF programming language. Readers familiar with BF are encouraged to skip to the following section.

BF is a minimalistic programming language consisting of eight one-character instructions `+-><[],.`. All other characters act as a no-op. BF instructions operate on a tape divided into cells, much like the tape of a Turing machine. Each cell holds one of 256 values `00`, `01`, ..., `ff`

Program
`+ + + + + + + + + + [ >`

Tape
`01` `46`

head

Figure 2: BF example. The
with *head* currently pointing
rent instruction is a −, whic
to by *head*, setting it to ze
checks if the byte pointed t
program executes the next i
one cell to the right. Cell
setting its value to `0x44`. F
ASCII character 'D' and ha

`[` if the cell pointed to by
next instruction; otherv
following the matching
`]` if the cell pointed to b
next instruction; otherv
following the matching
`,` store input to the cell po
`.` output the cell pointed t

The increment and decrem
modulo 256 and the loop ins
Except for the loop inst
scribed above, BF instructi
A program counter, *pc*, ke
ecuting instruction. The p
pc moves past the last instr
example program that outp

4.2 Basic building blo

We construct our BF interp
basic building blocks desc
building blocks can be used
of gadgets beyond those n
Some of these are describe

Unary functions. The b
gets involves mapping son
and Figure 1 illustrate the
functions. It is frequently
$g : \{0,1\}^8 \to \{0,1\}^{32}$. We

mentally requires using self-modifying constructs. Each of our DMA gadgets consists of a collection of control blocks, chained together using the next control block fields, and zero or more tables of constant data. Most of the control blocks in each gadget modify one of the source, destination, or next control block fields in a subsequently-executed control block. For gadgets that perform basic operations such as increment values in memory, the final control block will copy the result to memory and then transition to the next gadget. For gadgets that perform control flow, the initial control blocks compute the address of the next control block to "execute" and store it in the next control block field of the final control block — a trampoline — which performs no memory transfer.

In order to compute simple functions $f : \{0,1\}^8 \to \{0,1\}^8$, we use 256-byte tables where the nth entry in the table corresponds to $f(n)$. These tables are stored 256-byte aligned in memory. By putting the address of the table in the source field of a control block with a transfer length of 1, a preceding control block can select the index n by copying a byte to the least significant byte of the source address pointing to the table. Figure 4.1 demonstrates this by giving the control blocks and table for computing the function $n \mapsto n^2 \bmod 256$.

In Figure 4.1 and subsequent figures, the source, destination, transfer length, and next control block fields of the control blocks are drawn as follows. Arrows represent pointers and shaded fields or partial fields are modified by previous DMA transfers.

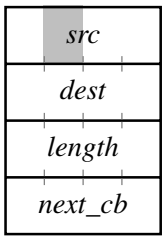
In the next section, we describe how to build a Turing-complete set of DMA

of DMA transfer, including the di-
r, unit size in which to transfer (e.g.,
f bytes to transfer in a burst. DMA
y software via a request for data
l push data directly to the system.
he Raspberry Pi 2 for implementa-
ecifically, the Pi is equipped with
processor which contains a third-
r [9]. We initiate DMA transfer in
control block data structure from
gisters. Specifically, a control block
it members that correspond to their
us specifying operation on load.

ware is malware that runs on an auxiliary
nd leverages that processor's DMA access
ntirely in the DMA engine.

the index n by copying a byte to the least significant byte of the source address pointing to the table. Figure 1 demonstrates this by giving the control blocks and table for computing the function $n \mapsto n^2 \bmod 256$.

In Figure 1 and subsequent figures, the source, destination, transfer length, and next control block fields of the control blocks are drawn as follows.



Arrows represent pointers and shaded fields or partial fields are modified by previous DMA transfers.

gadgets which we use to build an interpreter for a simple programming language.

2

4.4 A Turing-complete gadget set

In 1964, Böhm described the simple programming language $\mathcal{P}''$ and showed that it is Turing-complete. That is, it can compute every Turing-computable function.[174,175] It holds that a program written in the language can simulate any other computational device or language. In fact, such a program can be written using only six distinct expressions in $\mathcal{P}''$.

The toy programming language Brainfuck (hereafter referred to as BF) consists of six instructions semantically equivalent to the six $\mathcal{P}''$ expressions and two additional instructions used for input and output. To show that we can compute any arbitrary, Turing-computable function, we build an interpreter for BF out of DMA gadgets. In order to implement the I/O instructions, we use DMA gadgets which interact directly with memory-mapped registers for a UART, thus demonstrating that DMA gadgets are resource-complete as well.

4.4.1 BF details

In this section, we give a brief overview of the BF programming language. Readers familiar with BF are encouraged to skip to the following section.

BF is a minimalistic programming language consisting of eight one-character instructions `+-><[],.`. All other characters act as a no-op. BF instructions operate on a tape divided into cells, much like the tape of a Turing machine. Each cell holds one of 256 values `00`, `01`, ..., `ff` and is initially empty. There is an implicit tape head, *head*, which points to the current cell on the tape. The eight instructions have the follow semantics.

`+` increment the cell pointed to by *head*

`-` decrement the cell pointed to by *head*

`>` increment *head* to point to the next cell

`<` decrement *head* to point to the previous cell

`[` if the cell pointed to by *head* is *nonzero*, execute the next instruction; otherwise, jump to the instruction following the matching `]`

`]` if the cell pointed to by *head* is *zero*, execute the next instruction; otherwise, jump to the instruction following the matching `[`

`,` store input to the cell pointed to by *head*

`.` output the cell pointed to by *head*

The increment and decrement instructions `+`/`-` operate modulo 256 and the loop instructions `[]` nest as expected.

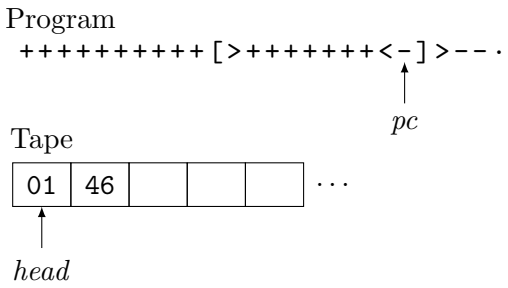


Figure 4.2: BF example. The program is in mid-execution with *head* currently pointing to cell 0 on the tape. The current instruction is a `-`, which decrements the byte pointed to by *head*, setting it to zero. Next, the right condition checks if the byte pointed to by *head* is zero; it is, so the program executes the next instruction which moves *head* one cell to the right. Cell 1 is then decremented twice, setting its value to `0x44`. Finally, the program outputs the ASCII character 'D' and halts.

Except for the loop instructions which behave as described above, BF instructions are executed sequentially. A program counter, *pc*, keeps track of the currently executing instruction. The program terminates when the *pc* moves past the last instruction. Figure 4.2 illustrates an example program that outputs the ASCII character D.

4.4.2 Basic building blocks

We construct our BF interpreter (Section 4.4.3) using the basic building blocks described in this section. These building blocks can be used to implement a wide variety of gadgets beyond those needed for the BF interpreter. Some of these are described in Section 4.4.4.

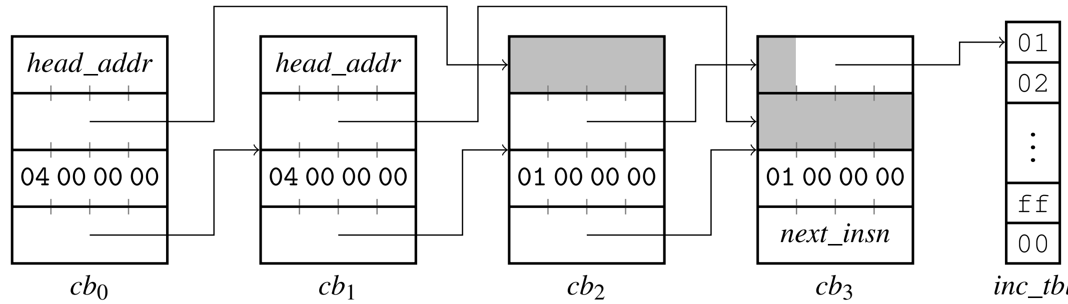


Figure 4.3: Increment gadget. The tape *head* is stored in a fixed location, *head_addr*. The first two control blocks copy *head* to cb_2's source and cb_3's destination, respectively. Then, cb_2 copies the cell pointed to by *head* into the least significant byte of cb_3's source which acts as an offset into the increment table. Finally, cb_3 stores the selected value back into the tape.

4.4.2.0.1 Unary functions.

The basic operation of most gadgets involves mapping some input to output. Section 4.3 and Figure 4.1 illustrate the construction of 8-bit, unary functions. It is frequently useful to compute the function $g : \{0,1\}^8 \to \{0,1\}^{32}$. We do this by constructing a table of the 32-bit output values and using $f(x) : \{0,1\}^8 \to \{0, 4, \cdots, 252\}$ as an offset into the table. I.e., $g(n) = table[f(n)]$.

4.4.2.0.2 Variable dereferencing.

In order to operate on data stored at a location pointed to by a pointer, we can use a control block to copy the value pointed to by the pointer into the source or destination fields of a subsequent control block. Figure 4.3 performs the operation

$$*head \leftarrow *head + 1$$

Figure 3: Increment gadget. The tape *head* is stored in a fixed location, *head_addr*. The first two control blocks copy *head* to cb_2's source and cb_3's destination, respectively. Then, cb_2 copies the cell pointed to by *head* into the least significant byte of cb_3's source which acts as an offset into the increment table. Finally, cb_3 stores the selected value back into the tape.

into the source or destination fields of a subsequent control block. Figure 3 performs the operation

$$*head \leftarrow *head + 1$$

by first copying the 32-bit address pointed to by *head* into the source field of cb_2 and the destination field of cb_3.

Conditional goto. Conditional computation is achieved by writing the address of a control block to the next control block field of a trampoline control block. Which address is written is data-dependent. These conditional gotos can be used to implement if-then-else statements as well as while and do-while loops.

As a minor space-optimization, we implement conditionals using a 512-byte aligned, 512-byte address table consisting of 128 addresses paired with a 256-byte condition table. The *m*th conditional goto in the program is associated with a pair of addresses: the addresses of the control blocks corresponding to the false condition, $cb_{m,F}$, and the true condition, $cb_{m,T}$. The two addresses are stored 256-bytes apart in the address table. For example, if the address table is stored in memory at address `0x2000`, then $cb_{m,F}$ is stored at address `0x2000` $+4m$ and $cb_{m,T}$ is stored at address `0x2100` $+4m$. Each entry in the condition table stores either the second least significant byte of the address of the table or that value plus 256. In the previous example, for each *n* for which the condition is false, the *n*th entry in the condition table would be `0x20` and for each *n* for which the condition is true, the *n*th entry would be `0x21`.

By overwriting the second least significant byte of the source field of a control block — whose source is the address table — with the value from the conditional table, that control block can copy the address of either cb_m or

which holds the addresses of the various control blocks associated with the switch cases.

Control blocks cb_1 through cb_3 in Figure 5 along with the dispatch and instruction tables are an example of a simple switch statement. ASCII values are mapped to their corresponding BF gadgets by using the dispatch table as the lookup table and the instruction table as the address table.

Memory-mapped I/O registers. Memory-mapped I/O registers are used to control hardware peripherals such as general purpose I/O (GPIO) pins, UARTs, I^2C or SPI buses, and yes, DMA engines. Interacting with such peripherals typically consists of looping, where we read a memory-mapped flag or status register over and over until a particular status is indicated (e.g., transmit buffer not full or receive buffer not empty), and then read or write a value to a memory-mapped data register. This building block is straight-forward to construct using conditionals for the loop and unary functions for the condition test.

4.3 BF interpreter gadgets

In this section, we use the basic building blocks defined in Section 4.2 to construct BF instruction and interpreter-specific gadgets. In addition to the gadgets described below, the BF interpreter requires a BF program to interpret, a region of memory to act as a tape, and three words at known addresses: a program counter, *pc*, a tape head *head*, and a loop counter, *lc*. The program counter and tape head behave as described in Section 4.1. The loop counter is used to find matching brackets in the implementation of the loop instructions.

Dispatch gadget. This specific gadget dispatches a BF instruction. We use the switch building block with the

by first copying the 32-bit address pointed to by *head* into the source field of cb_2 and the destination field of cb_3.

4.4.2.0.3 Conditional goto.

Conditional computation is achieved by writing the address of a control block to the next control block field of a trampoline control block. Which address is written is data-dependent. These conditional gotos can be used to implement if-then-else statements as well as while and do-while loops.

As a minor space-optimization, we implement conditionals using a 512-byte aligned, 512-byte address table consisting of 128 addresses paired with a 256-byte condition table. The mth conditional goto in the program is associated with a pair of addresses: the addresses of the control blocks corresponding to the false condition, $cb_{m,F}$, and the true condition, $cb_{m,T}$. The two addresses are stored 256-bytes apart in the address table. For example, if the address table is stored in memory at address `0x2000`, then $cb_{m,F}$ is stored at address `0x2000` $+ 4m$ and $cb_{m,T}$ is stored at address `0x2100` $+ 4m$. Each entry in the condition table stores either the second least significant byte of the address of the table or that value plus 256. Continuing the example, for each value n for which the condition is false, the nth entry in the condition table would be `0x20` and for each n for which the condition is true, the nth entry would be `0x21`.

By overwriting the second least significant byte of the source field of a control block — whose source is the address table — with the value from the conditional table,

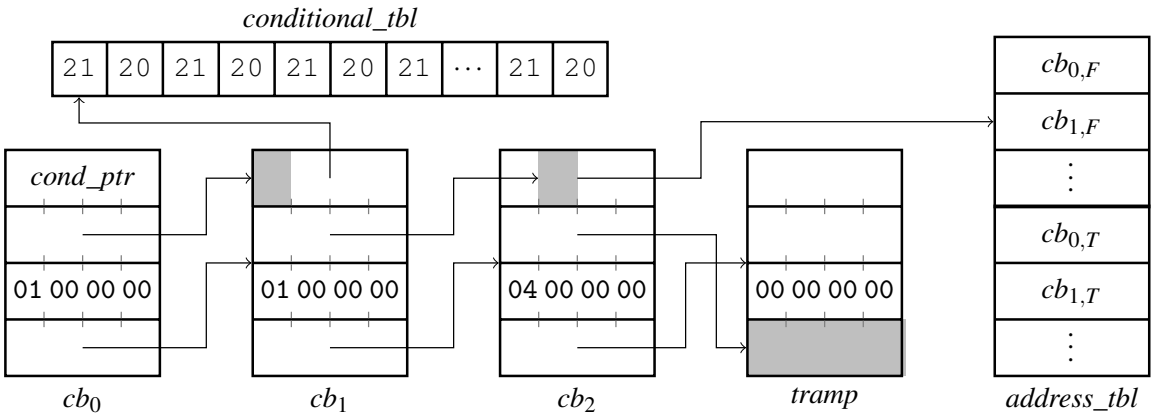


Figure 4: Conditional goto gadget. First, cb_0 copies the byte pointed to by *cond_ptr* into the least significant byte of cb_1's source to use as an index into the conditional table. Then, cb_1 copies the selected byte into the second least significant byte of cb_2's source. This byte selects which of $cb_{1,T}$ or $cb_{1,F}$ are copied into *tramp*'s next control block field. If the address table is at address `0x2000`, then if the byte pointed to by *cond_ptr* is even, then $cb_{1,T}$ will be the next control block executed. Otherwise, $cb_{1,F}$ will be.

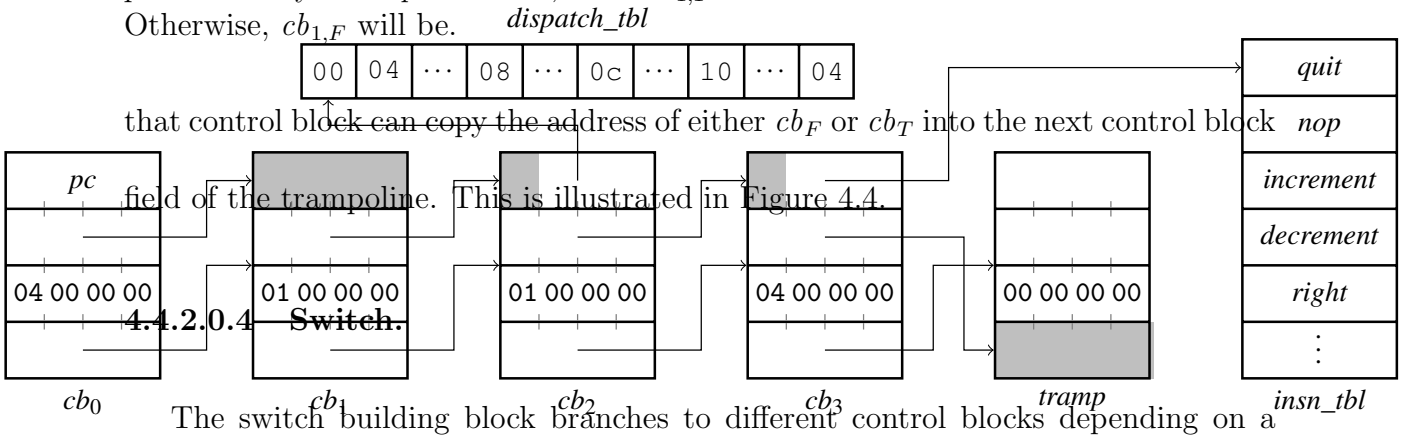


Figure 5: Dispatch gadget. The byte pointed to by the program counter is used as an offset into the dispatch table. The dispatch table contains the offset into the instruction table for the corresponding instruction. For example, the byte '+' has ASCII value 43; the 43rd entry of the dispatch table is 8; and the address of the increment gadget (see Figure 3) is stored at offset 8 in the instruction table. The control block cb_3 copies the corresponding entry from the instruction table into the next control block field of a trampoline control block.

ment (resp. decrement) and the address of the next control block to execute when the operation is complete. These work by operating on a byte at a time. First, we increment (resp. decrement) the least significant byte of the 4-byte word. If the result is `00` (resp. `ff`), then we repeat with the second least significant byte, and so on. This is a straight-forward application of unary functions, variable dereferencing, and conditionals.

Next instruction gadget. The next instruction gadget increments the *pc* by one using the increment word gadget and then jumps to the dispatch gadget.

4.4.2.0.4 Switch.

The switch building block branches to different control blocks depending on a data value. The value is used as an index into a 256-byte offset table. The entries in the offset table are the offsets into an address table which holds the addresses of the various control blocks associated with the switch cases.

Control blocks cb_1 through cb_3 in Figure 4.5 along with the dispatch and instruction tables are an example of a simple switch statement. ASCII values are mapped to their corresponding BF gadgets by using the dispatch table as the lookup table and menting *head* using the generic increment and decrement word gadgets and then jump to the next instruction gadget.

Loop instruction gadgets. The left and right loop instruction gadgets use the increment/decrement byte and word, conditional, and switch gadgets in its implementation. We use the switch gadget and define our lookup table, or *bracket table*, to contain an offset into two distinct address tables, or *scan right table* and *scan left table*, at the *n*th index, where *n* equals 0 or the ASCII byte representation of '`[`', or '`]`'. The scan right table assigns

the instruction table as the address table.

4.4.2.0.5 Memory-mapped I/O registers.

Memory-mapped I/O registers are used to control hardware peripherals such as general purpose I/O (GPIO) pins, UARTs, I^2C or SPI buses, and yes, DMA engines. Interacting with such peripherals typically consists of looping, where we read a memory-mapped flag or status register over and over until a particular status is indicated (e.g., transmit buffer not full or receive buffer not empty), and then read or write a value to a memory-mapped data register. This building block is straight-forward to construct using conditionals for the loop and unary functions for the condition test.

4.4.3 BF interpreter gadgets

In this section, we use the basic building blocks defined in Section 4.4.2 to construct BF instruction and interpreter-specific gadgets. In addition to the gadgets described below, the BF interpreter requires a BF program to interpret, a region of memory to act as a tape, and three words at known addresses: a program counter, *pc*, a tape head *head*, and a loop counter, *lc*. The program counter and tape head behave as described in Section 4.4.1. The loop counter is used to find matching brackets in the implementation of the loop instructions.

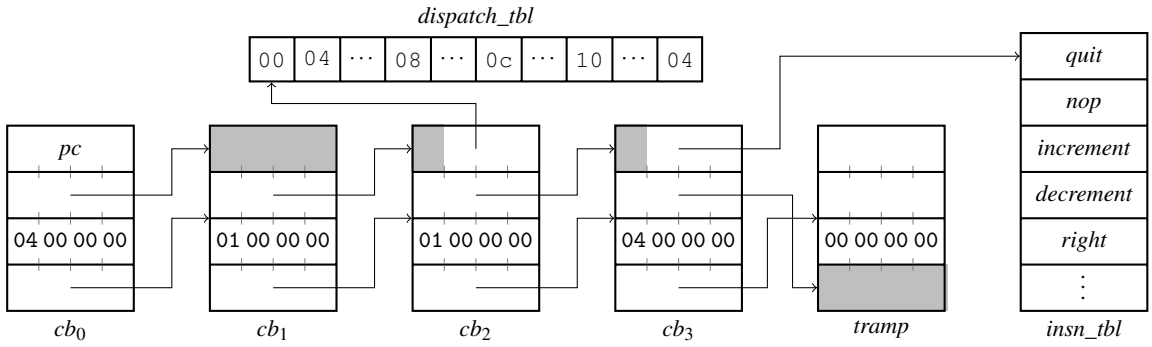


Figure 4.5: Dispatch gadget. The byte pointed to by the program counter is used as an offset into the dispatch table. The dispatch table contains the offset into the instruction table for the corresponding instruction. For example, the byte '+' has ASCII value 43; the 43rd entry of the dispatch table is 8; and the address of the increment gadget (see Figure 4.3) is stored at offset 8 in the instruction table. The control block cb_3 copies the corresponding entry from the instruction table into the next control block field of a trampoline control block.

4.4.3.0.1 Dispatch gadget.

This specific gadget dispatches a BF instruction. We use the switch building block with the *dispatch table* as the offset table and the *instruction table* as the address table. The dispatch gadget is shown in Figure 4.5.

4.4.3.0.2 Increment/decrement word gadgets.

We implement generic 4-byte increment and decrement gadgets which take as input the address of the value to increment (resp. decrement) and the address of the next control block to execute when the operation is complete. These work by operating on a byte at a time. First, we increment (resp. decrement) the least significant byte of the 4-byte word. If the result is `00` (resp. `ff`), then we repeat with the second least significant byte, and so on. This is a straight-forward application of unary

functions, variable dereferencing, and conditionals.

4.4.3.0.3 Next instruction gadget.

The next instruction gadget increments the *pc* by one using the increment word gadget and then jumps to the dispatch gadget.

4.4.3.0.4 Increment/decrement instruction gadgets.

These gadgets increment or decrement the cell pointed to by *pc* using the generic increment and decrement word gadgets and then jump to the next instruction gadget.

4.4.3.0.5 Move right/left instruction gadgets.

These gadgets move the head right or left by incrementing or decrementing *head* using the generic increment and decrement word gadgets and then jump to the next instruction gadget.

4.4.3.0.6 Loop instruction gadgets.

The left and right loop instruction gadgets use the increment/decrement byte and word, conditional, and switch gadgets in its implementation. We use the switch gadget and define our lookup table, or *bracket table*, to contain an offset into two distinct address tables, or *scan right table* and *scan left table*, at the nth index, where n equals 0 or the ASCII byte representation of '`[`', or '`]`'. The scan right table assigns its

indexes with the following control block addresses in order: scan right, increment loop counter, decrement loop counter, and quit. The scan left table inverts all operations.

We implement the left condition to first check whether the cell pointed to by `head` is zero. If it is, the gadget jumps to the next instruction gadget. Otherwise, it increments *lc* using the increment word gadget and scans right, incrementing and decrementing *lc* as brackets are encountered until $lc = 0$ at which point it jumps to the next instruction gadget.

The right condition is similar with a few exceptions. First, we jump to the next instruction if the cell pointed to by *head* is zero. At the start of scan left we decrement the *pc* using the decrement word gadget. The scan left table, as stated above, simply inverts all operations of the scan right table. This has the effect of scanning left until the matching bracket is found at which point it jumps to the next instruction gadget.

4.4.3.0.7 Input/output instruction gadgets.

Using the memory-mapped I/O building block, the input and output instruction gadgets use the Pi's UART to receive a byte and store it in the cell pointed to by *head* or to transmit the byte in the cell.

4.4.4 Other gadgets

In previous sections, we demonstrated that DMA transfers are Turing- and resource-complete by building gadgets to interpret the BF programming language and interact

with memory-mapped I/O registers. In this section we sketch the construction of a handful of building blocks that could be used to implement more efficient programs than those built using BF.

Similar to the unary function building block, we can construct arbitrary binary functions $f : \{0,1\}^8 \times \{0,1\}^8 \to \{0,1\}^8$ by using a 64-kilobyte table, appropriately aligned such that concatenation of the left and right operands forms an offset into the table. Larger binary operations can be constructed by operating 8-bits at a time. For arithmetic operations such as addition, an additional table containing a carryout bit could be used to implement carries.

Relational operators can be implemented in much the same way or they can leverage a subtraction.

Finally, DMA-specific features can be used to easily implement functionality which would otherwise be more difficult to implement or be less performant. For one example, the DMA engine on the Raspberry Pi 2 is capable of zeroing regions of memory. Another example is the Pi is capable of performing moderately complex copying modes including nonconsecutive 2D copies. Lastly, as mentioned above, the DMA is usually responsible for communicating directly with hardware peripherals and DMA engines typically support gating the transfers between devices and memory using a variety of hardware signals. This would significantly simplify access to supported peripherals.

4.5 A DMA rootkit

The most common operating system used on the Raspberry Pi is a Debian-derived distribution called Raspbian which has a Linux kernel. Linux maintains a circular linked list of `task_structs` each of which holds information about a process. The head of the list, `init_task`, is an exported kernel symbol which is exposed using the *ksymtab* mechanism. Each `task_struct` contains a pointer to `cred` structure which contains various credentials, including the user ID (UID) of the process.

We implemented a DMA rootkit that first finds the address of `init_task` and then continually walks the linked list. For each process, the rootkit examines the process's UID. If the UID matches the target UID, then the UID is changed to 0, effectively giving the process super user privileges. Any processes with the target UID that are running are modified shortly after the rootkit is started. Similarly, any processes with the target UID that are started after the rootkit are quickly modified.

Unlike the DMA gadgets described in Section 4.4, for the rootkit we utilize the DMA engine's ability to perform a 2D transfer. This enables the rootkit to copy the `task_struct`'s next struct pointer and its `cred` pointer to a known location in memory given only the address of the next `task_struct` pointer.[4]

In more detail, starting with a four-byte kernel virtual address, *va* for a `task_struct`'s next struct pointer which is stored in a fixed location p, the rootkit first converts *va*

[4]Lists in the Linux kernel contain pointers to the next element's *next element pointer* rather than to the beginning of the structure. In normal kernel code, this leads to an additional arithmetic instruction to recover a pointer to the structure.

to a bus address ba. Next, it loads the two words at ba and $ba + \Delta$ — where $ba + \Delta$ is the bus address of the `cred` pointer — to p and $p + 4$ using a 2D transfer with an appropriate stride constant Δ. After this transfer, location p contains a kernel virtual address for a `task_struct`'s next struct pointer and $p + 4$ contains a kernel virtual address for the current `task_struct`'s `cred` struct. The latter address is converted to a bus address, the UID is loaded, compared to the target UID, and on a match, 0 is written. In either case, the loop repeats.

Since the list is circular, the rootkit's logic is particularly simple. It consists of two DMA control blocks to get the address of `init_task` and an additional 18 to implement the loop, UID test, and UID setting.

4.6 Implementation

We implemented the BF interpreter described in Section 4.4.3 and the rootkit described in Section 4.5 on a Raspberry Pi 2. We were running the common Debian-based operating system, Raspbian. By default, Raspbian exposes the physical address space — including both the SDRAM main memory and the memory-mapped I/O registers — through the psuedo device file `/dev/mem`.

Our code is setuid root. It opens `/dev/mem`, maps pages of physical memory and I/O memory into the process's virtual address space, then closes the file and drops privileges. Next, it crafts DMA control blocks and tables as described above in an

unused region of physical memory. Finally, a `run_dma()` function loads the address of the first control block in the DMA engine's memory-mapped I/O control block register which begins execution of the DMA program. All of our code is available at `https://github.com/stevecheckoway/rundma`.

For input and output, we connected an FTDI UART to USB cable to the UART pins on the Pi.

4.7 Related work

There are two, mostly disjoint, lines of research related to our work: the security of auxiliary processors inside computers, and unintended, Turing-complete computation.

4.7.0.0.1 Auxiliary processors.

Security researchers have only recently begun examining the security of auxiliary processors and the firmware that runs on them. The most obvious example of an auxiliary processor is the GPU which uses DMA to transfer graphics data between the graphics card and main memory. Vasiliadis et al.[176] use the GPU to implement malware unpacking and runtime polymorphism in order to harden malware against detection. Ladakis et al.[177] use the GPU to build akey logger that monitors the system's keyboard buffer.

Duflot and Perez[178] examine the processor that runs on network interface cards (NICs). They exploit a vulnerability in the NIC's firmware to achieve arbitrary code execution

and mount a DMA attack to add a backdoor in the kernel. Triulzi[179,180] uses both the NIC and video card in concert to recover sensitive data in memory such as cryptographic keys. In follow-up work, Duflot et al.[181] construct an anomaly detection system that uses an IOMMU mechanism to limit access to main memory.

The IEEE 1394 FireWire specification allows the FireWire bus to communicate via DMA to minimize interrupts and buffer copies. Numerous researchers exploit this feature to access main memory directly.[182–185] Kalenderidis and Collinson[186] exploit Intel Thunderbolt in a similar fashion.

The Intel Management Engine (ME) is a microcontroller embedded in the Intel chip set with a separate NIC, DMA access to main memory, and remote out-of-band management technology called Intel Active Management Technology (AMT). Stewin and Bystrov[187] use the ME to build a DMA key logger, and Tereshkin and Wojtczuk[188] use AMT to construct a "Ring −3" rootkit. Similarly, Farmer[189] and Moore[190] examine vulnerabilities in the Intelligent Platform Management Interface (IPMI).

Other exploitable auxiliary processors include laptop batteries[191] and webcams.[192]

4.7.0.0.2 Unintended computation.

The ability to craft input data to drive programs in the target system has been discussed by the hacker community as far back as Aleph One's seminal article on buffer overflows.[193] Return-to-libc,[194] Krahmers borrowed code chunks technique,[195] and

return-oriented programming (ROP)[153] represent an evolution of exploitation techniques leading to Turing-complete computation built by borrowing existing program code.

ROP was first introduced by Shacham[153] as a technique to perform arbitrary, Turing-complete computation by executing a string of gadgets: short sequences of instructions, linked together by an "update-load-branch" mechanism,[158] that exist within the program or linked library. ROP has since been extended to various architectures.[154–157,159] More recent work has focused on the automation of each step in the technique.[160–162,196] For example, Bittau et al.[162] explores the limits of ROP by crafting an exploit without possessing the target's binary.

Turing-complete gadget sets need not be comprised of misappropriated CPU instructions. Indeed, parsers for complex file and record formats can be abused to provide Turing-complete computation. Oakley and Bratus[197] uses the Debugging With Attribute Records Format (DWARF) to perform arbitrary computation with the DWARF bytecode. Shapiro et al.[152] use the ELF loader mechanism to effect computation.

The prior work most similar to ours combines specialized hardware and unintended computation. Bangert et al.[163] demonstrate a Turing-complete execution environment using the IA32 arcitecture's page fault handling mechanism. Neither the page fault handling hardware nor the DMA hardware was designed with computation in mind; however, computation emerges from the hardware's complexity.

4.8 Conclusions

In this work, we have shown that DMA engines can be used to perform Turing-complete computation even though it is not their intended function. In particular, we have crafted DMA Turing- and resource-complete gadget sets that we used to build an interpreter for BF. In addition, we built a DMA rootkit to performs privilege escalation for targeted programs.

Although we are the first to build malware entirely out of DMA transfers, we are not the first to consider the capabilities DMA provides to auxiliary processors running in the system (see Section 4.2). Indeed, researchers have considered various countermeasures to such DMA malware. These countermeasures are applicable to our work as well. Example countermeasures include using the input/out memory management unit (IOMMU),[198] peripheral firmware load-time integrity,[187,199] anomaly detection systems,[181] and bus agent runtime monitors (BARMs).[198]

Several of these defenses have been found lacking. Researchers have noted that peripheral firmware load-time integrity is inadequate because it does not provide run-time integrity.[181,187] Stewin and Bystrov[187] further describes the IOMMU as lacking because it can be configured improperly, and it cannot be applied if there are memory access policy conflicts.

Given the current lack of strong defenses against DMA abuse and the ability of DMA to do both Turing-complete and resource-complete computation, it is clear that more work on secure defenses is needed.

Chapter 5

MalloryWorker: Stealthy Computation and Covert Channels using Web Workers

Adobe Flash is an example third-party plugin that was necessary to extend functionality like video streaming to web applications. HTML5 eliminates this necessity by providing new APIs that improve core functionality of the web browser (herein browser). Web Workers is one such API specified by the World Wide Web Consortium (W3C)[200] and Web Hypertext Application Technology Working Group (WHATWG).[201] Web Workers enable web applications to spawn background *workers* (i.e., threads) in parallel to the main page. Workers are intended for long-lived and computationally intensive operations that would otherwise block the UI.

Encryption, motion detection, and simulated annealing are some use cases for workers. In general, any application that has to have its execution broken up to avoid being prematurely terminated by the browser is a candidate for workers.

Despite the usefulness of concurrency in JavaScript, permissive execution of workers enables stealthy computation. Specifically, workers are instantiated unbeknownst to the user of a web application and can perform any number of computations. An attacker can cause a user to perform work for her by exploiting a cross-site scripting (XSS) vulnerability on a legitimate website or by placing an advertisement that hides the work in a worker.

We demonstrate the feasibility of stealthy computation using workers by implementing a distributed password cracker that uses the Web Workers API. We can compute 500,000 MD5 hashes per second using the Chrome browser on a 2012 MacBook Air. We also implement a denial-of-service (DoS) attack that is unique to OS X. We define *wasteful* stealthy computations that exploit garbage collection mechanisms in Chrome, Firefox, and Safari. The result is high CPU and memory utilization that eventually fills the swap partition and causes a deadlock.

We again use wasteful stealthy computations against the Android Chrome browser. This time, we find exploiting garbage collection results in a resource depletion attack. We did not attempt this on the mobile Safari browser for iOS but believe that it is also susceptible because it is built on WebKit much like its browser counterpart.

A natural criticism to both the DoS and resource depletion attack is that a worker

is unnecessary to perform either attack. However, we attempted all stealthy computation attacks in the single JavaScript and UI thread. The result was the page becoming unresponsive, followed by the Browser terminating the process. Neither of which happen in our worker implementations.

We describe and implement a covert channel that is not unique to workers but is easily implemented using them. Our covert channel uses CPU and memory throttling to transmit bits to an unauthorized application. We find that CPU throttling is noisier than memory throttling because other processes can obscure our covertly transmitted bits (i.e., a random peak can corrupt bits or semantic structures such as a preamble). We throttle memory by exploiting garbage collection to create a peak and then terminating the web worker to force garbage collection anyway.

We scanned 7000 websites from Alexa's top sites to determine the prevalence of worker use. We found that 1.2% of them use workers to perform some computation. Websites such as yahoo.com, usbank.com, and mediafire.com use workers for various reasons. For example, usbank.com uses a worker defined in foresee-worker.js to compress session event logs.

In this paper, we are concerned with using the Web Workers API to create workers that enable stealthy computation and covert channels. We demonstrate the feasibility of these by implementing our own distributed password cracker using workers, a DoS attack against OS X, a resource depletion attack against Android, and a covert channel using memory throttling. We provide the necessary background for JavaScript

code execution and Web Workers, discuss related work focused on HTML5 vulnerabilities, and we give the first mitigation strategy for the misuse of workers.

5.2 Background

Web browsers typically have one thread that JavaScript and the UI share. Therefore, UI updates are blocked while the JavaScript interpreter executes code and vice versa. A shared task queue enables asynchronous execution of JavaScript and UI updates, allowing either to execute when the thread is available. Asynchronous execution does not solve the problem of an arbitrary script taking unusually long. The browser attempts to terminate any script that takes longer than some threshold regardless of its purpose or importance. The user is aware of this when the UI freezes. Not much later, the browser presents a status (i.e., terminate or continue) or crash message.

The browser's approach to ending long-running scripts is undesirable because it provides no context per the scripts execution. The user is unaware of what the script is meant to do and how long it has been running. Web application developers approach this issue by leveraging asynchronous execution and dividing their scripts into logical chunks that execute on some period. This method does not benefit from parallel execution where a computation is uninterrupted until it finishes.

HTML5 addresses these limitations with the Web Workers API. This API enables

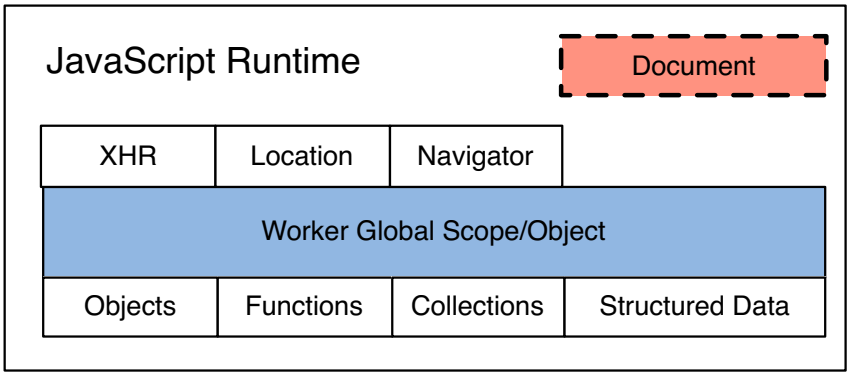


Figure 5.1: Web Worker JavaScript Runtime.

web applications to spawn background workers in parallel to the main page. We show in Figure 5.1 that workers are unable to access the Dynamic Object Model (DOM) or the callers (i.e., parent object) variables and functions. Workers are instantiated as one of two types: shared or dedicated.

Shared workers can be accessed by multiple web applications but dedicated workers cannot. Web applications instantiate both shared and dedicated workers by providing a script object to the `Worker` constructor. The script object is either an externally loaded file or defined *inline* as a string description of the web worker.

The string description is provided as input to the `blob` constructor, a file-like object, and is referenced by an output URL handle. This URL handle is provided to the `Worker` constructor. See Listing 5.1 an example inline instantiation.

```
<script id="mw" type="javascript/worker">
  self.onmessage = function(event) {
    self.postMessage({'msg': 'hello.',});
  }
```

```
</script>
<script language="javascript">
  var blob = new Blob([document.querySelector('#mw').textContext]);
  var m_worker = new Worker(window.URL.createObjectURL(blob));
</script>
```

Listing 5.1: Instantiate worker using blob.

Workers support communication with each other and its parent object via message passing. The `onMessage` method listens for messages and upon receiving one it will call the `postMessage` method to send a message. Workers continue to listen for messages until the user navigates away from the web application, or the parent object calls the `terminate` method on the worker. Terminating a web worker causes garbage collection on all allocated memory.

5.3 Threat Model

We use the definition of a *web attacker* and *gadget attacker* by Akhawe et al.[202] to define an attacker that maliciously misuses workers. A web attacker operates a malicious web application but has no visibility into the network beyond the requests directed to her application. A gadget attacker can inject content into otherwise legitimate web applications.

A web attacker that misuses workers hosts a web application with a mechanism for

generating traffic (e.g., misleading domain name or social engineering). Every time a user visits the web application, stealthy computation is performed via a worker or workers. A gadget attacker that misuses workers exploits web vulnerabilities such as cross-site scripting to inject her workers. She may also purchase a web advertisement and bundle her workers in the ad. A user that visits a legitimate site will now perform some stealthy computation.

A web attacker is considered an insider threat; for example, a web application administrator. A gadget attacker is an outside threat. She is simply a web application user. We consider both attackers to be unsophisticated as neither has visibility or control of the network. Also, both attackers rely on generally accessible tools such as a laptop, internet access, and at most a web server.

The goals of both a web attacker and gadget attacker that misuse workers include: performing stealthy computation, mounting a DoS or resource depletion attack, and establishing a covert channel with an unauthorized application.

5.4 Web Worker Primitives

While creating stealthy computation is as simple as writing function x, a wasteful computation needs to exploit garbage collection mechanisms for multiple browsers. Covert channels also require a mechanism for throttling a system's CPU and Memory. We introduce three primitives to achieve wasteful stealthy computation, CPU

throttling, and memory throttling.

5.4.0.0.1 Infinite Loop Sequences.

An infinite loop is a sequence of instructions which loops endlessly because the boolean condition never changes (e.g., it always evaluates true). If an infinite loop is executed by the JavaScript interpreter, the browser UI will freeze due to blocking on the shared thread. However, blocking does not occur if this loop is executed in a worker.

We use an infinite loop such as `while(true){}` to perform a wasteful stealthy computation. This type of computation enables CPU and memory throttling. Again, the execution of this loop is unknown to the user because it does not block the UI thread.

5.4.0.0.2 CPU Throttling.

Executing an empty infinite loop alone will not throttle a modern CPU. Instead, we achieve throttling by looping on intensive operations such as recursive function calls and large data manipulation. Listing 5.2 implements a data manipulation loop that randomly fills two 1024-byte arrays and then concatenates them.

```
var cpu_work = function() {
  var scratch = [];


  // Fill the ArrayBuffer with random values.
```

```
  for(var j = 0; j < 1024; j++) {
      scratch.push(Math.random());
  }

  var firstArr = new Uint8Array(scratch);
  var secondArr = new Uint8Array(scratch);

  // ArrayBuffer concatenation.
  var concatBuf = new Uint8Array(firstArr.byteLength + secondArr.
     byteLength);
  concatBuf.set(new Uint8Array(firstArr), 0);
  concatBuf.set(new Uint8Array(secondArr), firstArr.length);
}
```

Listing 5.2: Browser CPU throttling.

5.4.0.0.3 Memory Throttling.

Throttling memory is browser specific as it exploits corner-cases not yet handled by the browser's garbage collection. We note that the browser does, in fact, do garbage collection correctly; however, the process is approximate as deciding whether memory can be freed is *undecidable*. We use this knowledge to our advantage to discover browser-specific memory leaks and use them to throttle system memory.

In Listing 5.3 we use a technique outlined by Glasser[203] to demonstrate a memory leak in Firefox. This technique relies on JavaScript closures. Specifically, both `unused`

and `bucket` are both defined inside of `RD_ATTACK_FIREFOX_SAFARI` scope, and if both functions access the variable `leak` it's imperative that both get the same object. So `leak` is never garbage collected.

In our experimentation with these primitives, we crashed Firefox and Chrome when throttling CPU and memory. We mitigate this by using the worker method `terminate()`. This method helps us avoid crashing the browser and completes our throttling primitives by exposing a mechanism for quickly freeing system resources.

```
var bucket = null;
var RD_ATTACK_FIREFOX_SAFARI = function () {
  var leak = bucket;
  var unused = function () {
    if (leak) {
      var hole_in_bucket = 1;
    }
  };
  bucket = {
    longStr: new Array(10000000).join(Math.random()),
    someMethod: function () {
      var hole_in_bucket = 2;
    }
  };
  // Placeholder for doing some repetitive operation.
  cpu_work();
};
```

Listing 5.3: Firefox memory throttling.

5.5 Stealthy Computation

We demonstrate the feasibility of stealthy computation using workers by implementing a distributed password cracker that uses the Web Workers API. We implement the main HTML page to define a target MD5 password hash, a worker instantiation, and an event listener to receive the result of password cracking (i.e., an MD5 collision was found).

The worker instantiation is on input `md5cracker.js`. This worker script defines the MD5 hashing algorithm, a dictionary download method, and the event listeners start and stop.

The start listener waits to receive the string start. When it receives the string, it downloads an array of passwords using the method `importScripts()`. This method synchronously imports a script into the worker's scope. We use it to import an array of passwords because we want the worker to be self-contained. Specifically, if an attacker should inject a worker or upload an advertisement with a worker, she can not rely on the calling parent object to pass in any data such as an array of passwords.

After downloading the array, the worker selects a random index into the array and begins to hash each password and compare it to the target hash. If it finds a

collision, it returns the result to the parent object, or it could use a web socket to send it elsewhere (e.g., the attacker's server).

The stop listener simply kills the worker once it is no longer useful.

We send 1 million passwords to the worker using `importScripts()` which is approximately 13MB. This step adds approximately 50% latency on the dataset and takes 3 seconds to download. We can minimize this time by compressing the password array and partitioning the array into multiple arrays. The password cracker performs 500K hashes per second on a 2012 MacBook Air.

The average user visits a website for no longer than 15 seconds. Thus, one criticism we receive is that stealthy computation doesn't have the much time to do any worthwhile computation. We argue that stealthy computation on video streaming sites such as Youtube is plausible. In addition, there exist other projects on the internet that do stealthy computation using workers such as bitcoin mining.[204]

5.5.1 Denial-of-Service

We use our loop and memory throttle primitives to mount a DoS attack against all OS X devices. This DoS is unique to OS X because of the way virtual memory is handled. Specifically, OS X can grow its swap file to the maximum available size of the hard disk. If we exploit garbage collection for a very long period, OS X will grow the size of its swap to the point that it deadlocks.

The steadily growing swap in Figure 5.2 depicts our exploitation of garbage col-

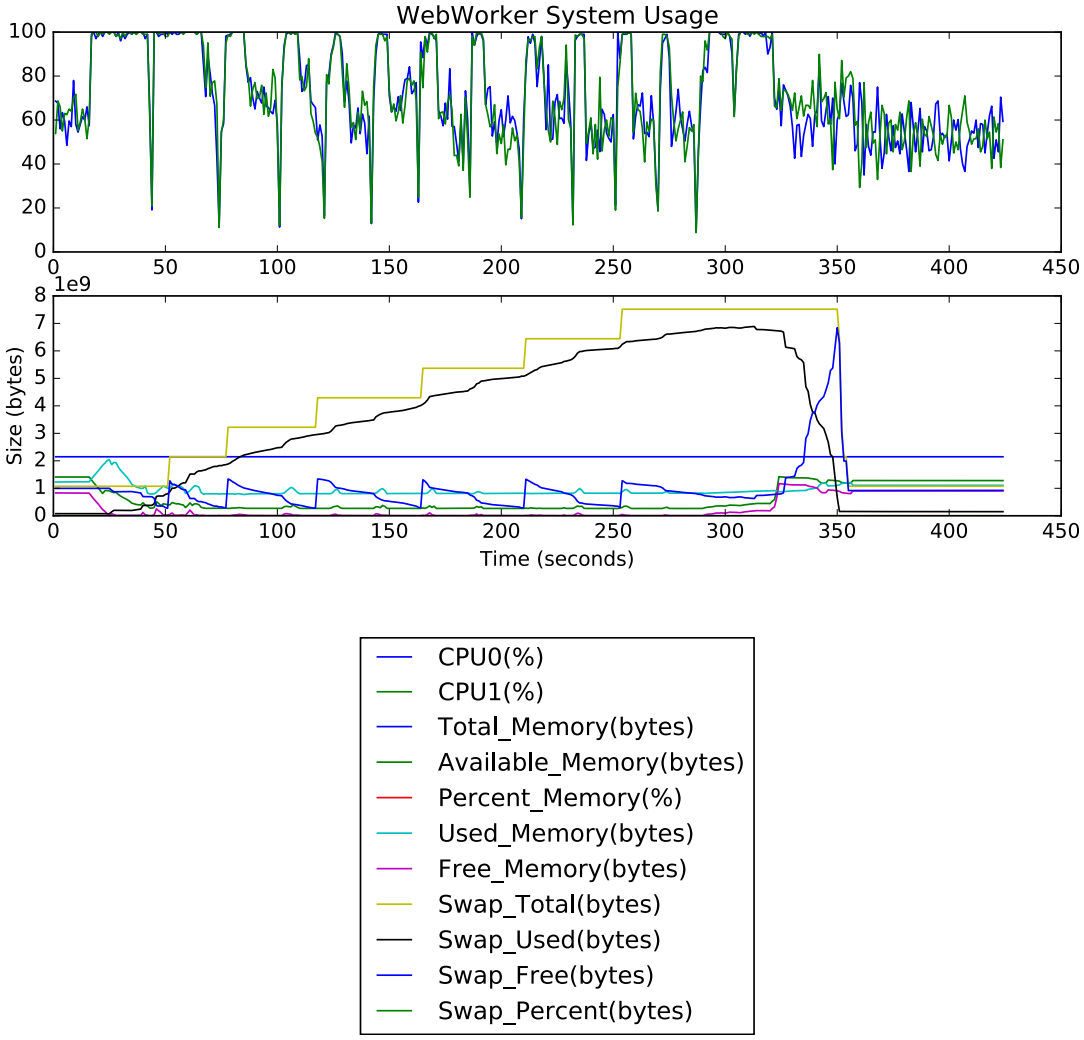


Figure 5.2: OS X Firefox DoS attack.

lection on Firefox. OS X needs to be hard rebooted when it deadlocks. Fortunately, disk space is recovered and the swap returns to its original size.

5.5.2 Resource Depletion

The mobile Chrome browser also supports the Web Workers API. Figure 5.3 depicts user memory usage as it steadily increases from the stealthy computation.

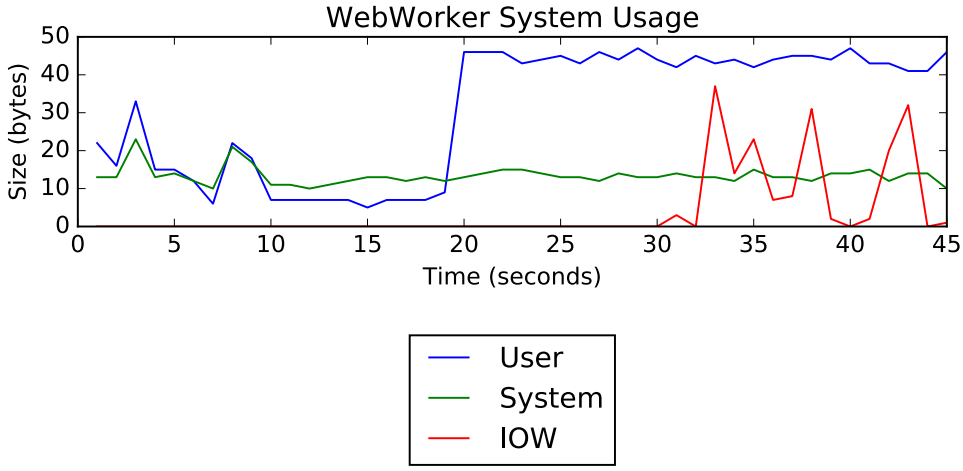


Figure 5.3: Android Chrome resource depletion attack.

The over usage of memory results in I/O waiting toward the end of our experiment. Stealthy computation can excacerbate resource depletion as it uses system resources to perform wasteful work.

5.6 Covert Channel

A covert channel is a communication mechanism for two processes that are not supposed to be able or allowed to communicate. We use our CPU throttling and memory throttling primitives to create a covert channel between a visited web application and some unauthorized application on the user's system using workers.

We first try CPU throttling to observe messages with a simple structure. Specifically, we do not define a pre or postamble; rather, we define a period in which to observe a bit based upon a CPU usage spike. We find that the CPU channel is noisy, as seen in Figure 5.4, and we can only achieve good accuracy by employing a

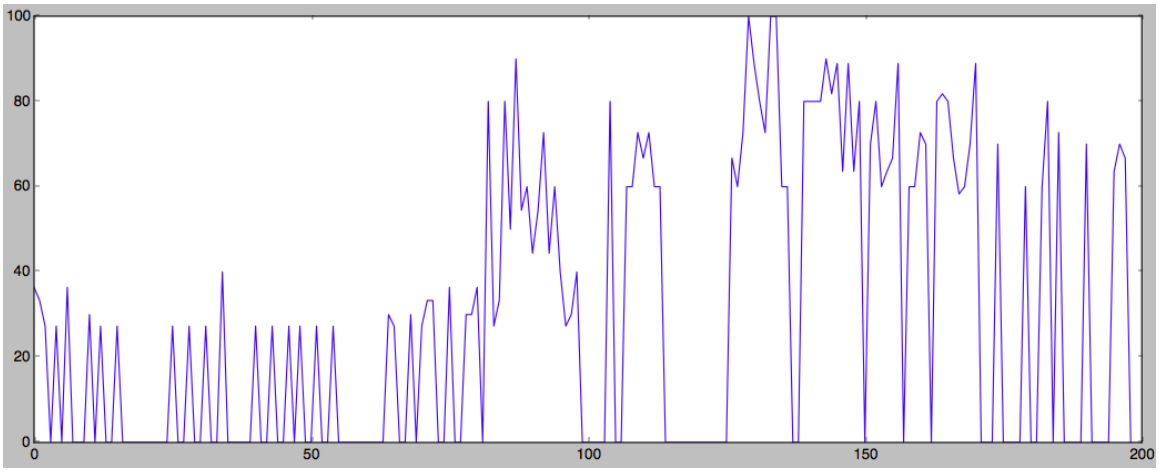


Figure 5.4: CPU noise during regular use.

high sampling rate. Unfortunately, we use PSUTIL to get current CPU usage and it imposes a sampling rate with a minimum bound of 100 milliseconds. Also due to JavaScript runtime limitations, anything less than one millisecond isn't feasible.

We attempt to minimize CPU noise by increasing the length between CPU spikes to 500 milliseconds and 1 second. We can obtain bits in the covert channel but under ideal conditions. For example, if any other work is done in the browser it significantly impacts our ability to discern relevant CPU spikes.

Next, we try our memory throttling primitive. Memory usage is a more deterministic channel and thus less noisy than CPU usage. This makes it more viable as a covert channel. We use our memory throttling primitive to fill a 40MB array and then clear the memory with a `terminate` worker method call. We can successfully send 1 bit per 5 seconds. We send the bits for "hello world" in Figure 5.5. Unlike the CPU covert channel, the memory covert channel is usable when the user browses the

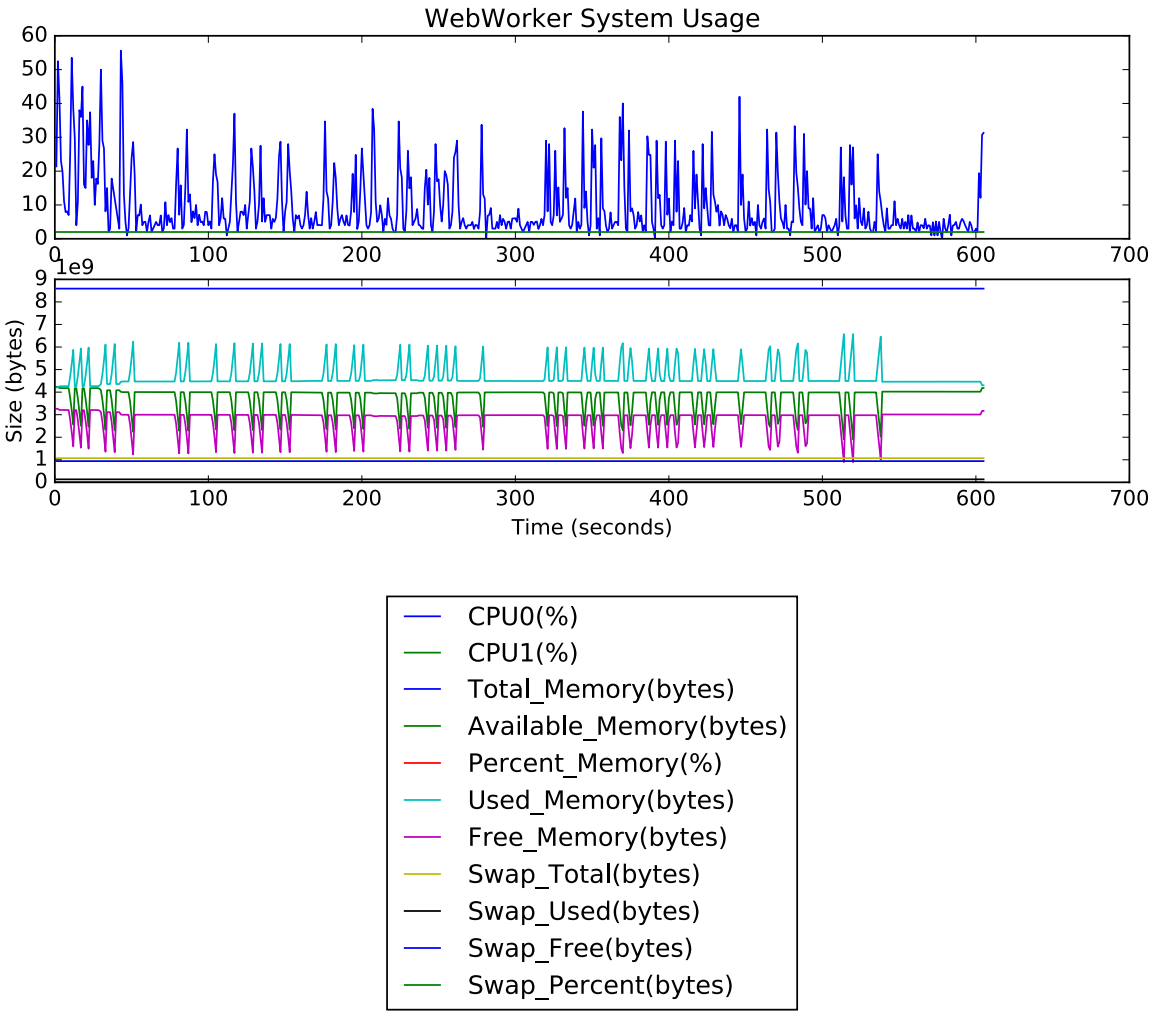


Figure 5.5: Memory covert channel sending hello world.

internet or stream videos. This finding is a consequence of the amount of memory used which far exceeds the memory needed to buffer a video in our tests.

We note that our covert channel does not require a web worker. However, when executing the covert channel in the UI thread, the browser is less responsive due to the looping execution of the memory primitive. In addition, our ability to force garbage collection by terminating the worker must be exactly emulated in the UI thread or it

will break the covert channel (i.e., no discernable memory usage peaks).

We implement and test this covert channel on OS X using the Firefox and Chrome browsers. The covert channel is inefficient regarding channel bandwidth; we can send approximately 1 bit per 5 seconds. We can speed this up by reducing the amount of memory throttled (e.g., less than 40MB). We could also use more than worker.

5.7 Potential Mitigations

The challenge for the Web Workers API is how to inform users a worker is executing, what the intent of the execution is, and how the execution is impacting system resources. We assert that the most effective solution is to provide fine-grained controls for workers similar to pop-up controls, and to restrict the Web Workers API in the ECMAScript specification. For instance, requiring an explicit intent for every worker would provide context to what the purpose of the worker is and enable a user to decide whether to allow it. A Google Chrome extension is a good example because it uses a manifest file to specify the capabilities of the extension. These capabilities are analogous to a worker intent.

In the interim, we implement a browser extension to mitigate worker stealthy computations partially. This mitigation is partial because the browser extension only informs the user of when a worker has executed. If the worker is named appropriately, the user is provided with some context of the workers intent, but name mangling and

poor coding practices will undo this. We call our browser extension *wAudit*.

wAudit is a Google Chrome content script. Content scripts use the Document Object Model (DOM) to read and modify details of a visited web page. These scripts, however, cannot use or modify variables or functions defined by the visited web page. For wAudit to determine whether a worker exists it must be able to the later.

We programatically inject wAudit as a script into visited web pages using `document.createElement`. This function creates an HTML script element that we append to the document object's root element using the function `document.documentElement.appendChild`. The injected script recursively searches all DOM objects and identifies object types of `[object Worker]`.

The script alerts the user if it finds a worker or workers by drawing a banner at the bottom of the browser window. This banner includes the name of the worker and a UI button for terminating a selected worker. We implement the terminate function by crafting the string `"workers[i]+".terminate()`. This string contains the worker name and the method call to terminate. We call `eval` on the string input to execute.

5.8 Related work

Security researchers have found numerous vulnerabilities in the HTML5 APIs that enable traditional web application attacks such as CSRF and clickjacking, and HTML5-specific attacks such as cache poisoning and botnets.

Tian et al.[205] show that the HTML5 screen-sharing API can allow for cross-site request forgery (CSRF) attacks, even if the target website utilizes CSRF defenses such SSL and secure random tokens. The authors are also able to sniff user account, autocomplete, and browsing history data because it can be viewed directly on the user's screen. Potential defenses are enumerated as restrict the loading of view-source links, enable fine-grained sharing, and constrain cross-origin content.

The HTML5 FullScreen API displays web content that fills the user's entire screen. Aboukhadijeh[206] describes how a malicious website can trick users into clicking a link to a legitimate website (e.g., https://www.bankofamerica.com/), and then display a malicious website in fullscreen. The malicious website imitates the legitimate website and obscures the domain name and SSL visual indicator.

Kuppan[207] overviews multiple HTML5-specific attacks. For example, an attacker can use the HTML5 Drag and Drop API to trick users into setting target form fields with attacker controlled data, a clickjacking attack. An attacker can poison HTML5 caches designed to enable offline browsing with her own pages that recover user supplied data. Specific to our work, workers enable HTML5 botnets. These botnets can mount distributed denial-of-service (DDoS) attacks by sending cross-domain XMLHttpRequests.

Anibal Sacco et al.[208] use workers to optimize heap-spray attacks. By employing multiple workers, the authors show that they can populate the target systems' memory faster than conventional heap-spray attacks. They leverage HTML5 can-

vas objects to obtain both full control over consecutive heap pages and to provide byte-level access to pixel information. This gives four bytes per pixel for use in spray contents – typically a use-after-free exploit, heap-based buffer overflow, or ROP chain. Also, due to the increasing prevalence of browser-based devices with HTML5 support (smartphones, smart TVs, game consoles, etc.) the use of workers as an attack vector are largely platform and browser agnostic.

The Open Web Application Security Project (OWASP) blog[209] mentions the use of workers to perform denial-of-service attacks. The post gives a cursory treatment of these vulnerabilities and does not provide any concrete details regarding implementation, measurement, or countermeasures.

In general, defenses for HTML5 API vulnerabilities include modifications to the APIs. Son and Shmatikov[210] find that many web applications perform origin checks incorrectly, if at all. The lack of stringent checking allows for cross-site scripting (XSS) attacks, as well as data injection into local storage. The authors propose accepting only messages from the origin of the page that loaded a frame and the parent of that frame.

Akhawe et al.[211] find that HTML5 web applications need better privilege separation. Rather than advocate for browser redesign or artificial limits on partitions, the authors propose a way for HTML5 applications to create an arbitrary number of unprivileged components. Each component executes with its own temporary origin, isolated from the rest of the components. Unprivileged components interact via a

privileged component that executes priviledged calls in the main origin of the web application. The authors show that their system helps reduce the amount of trusted code by a factor of 6 to 10000.

5.9 Conclusions

We described how the Web Workers API can be used to create workers that enable stealthy computation and covert channels. We demonstrated the feasibility of stealthy computation by implementing a distributed password cracker using workers, a DoS attack against OS X, and a resource depletion attack against Android. We evaluated the feasibility of a covert channel using CPU and memory throttling, and implemented the later. Lastly, we gave the first mitigation strategy for the misuse of workers.

5.10 Health and Medical Systems

Health and medical systems are increasingly becoming networked. An industry report by Parks Associates predicts that networked medical systems will exceed 14 million sales in 2018.[212] These medical systems often employ commodity operating systems such as Windows Embedded and can access and be accessed over the internet.

Manufacturers troubleshoot and upgrade health and medical systems remotely using this internet access while physicians control or modify settings. Some of the

devices host web-based UIs to interact with users locally or over the network.

We investigate the effects of running stealthy computation on Baxa ExactaMix. The Baxa ExactaMix is an embedded health and medical system that mixes total parenteral nutrition and other multi-ingredient solutions. The compounder runs Windows XP Embedded 2002 Service Pack 2 and has a 664 MHz VIA C5 x86 CPU with 496 MB of memory.[213]

5.10.1 Experimental Setup

Since our Baxa ExactaMix is running Windows XP Embedded 2002, it has Internet Explorer version 6.0, which does not support HTML5 APIs. However, since the Baxa ExactaMix can access the internet, we can install a modern browser. We installed Firefox 29 at the time of this experiment. We note that modern medical systems use more recent operating systems and thus support Web Workers without installing a third-party browser.

In our experiment, we first start the Baxa ExactaMix and wait for it to run its clinical software. We then begin measuring the CPU, memory, and swap usage of the device to establish a baseline of activity. Next, we launch Firefox and navigate to a website that we control. This website uses a worker to perform our stealthy computation, specifically, the DoS attack we describe earlier in Section 5.5. We continue our measurements for 3 minutes.

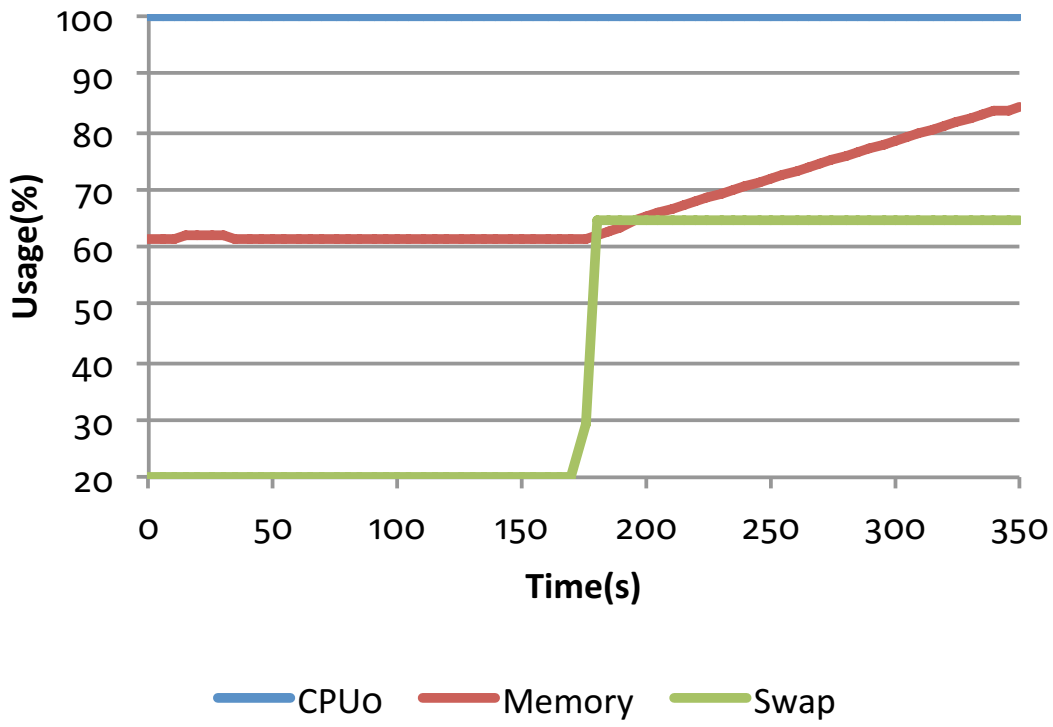


Figure 5.6: Stealthy computation on Baxa ExactaMix.

5.10.2 Results

We note a clear delineation between pre- and post-worker computation in Figure 5.6. Memory and swap usage are at 60% and 20%, respectively, when the Baxa ExactaMix first starts. As this is a single-core device, the CPU utilization remains high for the entire experiment because all processes are scheduled to execute on the same core. We note linearly increasing memory usage and a near-instantaneous spike in swap usage to 60% when we visit our website that performs the stealthy computation.

We also quantitatively evaluated the impact of stealthy computation on the Baxa

ExactaMix. We define the qualitative metric as a measure of the device's usability; specifically, if the Baxa ExactaMix becomes noticeably unresponsive. Repeating the experimental setup, we attempt to use the clinical software. We measure at a coarse granularity the time it takes for the clinical software to output a report about its configuration. With no stealthy computation performed, report generation takes approximately 5.5 seconds. The execution time increases by a factor of two with stealthy computation.

5.11 Linux Stealthy Computation

We experiment with stealthy computation and other operating systems. We find that Chrome and Firefox in Ubuntu 15.10 both allow stealthy computation using web workers. However, the DoS attack against OS X does not apply here. See Figures 5.7 and 5.8.

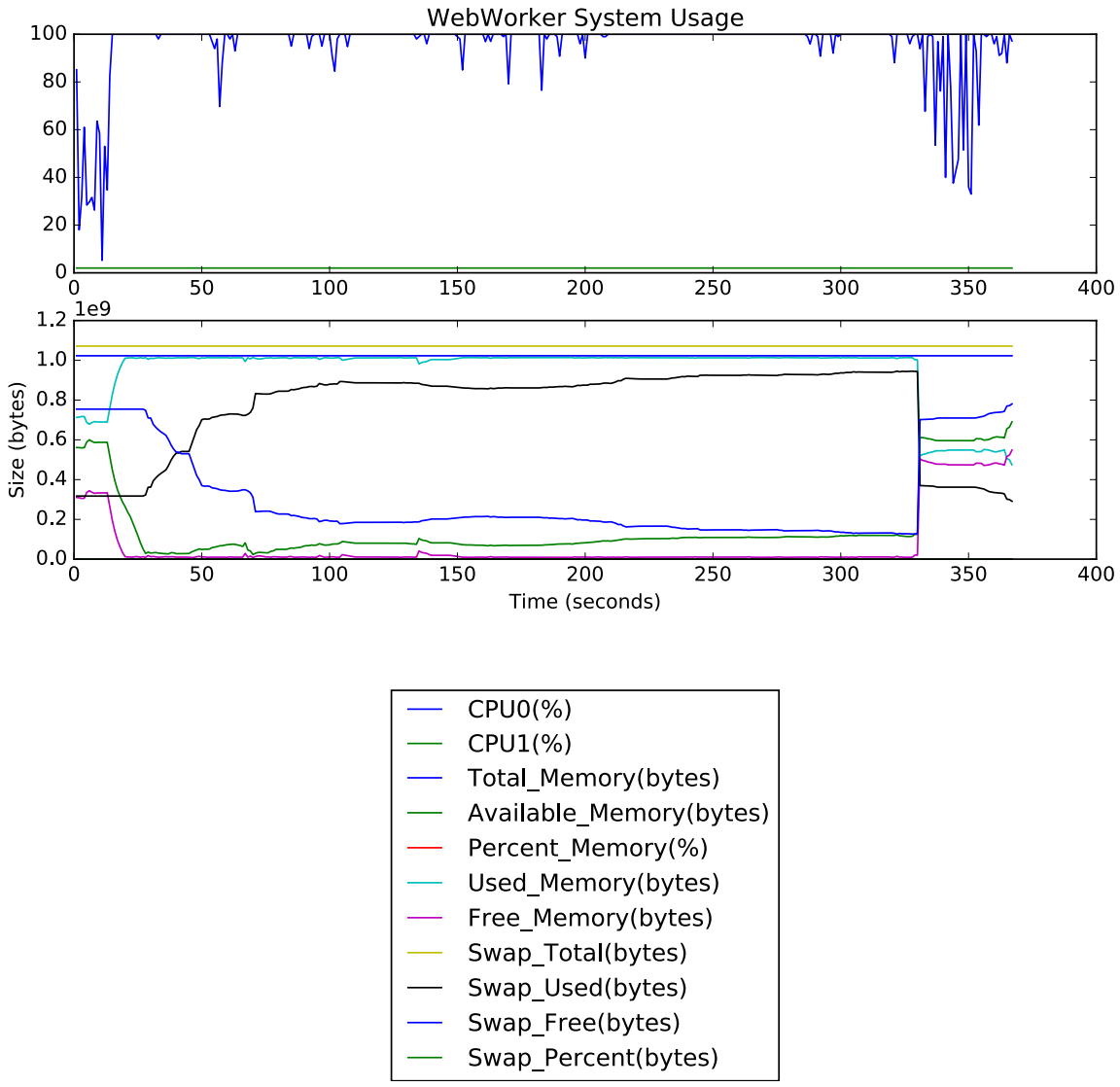


Figure 5.7: Stealthy computation on Ubuntu 15.10 using Chrome.

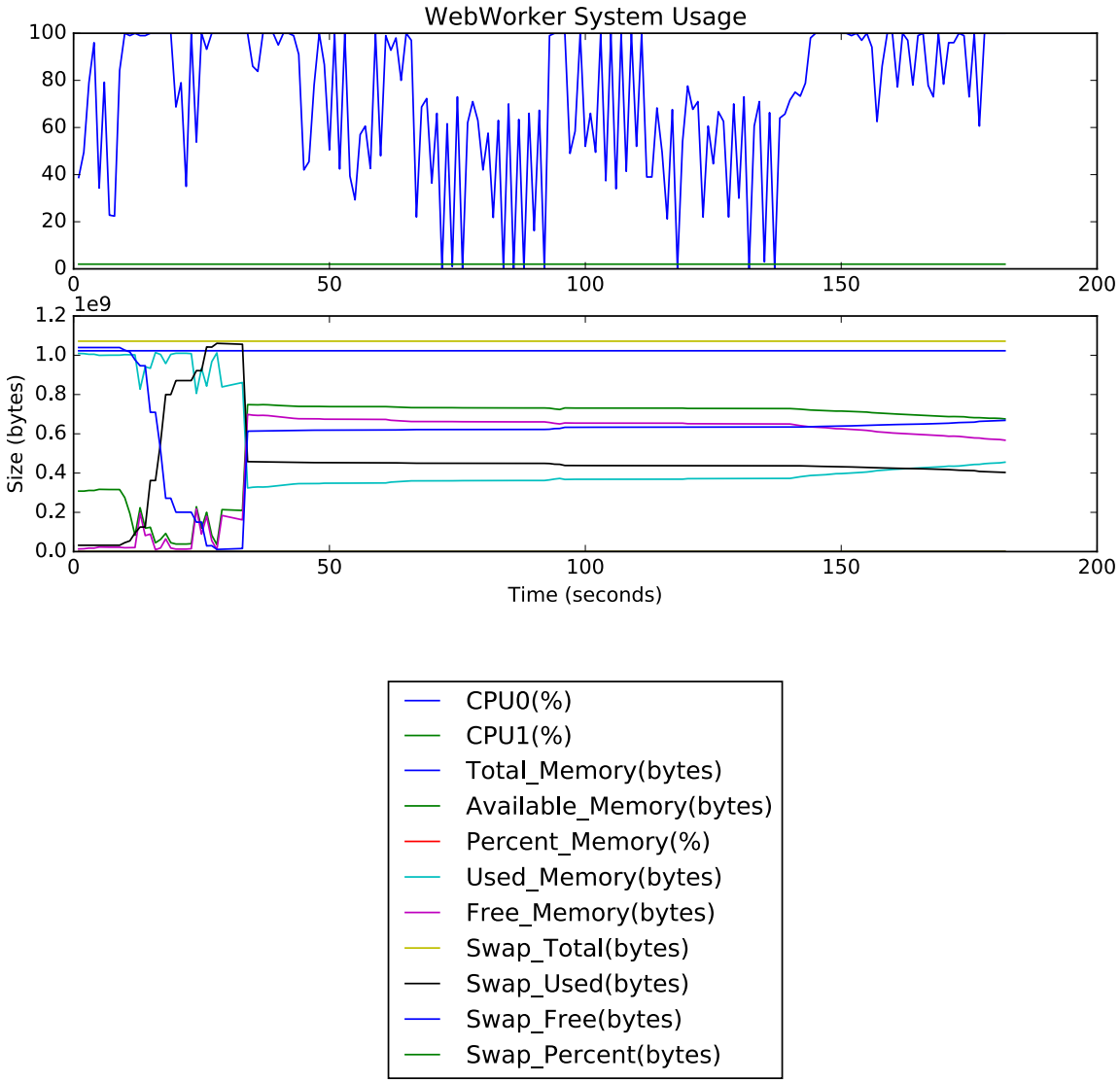


Figure 5.8: Stealthy computation on Ubuntu 15.10 using Firefox.

Chapter 6

KBID: Kerberos Bracelet Identification

The use of modern computer systems almost always requires that a user proves their identity through some process of authentication. A user can authenticate using methods such as public key authentication, biometrics, and passwords; something you have, are, or know, respectively. Password-based authentication remains the most widely used option for authentication because of its ease of use and simple design.

However, passwords have a human factor weakness as users often choose passwords that are too simplistic and easily guessed.[214] Administrators and systems require users to select more complex passwords as a consequence; thus, decreasing user satisfaction as password selection becomes seemingly difficult. Worst yet, complex passwords

interfere in critical workflow such as clinical care where a patient's need is most urgent.

In this chapter we describe an authentication system that requires the user to enter a password as infrequently as once a day. Specifically, authentication information is stored on a wearable device, a bracelet in our case, and is transmitted to devices to which the user wishes to authenticate. The transmission between the bracelet and device is achieved via using the user's body as a communication medium.[62,215] Our goal is to reduce the impact on user satisfaction and workflow (i.e., usability) by removing most of the difficulty of using a complex password.

While we focus on authentication in the medical community use case, we anticipate that other areas such as the financial sector may benefit from our system. In addition, it is important to note that this system is not a two-factor authentication solution. The bracelet is not a biometric component and does not provide any additional information outside of what it stores. It is meant to enhance the user experience and encourage the use of complex passwords.

6.2 Background

KBID originates from the idea of integrating a wearable device to achieve some additional property in an authentication system (e.g., de-authentication). In particular, we are inspired by the design of zero-effort bilateral recurring authentication

(ZEBRA) by Mare et al.[216] In ZEBRA, a user wears a bracelet that encapsulates a wireless radio, accelerometer, and gyroscope; these components record and transmit wrist movements to a currently used computer system. This system continually compares received measurements to input it receives from its keyboard and mouse. If these two measurements are not correlated, the current session is de-authenticated.

At the time, ZEBRA was only envisioned as a method to de-authenticate a user from a computer system and did not include a way to authenticate the user to the system. Assuming that the user has already accepted wearing a device that will effectively de-authenticate them, adding functionality to authenticate rapidly increases the usefulness of the system.

We avoided using radio frequency (RF) emissions for two reasons. First, RF by its nature emits information into the environment. That information, once emitted, can be received by various means. Second, RF relies on the underlying communication being secure. If an attacker discovers a security flaw in an RF communication framework, e.g. Bluetooth low energy (BLE),[217] then the systems as it exists could be vulnerable to the flaw. Specifically, there is no need to alter or even monitor the information exchanged between the system and a wearable device. An attacker would only need to extend the range of the wireless communication to gain access to the system.

We instead use body-coupled communication (BCC), or transmission of information over the human body as a medium. We are not the first to use BCC to transmit

a secret. For example, Chang et al. introduce a system for key exchange over a body area network.[62] By applying a small voltage to the tissue of a dead mouse, they were able to communicate at a rate of 5Hz or 5 bits per second. However, this data rate is not acceptable for our work as we would need to communicate authentication data of at least 256 bits, and this would take nearly a minute to transmit.

We designed our authenticated bracelet to be non-transferrable (i.e., authenticating and then giving the bracelet to someone else). To support this feature, we zero all authentication information upon bracelet removal.

6.3 Related Work

In addition to the ZEBRA, which we have described previously, there have been several previous attempts at developing wearable-authentication technology. Two of note include the Bionym Nymi[218] and the Intel Authentication Bracelet.[219] The Bionym Nymi is an authentication wristband that broadcasts a digitally signed authentication signal derived from a user's heartbeat to nearby devices using BLE.[220] The Intel Authentication Bracelet requires a user to log in to a system with a standard password. A credential is then transmitted to the bracelet using BLE. This credential is then broadcast to nearby bracelet-enabled devices in order to allow password-less login.

6.3.0.1 Limitations of Existing Work

Existing authentication wearables use wireless communication technology (typically BLE) to broadcast their authentication credentials. Thus the devices are likely vulnerable to ghost-and-leech attacks.[221] Ghost-and-leech attacks occur when an attacker uses a more powerful radio transmitter than the transmitter found on a wireless device in order to capture and rebroadcast the wireless signal in order to fool a target into believing that the wireless device is in closer proximity to the target than it actually is.

6.4 Threat Model

We describe KBID as an authentication mechanism that requires a wearable device that transmits short-range authentication data via BCC. We recognize confidentiality, integrity, and availability as security goals specific to KBID. Specifically, data stored on the device and transmitted to the authentication module should be kept secret from and not modifiable by unauthorized entities, and the data should be accessible to both bracelet and authentication module. We omit the privacy of the authenticating user because the system must validate her access to the system or resource.

Adversaries are typically distinguished based on their goals, capabilities, and relation to a system. We define the following adversarial classification criteria for KBID: active adversaries that can read, modify, and inject communication between the device

and authentication module, and passive adversaries that can eavesdrop on the communication; internal entities that have legitimate access to the device, and external entities that do not; single or coordinated group entities, and; sophisticated adversaries with specialized equipment (e.g., high-gain directional antenna or unauthorized authentication module), and unsophisticated adversaries with common equipment.

The KBID device or authentication module may both be used as attack surfaces. For example, an adversary may disrupt KBID authentication by physically damaging the wearable device or authenticator module. We classify this and other KBID security threats into the following categories: BCC threats, whereby the adversary can passively eavesdrop on communication, or actively jam, replay, modify, forge, or drop communication; hardware threats, whereby the adversary can induce incorrect outputs from a valid device, and; software threats, whereby the adversary can alter the logic of KBID's device, authentication module, or client software through software vulnerabilities.

As a hardware and software solution, the threat model for KBID includes many subjects that apply to any such system. These include attack types (denial of service, message forging or tampering, hardware tampering, and others) as well as a study of potential adversaries and other topics. Here we focus on two threats that are unique to KBID.

As a hardware and software system, KBID has two unique threats that both require an active adversary to be within close physical proximity of KBID and an

authentication module. First, it is possible to impersonate an authentication module. An attacker could use a counterfeit authentication module that issues *Get Status* commands when a user touches something connected to it, e.g. a doorknob. Second, while we are using body coupled communication to transmit data without emitting RF, it could be the case that the user's body acts as a broadcast antenna and emits the data into the environment. An attacker then intercepts the data.

6.5 Design

Here we describe in detail the design and implementation of the KBID system. First, we discuss the high-level design where we explain the four major components of the system. Next, we discuss the interface designs and the communication protocols between the major components. Finally, we discuss the system workflow.

6.5.1 High Level Design

The system is composed of four main parts: a bracelet (Figure 6.1), an authentication module (Figure 6.2) an authentication client, and a Kerberos authentication server. The bracelet is a wearable device that fastened to the user's wrist. The bracelet makes contact with the user's skin and applies a signal directly to the user's skin. The authentication module has a sensor with a button under it. When the user touches the sensor and depresses the button, the authentication module initiates

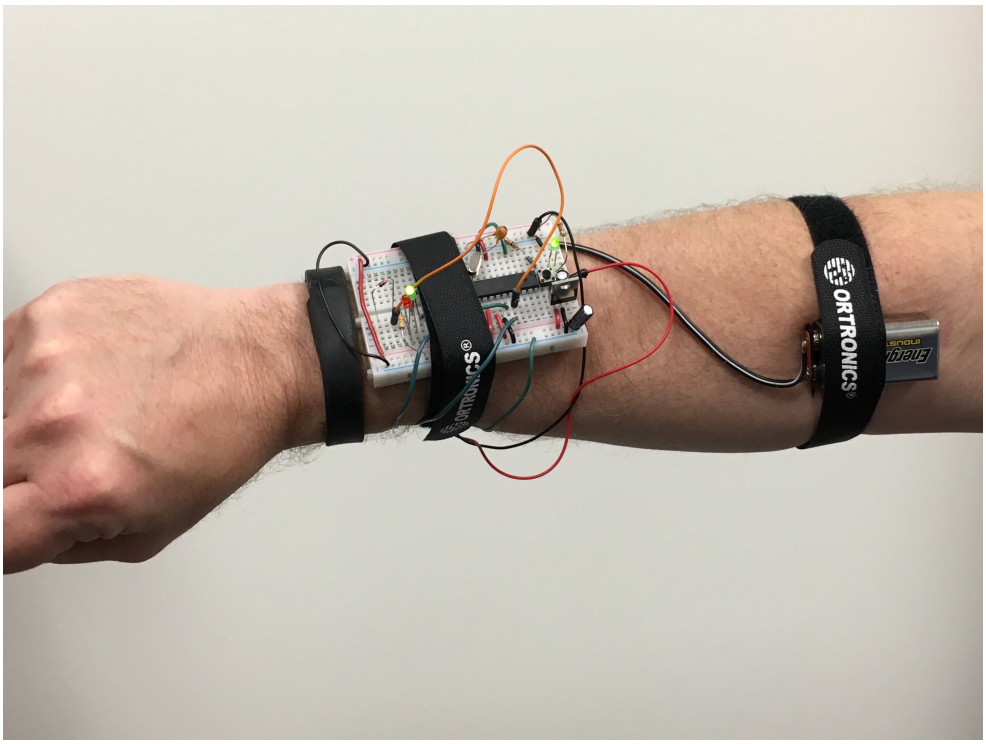


Figure 6.1: KBID Prototype Bracelet.

communication with the bracelet. An RS-232 serial cable attaches the authentication module to the computer system which the user wants to authenticate. A workstation hosts the authentication client. The client monitors the serial connection for data and when necessary, opens a connection to the Kerberos server for authentication. Finally, the Kerberos server is a default installation and uses the default implementation of the authentication protocol.

6.5.2 Interfaces and Communication

The KBID system includes three interfaces. The interface between the bracelet and the authentication module takes place over the user's skin. The interface between

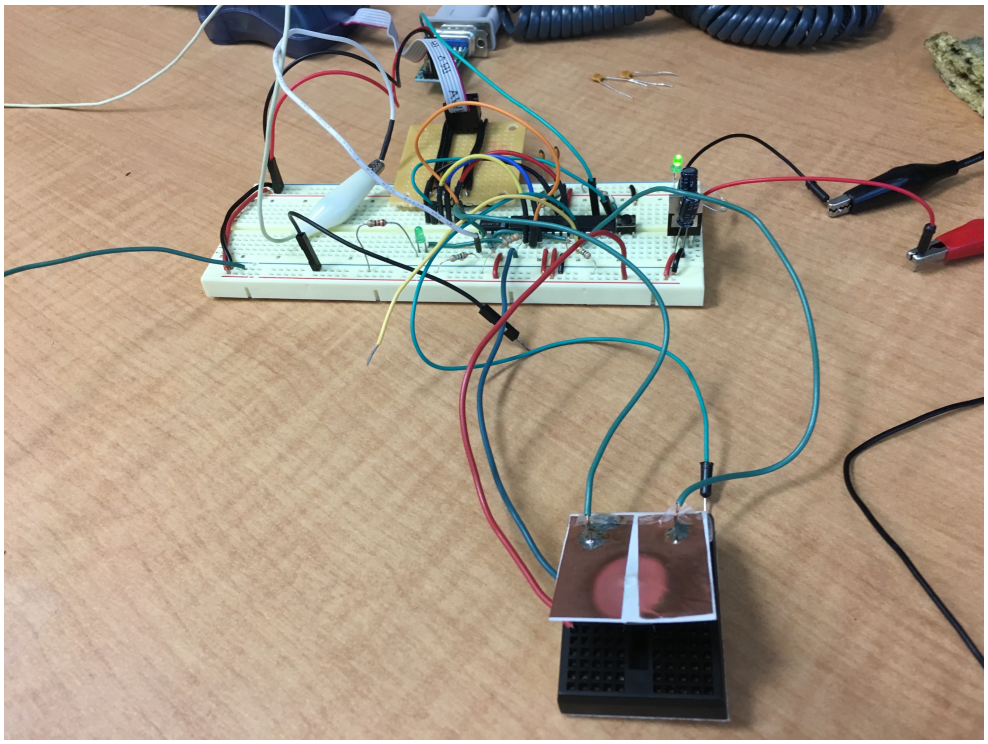


Figure 6.2: KBID Prototype Authentication Module.

the authentication module and the authentication client takes place over RS-232 serial. Finally, the interface between the authentication client and the Kerberos server uses the network. As Kerberos server and client communication is well documented, we will not discuss it in this chapter.

6.5.2.1 Bracelet to Authentication Module

The communication protocol between the bracelet and the authentication module is a lightweight protocol. The messages that the bracelet sends to the authentication modules are called *statuses*. Messages that the authentication module sends to the bracelet are called *commands*. Each message sent over this interface is a length

delimited series of bytes. A status message had the following structure: [Status ID] [Device ID] [Data Size (in bytes)] [Data]. The bracelet will send one of two statuses, *authenticated* or *un-authenticated*. Authenticated data is transmitted in the data field if and only if the status message returns authenticated.

A command message has the following structure: [Command ID] [Device ID] [Payload Size (in bytes)] [Payload]. The authentication module will send three commands: *Get Status*, *Set Token*, and *De-authenticate*. A Get Status command causes the bracelet to respond with a status message. A Set Token command causes the bracelet to store the payload in memory as authentication data and set its status to authenticated. A De-authenticate command causes the bracelet to clear any token it has and set its status to un-authenticated.

6.5.2.2 Authentication Module to Authentication Client

The authentication module and the authentication client communicate status and command messages as well. The authentication module can send three statuses to the authentication client. First is the *Un-authenticated Bracelet* message. This message is sent to the authentication client when the authentication module receives an un-authenticated status from a bracelet.

Next, the authentication module can send an *Authenticated Bracelet* status to the authentication client. It will send this status when the bracelet sends a status of authenticated. The Authenticated Bracelet status will contain the ticket information

that was in the token section of the bracelet's message.

Finally, the authentication module can send a *Ticket Written* status to the authentication client. This is a message that lets the client know that the ticket information has been successfully written to the bracelet.

The authentication client sends two commands to the authentication module. First the *Write Ticket* command. This command instructs the authentication module to pass the ticket included in the command to the bracelet with a Set Token command. The authentication client can also send a *De-authenticate Bracelet* command. This command instructs the authentication module to issue a De-Authenticate command to the bracelet.

6.5.3 System Workflow

We describe the system workflow in two use cases. For the sake of brevity, we do not include any error handling. In the first use case (Figure 6.3) the user is wearing a bracelet but the bracelet is not yet authenticated. The user touches the sensor on the authentication module; the authentication module sees that the user's bracelet is not authenticated and relays this information to the authentication client. The client prompts the user for their username and password. The client verifies this information with the Kerberos server, then instructs the user to touch the sensor on the authentication module again. The client then instructs the authentication module to write the ticket to the bracelet. The client unlocks the workstation once the ticket

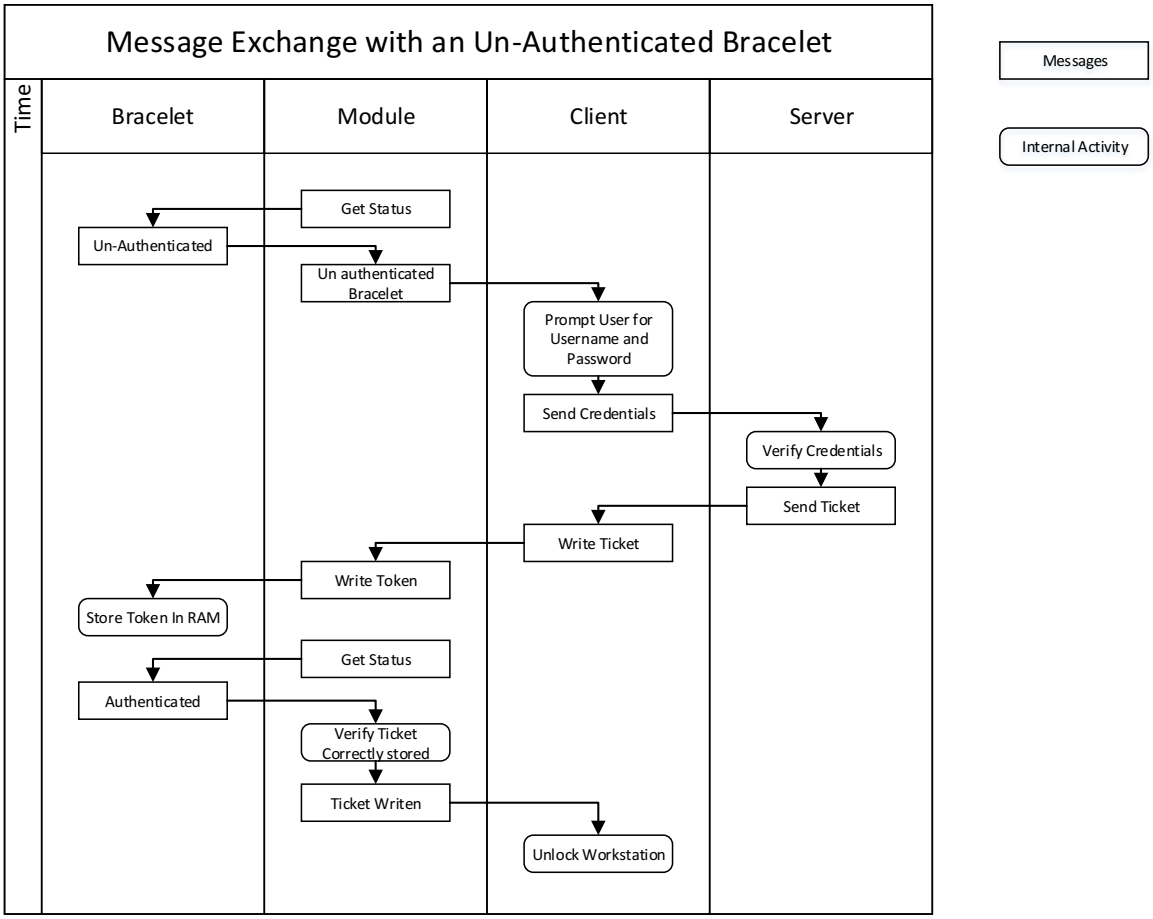


Figure 6.3: Un-authenticated Message Exchange.

is written.

In the second use case (Figure 6.4) the user has an authenticated bracelet. The user touches the sensor on the authentication module. The module asks for a status, and the bracelet provides it with the token it has stored. The module passes this information along to the client which interprets the token as a Kerberos ticket. The client verifies the ticket with the Kerberos server and unlocks the workstation. Our goal is to perform this use case in less than one second.

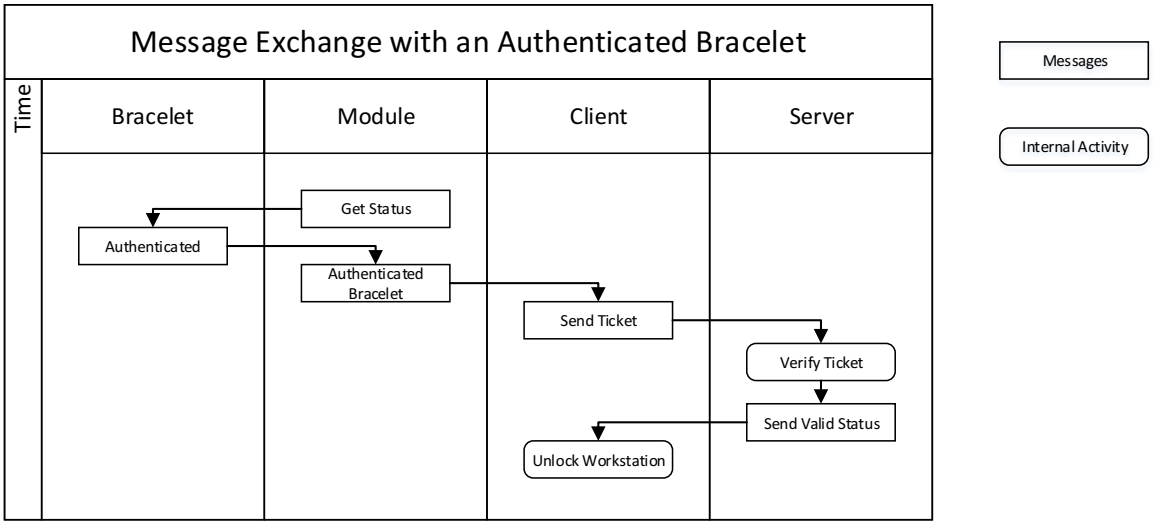


Figure 6.4: Authenticated Message Exchange.

6.6 Experiments and Results

6.6.1 Prototype

The hardware prototypes for the bracelet and the authentication module are based on the Atmel ATMega328 microcontroller operating at 20 MHz and an LM358AN Amplifier. Both the bracelet and the authentication module have copper pads that make contact with the user's skin. The signal from the skin is fed into the amplifier and the signal to the skin is driven by setting a pin on the microcontroller. We also built a resistive analogue to represent the resistance from a user's wrist to their fingertip. The authentication client is written in Python.

6.6.2 Results

Initial results are encouraging. We can send commands from the authentication module to the bracelet. The bracelet can correctly interpret those commands, and it responds when issued a Get Status command. The time elapsed for a Get Status command and a status message with a 256-byte token is approximately 500 milliseconds. We also implemented the functionality that clears the authentication information when the bracelet is removed. We do this by using one of the hardware interrupts on the microcontroller.

6.7 Future Work

We encountered two major hurdles during the development of the first prototype. First, to successfully send a signal, the bracelet, and the authentication module must have a common reference for voltage. Second, while we have been able to get the signal to transmit from the authentication module to the bracelet, we have not been able to get the signal to travel in the opposite direction. In particular, we need the signal to be interpreted by the authentication module. To solve both of these issues, we plan on using capacitive coupling to transmit the signal over the user's skin.

We also plan to implement the system using a microcontroller that can store larger keys. The Atmel ATMega328 only has 2 kilobytes of ram. Since the system requires some memory to perform general operations, the remaining memory to store a key is

approximately 1 kilobyte. This memory size is not sufficient to store authentication information in real world environments. We also plan on hardening the system by adding pre-shared message authentication codes (MACs) to protect against replay attacks.

6.8 Conclusion

Complex passwords interfere in critical workflow such as clinical care where a patient's need is most urgent. In this work, We described an authentication system that requires the user to enter a password as infrequently as once a day. We implemented this authentication system using a bracelet and contact-based authentication module. The bracelet stores authentication information, and the authentication module receives this information to authenticate a user to a given device. Our authentication system reduces the impact on user satisfaction and workflow (i.e., usability) by removing most of the difficulty of using a complex password.

Chapter 7

Applications of Secure Location Sensing in Healthcare

Tracking and managing assets in real-time are critical for large organizations such as Hospitals. For example, "more than [one-third] of nurses spend at least 1 hour per shift searching for equipment and the average hospital owns 35,000 inventory SKUs and utilization hovers around 32-48%, with nearly $4,000 of equipment per bed, lost of stolen each year".[222] Moreover, tracking needs to be secure; specifically, it needs to be resilient to active and passive attacks that aid in the misappropriation of assets. We implement a real-time tracking system using low-cost Bluetooth Low Energy (BLE) devices that provide authenticated wireless communication to track securely assets and people.

We track assets in our system with an external device that can receive BLE trans-

missions containing location data[1], and send location data to a trusted server via Wi-Fi. We implement a device we call *Beacon+* to broadcast location data via BLE. This type of BLE beacon extends the design of Apple's popular iBeacon specification[223] by modifying the advertisement, or unidirectional broadcast, to contain a monotonically increasing sequence number and message authentication code (MAC).

In particular, the sequence number provides temporal freshness that is resilient to clock skew without synchronization. The MAC authenticates the Beacon+ to a trusted server, where the trusted server maintains the absolute location of each Beacon+. Upon receiving the Beacon+ advertisement, the server updates the location of an asset.

We use the real-time tracking system as a foundation for secure location sensing applications. One such example is access control that enforces location-based restrictions. This application relies on the authenticity of received Beacon+ advertisements to compute the relative location to an asset and provide access to asset data if and only if the accessor (i.e., the person who requires the data) is within close physical proximity. Location here is only one factor in a multi-factor access control scheme. For example, nurses and physicians who are away from their personal computer but moving around with a hospital-issued tablet must log in to the tablet with their credentials and be within close physical proximity of a patient to access her medical record.

[1]Assets that support BLE do not require an additional device.

Another secure location sensing application we describe is BCMA physical proximity enforcement. BCMAs typically involve scanning barcodes on patients and medications to interface with electronic records. Koppel et al.[224] identify 31 unique causes where healthcare professionals use workarounds to BCMA processes that they consider impractical (e.g., time). However, these workarounds can result in the wrong administration of medication which impacts patient safety. Therefore, physical proximity enforcement can integrate BLE receivers into scanning devices and require the user to be in an approved location to enable scanning.

The linchpin of our applications is Beacon+. To build a secure and interoperable Beacon+ device we require the following capabilities: (1) perform symmetric key operations; (2) modify advertisement fields; (3) transmit unidirectional advertisements, and; (4) retain traditional beacon (e.g., iBeacon) advertisement structure. We are aware of only one similar, authenticated beacon called Trusted Beacon (TB).[225] Beacon+ differs from TB in its choice of cryptographic primitive and number of advertisements for a single transmission. Specifically, TB lacks (1), (2) and (4).

Moreover, TB uses a weak, factorable[226,227] 320-bit asymmetric RSA private to sign a random value that is valid for 5 minutes. An attacker can, therefore, replay a capture advertisement for up to 5 minutes. In contrast, Beacon+ uses a 128-bit symmetric AES key to compute a MAC on a monotonically increasing sequence number that is only valid for 1 second. Beacon+ conforms to the iBeacon standard because it fits in a single advertisement whereas TB requires multiple advertisements

(i.e., the signature is longer than the message to be signed).

7.1 Background

While prior work exists for the design of location-based access control protocols,[228–230] there has, to the best of our knowledge, been little work done regarding their implementation and evaluation. Existing technologies such as RFID,[231] GPS,[232] and WiFi[222] have had varying levels of success on tracking and managing assets. In this section, we will explore the functionality of these technologies, and discuss how their limitations necessitated Beacon+.

7.1.1 Radio Frequency Identification

Radio Frequency Identification (RFID) provides short-range asset tracking using *tags* and *readers*. Readers interrogate tags and receive unique identifiers along with other data, and typically placed at ingress and egress points of a particular area.[233] The readers then read all tags entering or leaving the monitored area. Communication range for RFID is limited to tens of centimeters, and different bands of RFID communication (low frequency, high frequency, ultra high frequency) can increase the range up to 12 meters.[234] However, higher frequency RFID requires expensive antennas to extend the range. Deploying these antennas throughout an extensive area is impractical and can be considered unsafe depending on hospital RF safety policies.

7.1.2 Global Positioning System

Global Positioning System (GPS) is a reliable global satellite system for providing time and location information to any receiver with a clear view of at least four satellites. GPS is well-suited to outdoor tracking applications, but it does not function well when there is no direct line of sight to at least four satellites. Thus, GPS is not suitable for establishing indoor positioning[235] because it is often not accurate enough within buildings.

7.1.3 Wi-Fi

Wi-Fi facilitates wireless networking over mid-ranged distances. Multiple wireless access points are often used to provide coverage to large areas.These access points each have unique identifiers that bind to specific locations. Therefore, an administrator could track the location of individual clients by observing the order and location in which the clients connect with access points over a given period.

Wi-Fi meets the accuracy, timeliness, and communication range requirements for indoor position management and tracking. Previous work has looked at using Wi-Fi tags for exactly this purpose.[236] One of the benefits of Wi-Fi-based solutions is easy adoption; Wi-Fi tags are attached to devices or staff and communicate with existing access points. However, adhesive Wi-Fi tags are not securely integrated with the devices they manage as tags can be mixed up or maliciously removed. Also, Wi-Fi is

not as power efficient as other technologies, it requires an additional layer of management (e.g., password, SSID, etc.), and it requires bidirectional communication that increases the attack surface. For example, an attacker can continuously communicate with the Wi-Fi device, attempting to authenticate and gain access.

7.1.4 Near Field Communication

Near field communication (NFC)[237] was invented for extremely short-range communication, on the order of several inches. Therefore, for the applications considered in this work, NFC is infeasible, as it would require an unreasonable number of NFC devices.

7.1.5 Bluetooth

Bluetooth[238] is a short-range communication protocol supported by most mobile devices (i.e., smartphones and laptops). Bluetooth-enabled devices initiate connections to host devices by entering *discoverable mode* and waiting for a scanning device to make a connection inquiry. The device then responds to the connection inquiry by sending information including a device name and a device class. If the host chooses to connect to the client device, then the two devices go through a *pairing process*.

Bluetooth technology has been used to build tracking systems .[239–242] Previous work has generally used older Bluetooth versions (older than v4.0) and did not con-

sider security as a design goal. Some tracking systems required tracked entities to establish connections with Bluetooth infrastructure devices resulting in two-way communication with potentially *untrusted entities*.[242]

Beacons. Nokia introduced Bluetooth low energy (BLE) in 2004 as a wireless personal area network that later integrated into the Bluetooth 4.0 standard in 2010.[243] BLE uses significantly less power than classic Bluetooth, and BLE devices can advertise information to a host device (*receiver* herein) without requiring the host device to pair. Conceptually, BLE is similar to NFC, but it is capable of operating at much longer ranges than NFC. In short, devices that need to broadcast small snippets of data at irregular intervals use BLE.

Beacon is one implementation of BLE. A beacon is an inexpensive BLE device (in the range of \$5[244] to \$30[245]) that repeatedly broadcasts a fixed unique identifier. Applications interpret these identifiers for a variety of purposes. For example, Apple's iBeacon[246] broadcasts what it calls an advertisement. The packet structure of an advertisement reveals a tuple of fixed identifiers that are interpreted by as coupon data.

Beacon+ bases itself on the iBeacon protocol and thus we adopt their advertisement structure. In particular, this structure is composed of the following fields:[246]

- UUID: a sixteen-byte unique number used to identify all iBeacons in a particular deployment.

- Major: a two-byte number used to identify groups of iBeacons within a deployment from other groups.
- Minor: a two-byte string used to identify individual iBeacons in a particular cluster of devices.

Although previous work has looked at using Beacons for indoor tracking,[247–250] the insecurity of the iBeacon protocol makes it poorly suited for this task in the presence of an attacker.

7.2 Threat Model

We describe Beacon+ as having unspoofable, temporal and authenticated advertisements; as such, we recognize the following security goals unique to Beacon+.

1. *Integrity.* Advertisements should not be modifiable by an unauthorized entity.
2. *Availability.* Advertisements should be accessible.

We omit confidentiality because Beacon+ advertisements contain no private data. Moreover, we do not claim any privacy goals for Beacon+ as the application of tracking relinquishes the privacy of an asset or person inherently.

Attackers are distinguished based on their goals, capabilities, and relation to Beacon+. Thus, we have the following classification criteria.

1. *Active/Passive Attacker.* Active attackers can read, modify, and inject advertisements (i.e., BLE communication). Passive attackers can eavesdrop advertisements.

2. *Internal/External entity.* Internal entities have legitimate Beacon+ access (e.g., hospital administrator).

3. *Single/Coordinated group entities.*

4. *Sophisticated/Unsophisticated Attacker.* Sophisticated attackers have access to specialized equipment (e.g., high gain antennas). Unsophisticated attackers have access to conventional equipment (e.g., BLE sniffers).

An attacker may use Beacon+, the BLE device, smartphone, and the trusted server as *attack surfaces*. For example, an attacker may disrupt Beacon+ advertisements by physically destroying Beacon+ devices, or jamming or dropping advertisements. We classify Beacon+ security threats into the following categories:

1. *BLE interface threats.* An attacker can passively eavesdrop on advertisements, or actively jam, replay, modify, forge, or drop advertisements.

2. *Software threats.* An attacker can alter the logic of Beacon+ through software vulnerabilities.

3. *Application threats.* An attacker can compromise the intended functionality of an application.

Application-specific threats are unique to Beacon+ and non-obvious. For example, an active attacker may attempt to circumvent location-based restrictions by physically moving all Beacon+s to one central location. There exists threats to BLE devices, smartphones, and trusted servers that we do not cover because it is beyond the scope of Beacon+.

7.3 Beacon+

Apple's iBeacon and the majority of other beacons lack authentication and therefore are susceptible to spoofing; i.e., an attacker can advertise another beacons UUID to trick receivers into believing that the beacon is within range. These beacons also lack a mechanism to provide receivers with a notion of time or, specifically, the notion of advertisement generation (temporal freshness).

Beacon+ prevents spoofing by adding lightweight authentication by way of a MAC, and it provides temporal freshness via a monotonically increasing sequence number. Each BLE advertisement has both MAC and sequence number appended to it. This advertisement maintains the single 27-byte payload structure and unidirectional broadcast protocol defined in the iBeacon specification.[223]

Upon initialization, each Beacon+ is assigned a unique identification number that we distinguish from the UUID of regular beacons by labeling it as *ID*, an initial value for the monotonically increasing sequence number, and a secret key that is used to

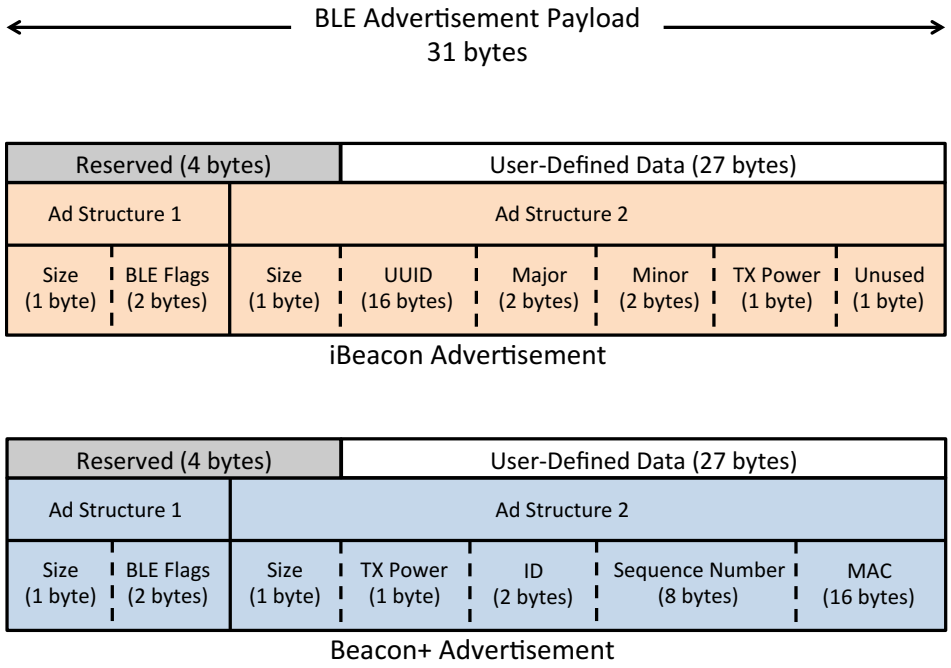


Figure 7.1: iBeacon and Beacon+ advertisement formats. BLE advertisements can support up to a 31-byte payload – 4 bytes are reserved for BLE structures and flags, leaving 27 bytes for user-defined data.

compute a MAC. The secret key is assigned a priori to deployment. As with current beacons, the TX Power (i.e., signal strength) to the Beacon+ at 1 meter in Decibel-milliwatts (dBm) is measured and set. The ID, current sequence number, secret key, and TX Power are stored in non-volatile memory on the Beacon+ to ensure that the values persist even if removing power.

The trusted server maintains both the initial sequence number and the secret key that will authenticate Beacon+ advertisements and check for temporal freshness. Beacon+ computes a MAC on the concatenation of TX Power, ID, and current sequence number with padding. Each second, Beacon+ increments it sequence number, computes a new MAC, and replaces the previous advertisement with the current one.

Figure 7.1 compares the advertisement format of Beacon+ and iBeacon. Beacon+ uses 2 bytes for the ID and 8 bytes for the monotonically increasing sequence number. One restriction of this specific byte allocation is that it supports only 216 or 65535 IDs. We choose to use 2 bytes for the ID in order to allocate 8 bytes for the sequence number.

Beacon+ broadcasts advertisements at a predetermined rate. Faster rates (e.g., eight times per second) improve the likelihood that receivers detect Beacon+ devices in range but increase the power consumption. Slower rates conserve power consumption but may result in receivers failing to detect Beacon+s in range. We configure Beacon+ to broadcast advertisements at a rate of eight times per second (i.e., every 125μs) which matches the rate of iBeacon.

We represent time using monotonically increasing sequence numbers that increment at a regular timeout of once per second. The trusted server maintains the initial and subsequent sequence numbers, and upon receiving an authenticated advertisement, it will compare the received sequence number with the highest seen so far. The advertisement is accepted if the received number is not more than some threshold below the highest seen.

7.3.1 Implementation

We implemented the Beacon+ specification using the Texas Instruments MSP430FR5969 LaunchPad Development Kit[251] and Bluegiga Bluetooth Low Energy BoosterPack for

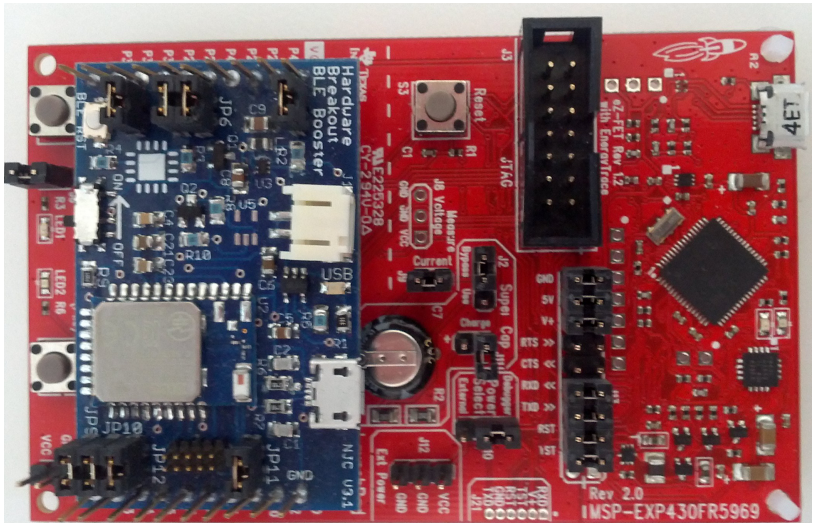


Figure 7.2: Beacon+ is implemented using the TI MSP430 LaunchPad (underlying red board) and Bluegiga Bluetooth BLE BoosterPack.

the LaunchPad[252] (see Figure 7.2). The MSP430 board runs the control logic of Beacon+. During initialization, each MSP430 board is assigned an ID, starting sequence number (usually 1), secret key, and the appropriately calibrated TX Power. We place the MSP430 board at a chosen location in the environment, and we share the ID, starting sequence number, secret key, and chosen location with the trusted server.

Once per the timeout rate, the MSP430 board increments the sequence number, computes the MAC using AES-128 bit CBC-MAC, and sends the new advertisement to the BLE BoosterPack via the UART communication interface. The BLE BoosterPack receives the latest advertisement from the MSP430 and sends it out at a regular interval of eight times per second. The transmitted advertisements are then collected by devices moving throughout the environment and passed to the trusted server for validation (see Section 7.4).

7.4 Applications

Beacon+ serves as a foundation for building many secure location sensing applications. We describe and implement two such applications, namely secure real-time asset tracking and location-based restrictions on access control. We also describe BCMA physical proximity enforcement.

7.4.1 Secure Real-Time Asset Tracking System

The tracking system is composed of three components: (1) Beacon+, (2) BLE–speaking devices that will be tracked (e.g. smartphone or tablet), and (3) backend server (*trusted server* hereon) that validates Beacon+ advertisements and calculates tracked devices' positions. The system is initialized by placing Beacon+s throughout the environment at chosen locations that provide good coverage of the area. This chosen location and the Beacon+'s assigned unique ID, secret key, and starting sequence number is shared with the trusted server, which is run by the system administrator [2]. As per the specification, each Beacon+ periodically broadcasts the authenticated BLE advertisement containing its unique ID, monotonically increasing sequence number, TX Power, and the corresponding MAC of the data.

Tracked BLE–speaking devices periodically collect the authenticated BLE advertisements and corresponding received signal strength (RSSI) from all Beacon+ within

[2]Administer can return to a Beacon+ to refresh keys, apply firmware updates, or even replace it entirely.

range. The device then sends a device update that contains the latest collected Beacon+ advertisements to the server using some other communication medium such as Wi-Fi, cellular, or wired LAN. This device functionality can be added to existing medical devices that support BLE with only a small modification, while older devices can use a BLE module or data collector (e.g., smartphone or computer).

To track personnel, each individual can carry their own smartphone or borrowed hospital-issued tablet. These types of computing devices are increasingly used in health-related environments due to the adoption of health information technology and Bring-Your-Own-Device (BYOD).[253] An App is installed on the devices that collects Beacon+ advertisements and sends them over Wi-Fi or cellular networks to the trusted server.

Figure 7.3 shows an example of two different devices that are tracked. The first device is a physician's iPhone, which can communicate directly to the trusted server. The second device is a heart rate monitor that cannot communicate directly with the trusted server, and relies on a data collection computer to forward communication. In both cases, the devices collect the authenticated BLE advertisements from the Beacon+ within range, aggregate the advertisements and corresponding RSSI values, and send them to the backend server, which will use this information to determine the location of the device.

Upon receiving a device update, the trusted server validates each of the Beacon+ advertisements contained within that update. The trusted server checks, using the

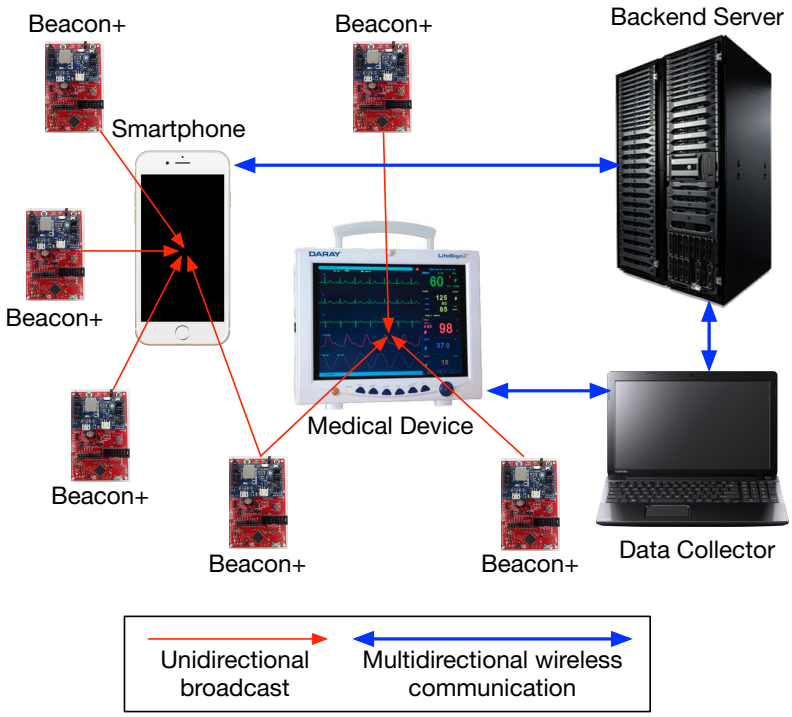


Figure 7.3: Secure Real-Time Asset Tracking System based on Beacon+.

shared secret key for each Beacon+, that the MAC appended on an advertisement matches the computed MAC over the data. If the MAC does not match, that advertisement is discarded and not included in the location calculation. In addition, each advertisement is checked for freshness by comparing the monotonically increasing sequence number on the advertisement with the highest received sequence number received so far from that Beacon+. If the sequence number on the advertisement is not within a valid range of the highest sequence number seen to date (e.g., more than X sequence numbers older), that advertisement is not valid.

After Beacon+ advertisements in a device update are validated, the trusted server can compute the location of the device. Given a device's RSSI value to a Beacon+, the

trusted server can calculate the distance between the two entities using the following equation:

$$rssi = -10n * log_{10}(d) + A \tag{7.1}$$

$$d = 10^{\frac{(rssi \quad A)}{10n}} \tag{7.2}$$

where $rssi$ is the measured received signal strength in dBm, A is the signal strength to the Beacon+ (in dBm) at 1 meter (i.e., the TX Power), d is distance in meters between the Beacon+ and the device, and n is the propagation constant or path-loss exponent (free space has $n = 2$ for reference, this value should be calibrated depending on the environment).

The trusted server can determine the location of the device using trilateration[254–256] given the distance calculation between the device and at least three Beacon+s and pre-existing knowledge of the physical location of each Beacon+. The device is located at the intersection of three circles, one circle centered at each Beacon+, where the radius of each circle is equal to the distance calculated between the device and that Beacon+. In order to track a device's position at all times using trilateration, it must be within range of at least three Beacon+ in order for the computation to succeed at the trusted server.

In addition to computing a device's location, the trusted server continually updates a database, which contains the location of each Beacon+, the location of each

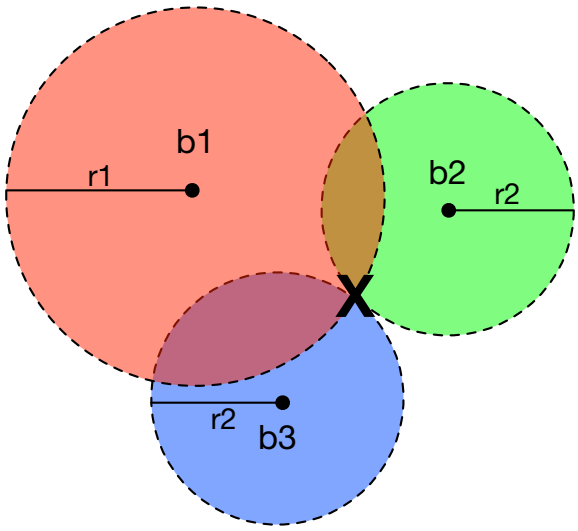


Figure 7.4: Trilateration Example. $r1$, $r2$, and $r3$ (radius of the b1, b2, and b3 circles respectively) correspond to the calculated distance between the tracked device and each Beacon+. The intersection of the three circles (marked by an X) determines the location of the device.

tracked device, acceptable boundaries for each device, and a log of system events. A web application reads the database an displays the location of each Beacon+ and tracked devices, the boundaries of each device, and the system events as they occur in real-time. The trusted server and web application can take action (e.g., raise an alarm, send an email or text message) in response to problematic events, such as when a device has left or is close to leaving the acceptable boundary.

Figure 7.5 shows a snapshot of an example web application that visualizes the location of 10 Beacon+ (blue circles), one device being tracked (solid red block), and the acceptable boundary of that device (red square outline) on a single floor of a university building. The web application enforces access control to ensure that the location of devices (and Beacon+) can only be seen by authorized individuals.

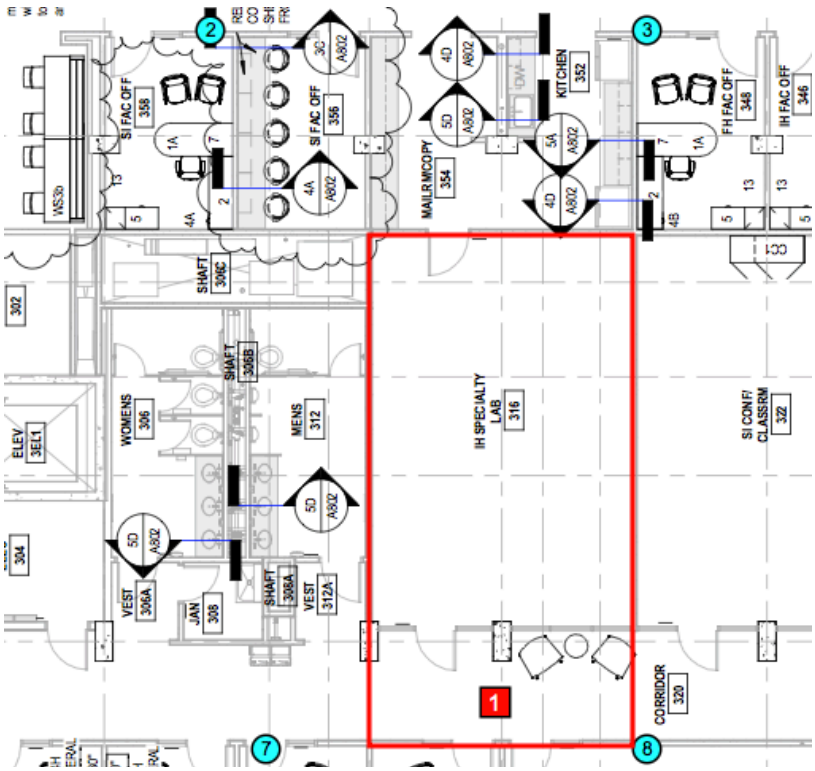


Figure 7.5: Example Web Application Showing Secure Real-Time Tracking System. The blue circles are Beacon+, the solid red block is a tracked device, and the red square outline is the acceptable boundary of that device.

Attack Mitigations. An active attacker may steal a device. However, since devices are tracked in real-time, the appropriate authority is notified if the device moves outside its intended location. The attacker may also physically damage a Beacon+, remove the power source, or perform a sophisticated wireless jamming attack. The tracking system expects Beacon+ advertisements and device updates (i.e., heartbeat) at regular intervals; therefore, the trusted server can implement a detection policy (much like a network intrusion detection system) that generates alerts. Or, the trusted server can generate audit logs for retroactive analysis.

7.4.2 Location–Based Restrictions

Sensitive data such as electronic medical records are protected using encryption and single-factor access control mechanisms (e.g., PIN numbers, passwords) to limit access to authorized individuals. However, this approach raises a major security concern as an attacker that is able to bypass or break the access control security gains access to all of the sensitive data in the database with a single breach. This threat is made worse in the context of a hospital, where computing devices are often used to access sensitive patient information, and a stolen or compromised device can provide an attacker with a large portion of private data.

To address this threat, we implement a prototype application that provides an access control mechanism that enforces location-based restrictions. The application relies on the authenticity of received Beacon+ advertisements to compute the relative location of an authenticated device compared to an asset and provides access to the asset data if and only if the device is within close physical proximity. In the hospital setting, nurses and physicians who are away from their personal computer but moving around with their smartphone must be within close physical proximity of a patient to access her medical record. With this scheme, an access control breach only results in a small fraction of sensitive data leakage, since an attacker that steals an authenticated device only gets access to data that is within proximity. The location is only one factor in a multi-factor access control scheme to authenticate a user.

Implementing the location-based restrictions application requires only minor addi-

tions to the secure real-time tracking system. Personnel can use the same smartphone or BLE device they sign into for the tracking system to access sensitive data. As personnel move about the organization, the trusted server tracks their location. When the tracked device enters the close proximity of assets, the trusted server checks the credentials of the device and authenticity of the Beacon+ advertisements and sends the device the appropriate data from assets in range. Similarly, when devices leave proximity of an asset, the trusted server revokes access to that asset's data and the App removes the record[3]. The trusted server can choose the level of granularity on which to enforce location-based restrictions. For example, in the hospital context, the trusted server may choose to organize patient records based on room, rather than solely using distance as the metric. In addition, the trusted server can tailor the information sent to the devices based on the credentials of the user (e.g., physicians may be sent more sensitive information about a patient than nurses).

This approach provides location-based restrictions without the need of additional authentication at every step. While an attacker that steals one of these authenticated devices can see the sensitive information about nearby patients, the threat is not much different from the existing accepted threat in which an attacker could walk around the hospital and take the paper medical records that often sit unattended outside of patient rooms. One possibility is to have physicians re-authenticate upon entering each room which prevents an attacker from walking around with a device to get basic

[3]The App is also setup to remove data from the display after a configurable timeout, which protects against an attacker that cuts network communication in an effort to force an asset's data to persist on the screen even after moving out of range of the asset.

patient information but puts a burden on physicians and nurses. This is a trade-off between privacy and usability which can be set as desired, and the App supports both configurations.

In some cases, a physician might require accessing more details of a patient's health records or may require accessing a medical record for a patient that is not in the same room. In this case, the App on the device allows physicians to provide further forms of authentication (e.g., fingerprint, additional password) to increase their access. Note that this access is only provided temporarily each time additional authentication is provided, preventing an attacker from breaking the location-based restrictions if she steals the device. Additionally, physicians can always return to their private offices to use traditional access control techniques to gain access to a wider range of medical records.

By using location-based restrictions for access control, hospitals get the technological and convenience benefits of electronic medical records with the traditional privacy model of paper medical records, in that successful attackers only get access to localized sensitive information rather than access to a large database of many records.

Attack Mitigations. An active attacker may perform a denial-of-service attack on the tracking system to cause patient harm or thwart productivity. This attack is mitigated by having authorized individuals use additional authentication meth-

ods to bypass the location-based restrictions and temporarily gain access to patient records, or return to an authorized computer system (e.g., office computer). This type of adversary can also steal an authenticated device (i.e., a physician logged in and misplaced the device) and use it to obtain patient records via the location-based restrictions application. The application mitigates this attack by deleting patient records on a set time interval and when it moves outside the range of patients.

Note that the location-based restrictions application requires that the trusted server have knowledge of patient locations in the hospital (either at a physical location or room-level granularity). The tracking system can be made to track patient locations by associating BLE devices with patients, or the trusted server can link with existing hospital management techniques that track patient locations.

7.5 Experiments

We deployed eight evenly spaced Beacon+ prototypes of one side of the floor in our building to emulate a setup that would be used in typical hospital settings. Each Beacon+ was placed at its chosen location and assigned a unique ID and secret key that is shared with the trusted server. Upon startup, each Beacon+ begins broadcasting an authenticated BLE advertisement containing its unique ID, latest sequence number (monotonically increasing once per second), calibrated transmit power at 1 meter, and MAC. Advertisements are broadcast every 125μs. We experimented with several

values for n, the propagation constant from equation 7.1, and ultimately decided on $n = 2.7$ for our experiments. It provided the most accurate measured distance from compared with the actual location of tracked devices.

We used a Google Nexus 4 smart phone as the tracked device. We created an Android App to periodically scan and collect all Beacon+ advertisements within range (aggregating the measured RSSI values for each Beacon+ ID). The collected advertisements are then bundled into a device update and sent via Wi-Fi to the trusted server, which authenticates each of the advertisements in the update and calculates the position of the device.

7.5.1 Tracking System Accuracy

To measure the accuracy of our Beacon+ tracking system, we placed the device at various locations and compared the calculated location from the tracking system with the actual location in the building. Initially, we measured the accuracy using the trilateration approach, using the measurements from the three Beacon+ prototypes with the strongest received signal strength for that update. However, we found that the measured signal strength from our BLE hardware contained a fair amount of noise, often causing the trilateration calculation to fail (i.e., the resulting circles created from the distance measurements did not intersect). Rather than using trilateration in our experiments, we calculated the position of devices using an approach that is less accurate, but more flexible.

Translated Midpoint Method. For each device update received, the trusted server sorts the valid Beacon+ advertisements in order of received signal strength and can calculate the device's position for this update as long as at least two advertisements are valid. If there are three or more valid advertisements, the trusted server uses the top three Beacon+ ads (based on RSSI values) and forms a triangle, with one vertex corresponding to each of the Beacon+ locations in the environment. Each vertex is then translated toward the midpoint of the opposite side of the triangle, with translation distance proportional (or in our case, equal) to the measured distance between the device and that Beacon+.

If there are only two advertisements, a line is formed between the two Beacon+ locations, and each point is translated toward the other point with a distance equal to the measured distance from the device to that Beacon+. Finally, the device's position is calculated as the centroid of the resulting triangle (in the case of three valid Beacon+ advertisements) or midpoint of the resulting line (in the case of two Beacon+ advertisements). Using the new approach resulted in position calculation with precision 1-2 meters in the best case and 9-10 meters in the worst case.

Using the translated midpoint method, the resulting Beacon+ tracking system is flexible and accurate, providing a position calculation with the precision of 1-2 meters in the best case and 9-10 meters in the worst case. Compared to the trilateration approach, the translated midpoint method achieves a better overall tracking system in the environment of our experimentation.

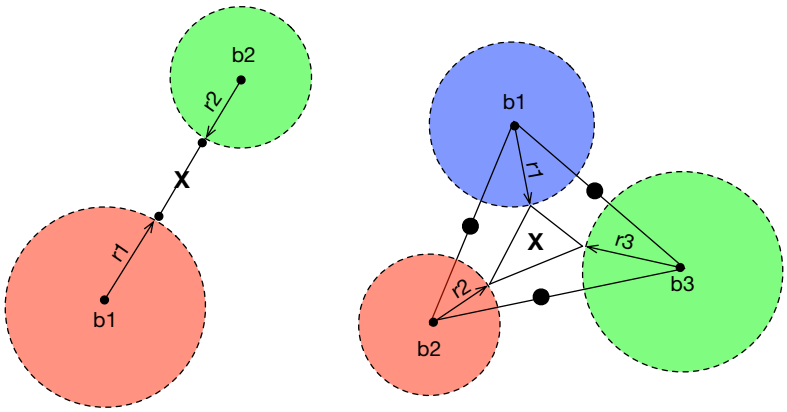


Figure 7.6: Translated Midpoint Method to calculate device position.

7.5.2 Power Consumption

We connected an MSP-430 LaunchPad to an Agilent programmable power supply. Since the MSP430 LaunchPad runs off of a +5V power source, we set the output voltage to 5 volts and maximum current to 1A. Our power supply showed that in the case of the MSP430 emulating an iBeacon, the power draw was between 15 and 20 mA. In the case of the MSP430 emulating a Beacon+, the power draw was between 22 and 25 mA. Therefore, the overhead of Beacon+ over a standard Beacon running on our test platform was between 20% and 46%.

7.5.3 Location-Based Restrictions

We created an Android App that collects and forwards Beacon+ advertisements to the trusted server and displays patient records sent in return. After validating a device and calculating its position, the trusted server compares the device position

with the location of patients in the building and only sends records of nearby patients (10 meters in our experiments). When a device moves out of range of a patient, that patient record is removed from the list in the App.

For this experiment, we created a mock patient record database on the trusted server based on the OpenMRS Demo Data,[257] and set the location of four of the patients in the database to locations in the building environment (yellow squares are shown in Figure 7.9). Then, we walked around the building with the smartphone running the App to view the records of the nearby patients, i.e., the patients that were within 10 meters of the device's tracked position.

Figure 7.9 shows four snapshots (a through d) of the experiment in action. The visual GUI of the Beacon+ tracking system is shown on the right. The GUI shows the location of the Beacon+ prototypes (blue circles), the patients in the building (yellow squares), and where the device is located at each snapshot (a through d). For each snapshot in Figure 7.9, we also include the screen capture of the device running the patient record access App at its respective location.

7.6 No Central Trusted Authority

We assume a central trusted authority (i.e., the trusted server) in our secure location sensing application architecture. However, this assumption is susceptible to an attacker who gains unauthorized access to the trusted server. If this should happen,

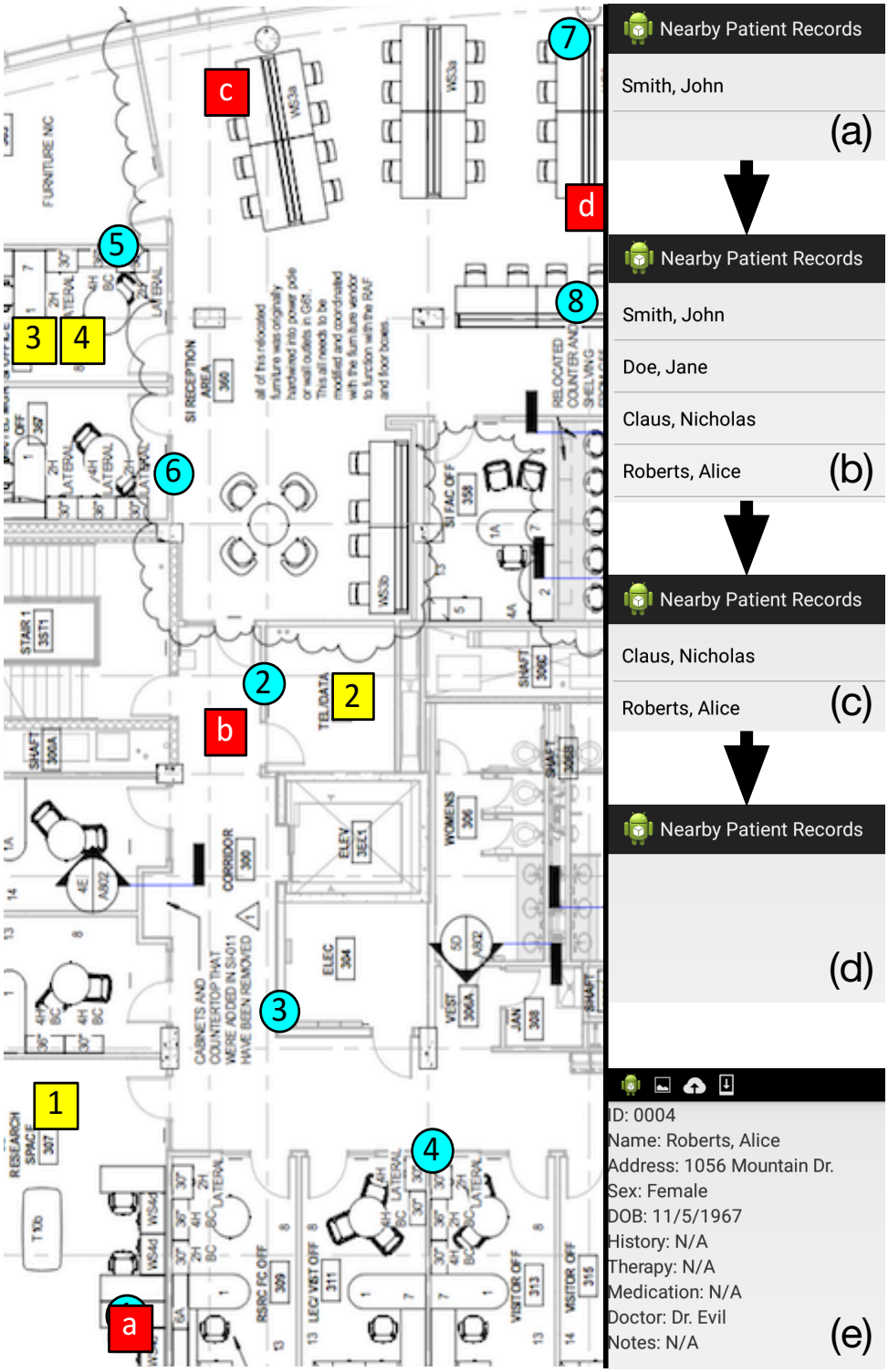


Figure 7.7: Location-Based Restrictions on Access Control.

all security guarantees are invalidated because the attacker would have access to the private keys of every Beacon+. Removing the trusted server is a complicated problem because Beacon+ supports unidirectional communication only; therefore, we cannot use a two-way protocol to assert trust nor can we introduce an out-of-band channel for weak authentication.[71,75]

To remove the trusted server we construct a protocol based on the timed efficient stream loss-tolerant authentication (TESLA) broadcast protocol. This protocol assumes a large set of mutually untrusted receivers in a sensor network with packet loss. A sender computes the MAC t of a message with a key k known only to itself. The sender broadcasts the authenticated message m, t, and some set of receivers buffer the message. Time t later, the sender discloses the key k and the receiver authenticates the packet. This protocol is unlike our previous Beacon+ protocol because it assumes that the sender and receiver clocks can be loosely synchronized. It also introduces hash chains to authenticate keys at the receiver.

A hash chain is generated by selecting a random element s and repeatedly applying a one-way function F. We can verify any element of the chain through commitment s_i by performing $F^{j-i}(s_j) = s_i$, where $i < j$. TESLA uses hash chains to generate authentication values, k from the above, and discloses k at time t (e.g., one key per second).

We design Beacon+'s new protocol without a trusted server as follows. We initially generate a random secret s and unique ID. We then calculate $H_N = H^N(s)$ where

H^N is the hash of s, N times. We put H_N and ID into a digital certificate C and sign it with a certificate authority's private key. C, s, and ID are placed on the Beacon+. At each time period i, the Beacon+ sends an advertisement containing C, ID, a message M containing the value i, a MAC on M computed with the key $H^{N-(i+1)}(s)$, and the value H^{N-i}.

The verifier in this protocol is the smartphone or medical device. The verifier collects advertisements from two adjacent time periods (i initial and j final) and checks that the advertisements are current based on its own internal clock. Next, the verifier validates C and hashes $H^{N-j}(s)$, j times, to obtain H_N. This value is in C, thus, it can be validated. The verifier then verifies time period i's MAC using the key output from time period j. We diagram this protocol in Figure 7.8.

We differ from TESLA in how we do synchronization. Specifically, TESLA requires a digital signature key pair on the sender and a nonce from the receiver. The receiver records the current time and sends the sender a nonce. The sender replies with its clock time and the nonce signed with its public key. Clock synchronization is useful for the receiver because it can check that the key k received has been disclosed yet. Beacon+ is strictly unidirectional, thus, cannot receive a nonce like the sender in TESLA. Instead, the verifier in our protocol can check if two adjacent time periods are current by querying the tracking server described below.

Removing the trusted server in our architecture adds new entities and roles. For example, the secure real-time asset tracking system adds a certificate authority, track-

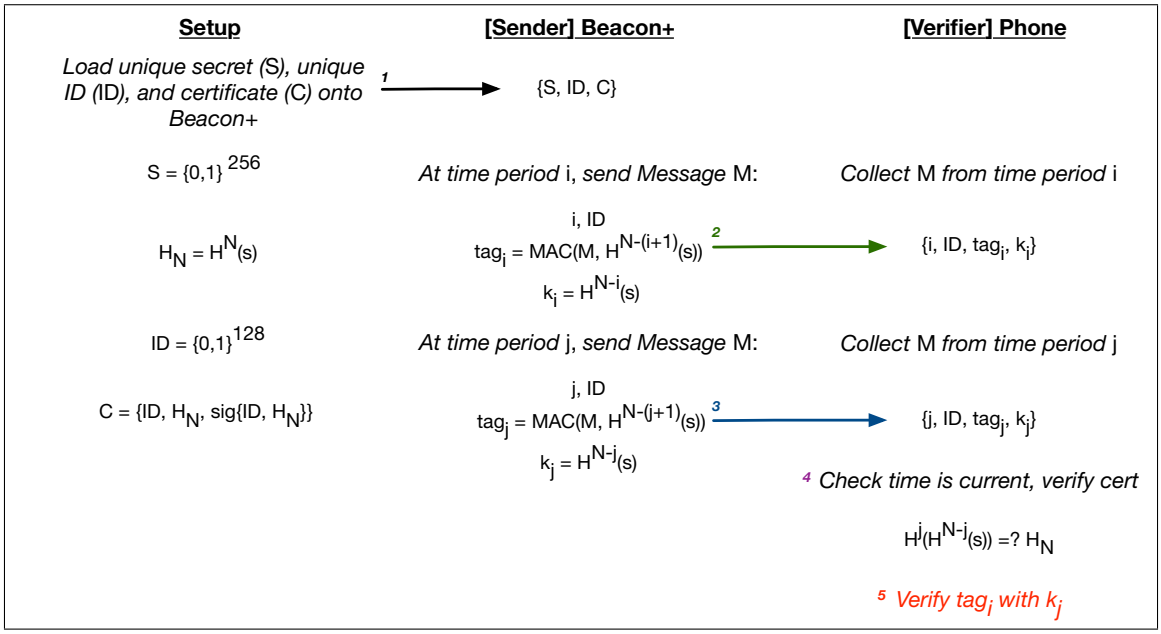


Figure 7.8: Beacon+ protocol without central trusted authority.

ing server, and map authority (i.e., database server). We logically separate the tracking server and map authority because these components could be distributed. The certificate authority issues a signed certificate to every Beacon+. The medical device and smartphone later verify the signature on the Beacon+ certificate when receiving Beacon+ advertisements.

The tracking server allows both the medical device and smartphone to make application-specific queries to the map authority. For example, a medical device would send a location query [4] that contains a set of unique Beacon+ IDs, the latest time period j and k_j where $k_j = H^{N-j}(s)$. The tracking server would verify $H^j(H^{N-j}(s)) = H_N$ where H_N is in the Beacon+ certificate. If and only if verification succeeds and k_j has not been previously seen, the tracking server processes the

[4] We expect other queries such as location-based access queries.

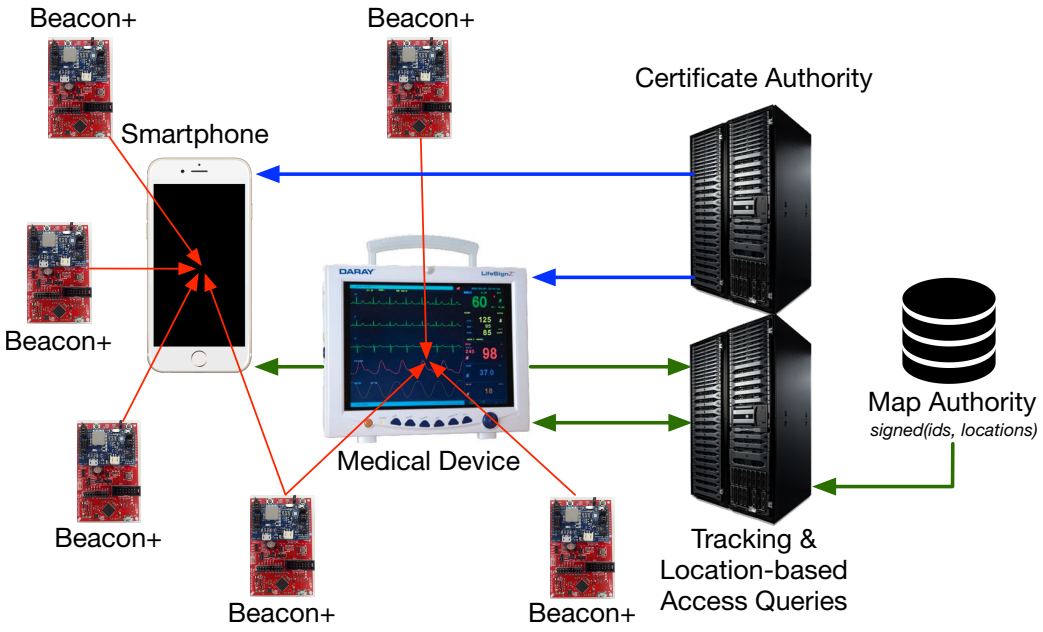


Figure 7.9: Secure Real-Time Asset Tracking System with no Trusted Server.

query using the map authority and returns a result.

There exists an implicit assumption that devices that can verify Beacon+ advertisements are also trusted. We can make this an explicit assumption by requiring mutual authentication between the smartphone or medical device and the tracking server. In this case, only trusted devices can communicate with the tracking server.

7.7 Conclusion

In this work, we have shown that Beacon+ can be used to implement secure location sensing applications that have the potential to improve healthcare processes in terms of security, efficiency, and safety. We implemented a secure real-time tracking system for hospitals that also provides a foundation for a novel application that applies

location-based restrictions on access control.

Chapter 8

Summary

In this dissertation, we have examined a particular set of challenges in health and medical systems and introduced new systems that improve security, privacy, and usability. We presented a survey of related work in embedded health and medical systems that uncover and motivate research challenges. We performed an analysis of Apple iMessage and found significant vulnerabilities that exemplify the challenges of BYOD and protocol development. We showed that memory copying performed by DMA engines could be chained together to perform arbitrary computation on numerous computer systems. We implemented and exploited stealthy computation and covert channels using the HTML5 Web Workers API. Health and medical systems such as patient portals and pharmaceutical compounders are directly affected by this finding. We introduced an authentication system that addresses the problem of complex passwords and poor usability. And, we implemented a real-time tracking

system that uses a custom-built security device called Beacon+. This device enabled other secure location sensing applications such as location-based access restrictions.

Bibliography

[1] J. Carrigan, P. Martin, and M. Rushanan, "Kbid: Kerberos bracelet identification," in *Proceedings of the 20th annual Conference on Financial Cryptography and Data Security*, ser. FC '16, 2016.

[2] M. Rushanan and S. Checkoway, "Run-dma," in *Proceedings of the 9th annual USENIX Workshop on Offensive Technologies*, ser. WOOT '15, 2015.

[3] M. Rushanan, A. D. Rubin, D. F. Kune, and C. M. Swanson, "Sok: Security and privacy in implantable medical devices and body area networks," in *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, ser. SP '14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 524–539. [Online]. Available: http://dx.doi.org/10.1109/SP.2014.40

[4] C. Garman, M. Green, G. Kaptchuk, I. Miers, and M. Rushanan, "Dancing on the lip of the volcano: Chosen ciphertext attacks on apple imessage," in *Submission to the 25th annual USENIX Security Symposium*, 2016.

[5] P. Martin, T. Tantillo, A. D. Rubin, and M. Rushanan, "Applications of se-

cure location sensing in healthcare," in *Submission to the 2nd International Conference on Health Informatics and Medical Systems*, ser. HIMS '16, 2016.

[6] M. Rushanan, D. Russel, and A. D. Rubin, "Malloryworker: Stealthy computation and covert channels using web workers," in *Submission to the 10th annual USENIX Workshop on Offensive Technologies*, ser. WOOT '16, 2016.

[7] M. Astani, K. Ready, and M. Tessema, "Byod issues and strategies in organizations," *Issues in Information Systems*, vol. 14, no. 2, pp. 195–201, 2013.

[8] J. Zhu, L. Chen, A. Chen, G. Luo, X. Deng, and X. Liu, "Fast 3d dosimetric verifications based on an electronic portal imaging device using a gpu calculation engine," *Radiation Oncology*, vol. 10, no. 1, pp. 1–11, 2015.

[9] "Exactamix compounder," http://www.baxtermedicationdeliveryproducts.com/pdf/801686_Electronic_Version.pdf/, Jun. 2013, accessed May 27, 2014.

[10] S. Sinclair and S. W. Smith, "What's wrong with access control in the real world?" *IEEE Security & Privacy*, vol. 8, no. 4, pp. 74–77, 2010.

[11] K. E. Hanna, F. J. Manning, P. Bouxsein, and A. Pope, *Innovation and Invention in Medical Devices: Workshop Summary*. The National Academies Press, 2001.

[12] (2011, Jun.) Insulin pumps - global pipeline analysis, opportunity assessment and market forecasts to 2016. [Online]. Available: http://www.globaldata.com.

BIBLIOGRAPHY

[13] (2011, Jun.) US healthcare equipment and supplies - diabetes. [Online]. Available: http://www.research.hsbc.com.

[14] M. Patel and J. Wang, "Applications, challenges, and prospective in emerging body area networking technologies," *Wireless Commun.*, vol. 17, no. 1, pp. 80–88, Feb. 2010. [Online]. Available: http://dx.doi.org/10.1109/MWC.2010.5416354

[15] G. Asada, M. Dong, T. S. Lin, F. Newberg, G. Pottie, W. J. Kaiser, and H. O. Marcy, "Wireless integrated network sensors: Low power systems on a chip," in *Proc. 24th European Solid-State Circuits Conference (ESSCIRC '98)*, 1998, pp. 9–16.

[16] J. Zheng and M. J. Lee, "Will IEEE 802.15.4 make ubiquitous networking a reality?: A discussion on a potential low power, low bit rate standard," vol. 42, no. 6, pp. 140–146, Jun. 2004. [Online]. Available: http://dx.doi.org/10.1109/MCOM.2004.1304251

[17] X. Zhang, H. Jiang, X. Chen, L. Zhang, and Z. Wang, "An energy efficient implementation of on-demand MAC protocol in medical wireless body sensor networks," in *Proc. IEEE International Symposium on Circuits and Systems (ISCAS 2009)*, 2009, pp. 3094–3097.

[18] S. Ullah, H. Higgins, B. Braem, B. Latre, C. Blondia, I. Moerman, S. Saleem, Z. Rahman, and K. S. Kwak, "A comprehensive survey of wireless body area

networks," *J. Med. Syst.*, vol. 36, no. 3, pp. 1065–1094, Jun. 2012. [Online]. Available: http://dx.doi.org/10.1007/s10916-010-9571-3

[19] A. Kailas and M. A. Ingram, "Wireless communications technology in telehealth systems," in *Proc. 1st International Conference on Wireless Communication, Vehicular Technology, Information Theory and Aerospace Electronic Systems Technology (Wireless VITAE 2009)*, 2009, pp. 926–930.

[20] Code of Federal Regulations, "Title 47 Part 95 Section 401 (e) C.F.R 47, 95.401 (e), Federal Communications Commission - The Wireless Medical Telemetry Service (WMTS)," http://transition.fcc.gov/Bureaus/Engineering_Technology/Orders/2000/fcc00211.pdf.

[21] G. Kolata. (2013, Oct.) Of fact, fiction and Cheney's defibrillator. [Online]. Available: http://www.nytimes.com/2013/10/29/science/of-fact-fiction-and-defibrillators.html.

[22] D. Halperin, T. S. Heydt-Benjamin, B. Ransford, S. S. Clark, B. Defend, W. Morgan, K. Fu, T. Kohno, and W. H. Maisel, "Pacemakers and implantable cardiac defibrillators: Software radio attacks and zero-power defenses," in *Proc. 29th Annual IEEE Symposium on Security and Privacy (SP 2008)*, May 2008, pp. 129–142. [Online]. Available: http://www.secure-medicine.org/icd-study/icd-study.pdf

[23] K. Fu and J. Blum, "Inside risks: Controlling for cybersecurity risks of medical

device software," *Communications of the ACM*, vol. 56, no. 10, pp. 21–23, Oct. 2013. [Online]. Available: http://www.csl.sri.com/users/neumann/cacm231.pdf

[24] W. Burleson, S. S. Clark, B. Ransford, and K. Fu, "Design challenges for secure implantable medical devices," in *Proc. 49th Annual Design Automation Conference (DAC '12)*, 2012, pp. 12–17. [Online]. Available: http://doi.acm.org/10.1145/2228360.2228364

[25] (2013, Jun.) Content of premarket submissions for management of cybersecurity in medical devices: Draft guidance for industry and Food and Drug Administration staff. http://www.regulations.gov/#!documentDetail;D=FDA-2013-D-0616-0002.

[26] A. B. Mullen. (2013, Sep.) Premature enforcement of CDRH's draft cybersecurity guidance. http://www.fdalawblog.net/fda_law_blog_hyman_phelps/2013/09/premature-enforcement-of-cdrhs-draft-cybersecurity-guidance.html.

[27] D. Halperin, T. S. Heydt-Benjamin, K. Fu, T. Kohno, and W. H. Maisel, "Security and privacy for implantable medical devices," vol. 7, no. 1, pp. 30–39, Jan. 2008. [Online]. Available: http://dx.doi.org/10.1109/MPRV.2008.16

[28] C. Li, A. Raghunathan, and N. K. Jha, "Hijacking an insulin pump: Security attacks and defenses for a diabetes therapy system," in *Proc. 13th IEEE Inter-*

national Conference on e-Health Networking Applications and Services (Health-Com 2011), 2011, pp. 150–156.

[29] M. Rostami, W. Burleson, F. Koushanfar, and A. Juels, "Balancing security and utility in medical devices?" in *Proc. 50th Annual Design Automation Conference (DAC '13)*, 2013, pp. 13:1–13:6. [Online]. Available: http://doi.acm.org/10.1145/2463209.2488750

[30] M. Zhang, A. Raghunathan, and N. K. Jha, "Towards trustworthy medical devices and body area networks," in *Proc. 50th Annual Design Automation Conference (DAC '13)*, 2013, pp. 14:1–14:6. [Online]. Available: http://doi.acm.org/10.1145/2463209.2488751

[31] S. S. Clark and K. Fu, "Recent results in computer security for medical devices," in *International ICST Conference on Wireless Mobile Communication and Healthcare (MobiHealth), Special Session on Advances in Wireless Implanted Devices*, Oct. 2011. [Online]. Available: https://spqr.eecs.umich.edu/papers/clark-mobihealth11.pdf

[32] D. Foo Kune, J. Backes, S. S. Clark, D. Kramer, M. Reynolds, K. Fu, Y. Kim, and W. Xu, "Ghost talk: Mitigating EMI signal injection attacks against analog sensors," in *Proc. 34th Annual IEEE Symposium on Security and Privacy (SP 2013)*, 2013, pp. 145–159. [Online]. Available: http://dx.doi.org/10.1109/SP.2013.20

BIBLIOGRAPHY

[33] P. Bagade, A. Banerjee, J. Milazzo, and S. K. S. Gupta, "Protect your BSN: No handshakes, just namaste!" in *IEEE International Conference on Body Sensor Networks (BSN)*, 2013, pp. 1–6.

[34] S. Heath, *Embedded Systems Design*, 1st ed. Butterworth-Heinemann, 1997.

[35] K. Sohraby, D. Minoli, and T. Znati, *Wireless Sensor Networks: Technology, Protocols, and Applications*. Wiley, 2007. [Online]. Available: http://www.wiley.com/WileyCDA/WileyTitle/productCd-0471743003.html

[36] V. Shnayder, B. Chen, K. Lorincz, T. R. F. Fulford Jones, and M. Welsh, "Sensor networks for medical care," in *Proc. 3rd International Conference on Embedded Networked Sensor Systems (SenSys '05)*, 2005, p. 314. [Online]. Available: http://doi.acm.org/10.1145/1098918.1098979

[37] United States Statutes at Large, "Federal Food, Drug, and Cosmetic Act (FD&C Act), Section 201 (21 U.S.C. 321)," http://www.fda.gov/RegulatoryInformation/Legislation/FederalFoodDrugandCosmeticActFDCAct/FDCActChaptersIandIIShortTitleandDefinitions/ucm086297.htm.

[38] (2011, Nov.) U.S Food and Drug Administration, Office of International Programs (OIP). [Online]. Available: http://www.fda.gov/AboutFDA/CentersOffices/OfficeofGlobalRegulatoryOperationsandPolicy/OfficeofInternationalPrograms/ucm236581.htm.

[39] Federal Communications Commission, "Report and Order (FCC No 00-211), Paragraph 24," http://transition.fcc.gov/Bureaus/Engineering_Technology/Orders/2000/fcc00211.pdf.

[40] M. Pajic, Z. Jiang, I. Lee, O. Sokolsky, and R. Mangharam, "From verification to implementation: A model translation tool and a pacemaker case study," in *Proc. IEEE 18th Real Time and Embedded Technology and Applications Symposium (RTAS '12)*, 2012, pp. 173–184. [Online]. Available: http://dx.doi.org/10.1109/RTAS.2012.25

[41] R. K. Shepard and K. A. Ellenbogen, "Leads and longevity: How long will your pacemaker last?" *Europace*, vol. 11, no. 2, pp. 142–143, 2009.

[42] Code of Federal Regulations, "Title 47 Part 95 Subpart I C.F.R 47, 95 Subpart I, Federal Communications Commission - Medical Device Radiocommunication Service (MedRadio)," http://transition.fcc.gov/Bureaus/Engineering_Technology/Orders/2000/fcc00211.pdf.

[43] B. Latré, B. Braem, I. Moerman, C. Blondia, and P. Demeester, "A survey on wireless body area networks," *Wireless Networks*, vol. 17, no. 1, pp. 1–18, Jan. 2011. [Online]. Available: http://dx.doi.org/10.1007/s11276-010-0252-4

[44] D. Foo Kune, K. K. Venkatasubramanian, E. Vasserman, I. Lee, and Y. Kim, "Toward a safe integrated clinical environment: A communication security per-

spective," in *Proc. 2012 ACM workshop on Medical Communication Systems*, 2012, pp. 7–12.

[45] M. Clarke, D. Bogia, K. Hassing, L. Steubesand, T. Chan, and D. Ayyagari, "Developing a standard for personal health devices based on 11073," in *Proc. 29th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBS 2007)*, 2007, pp. 6174–6176.

[46] (2011, Nov.) Health level seven international. http://www.hl7.org/.

[47] (2014, Mar.) Integrating the healthcare enterprise. http://www.ihe.net/.

[48] ASTM F-29.21, "Medical devices and medical systems — essential safety requirements for equipment comprising the patient-centric integrated clinical environment (ICE)," 2009.

[49] T. Denning, A. Borning, B. Friedman, B. T. Gill, T. Kohno, and W. H. Maisel, "Patients, pacemakers, and implantable defibrillators: Human values and security for wireless implantable medical devices," in *Proc. SIGCHI Conference on Human Factors in Computing Systems (CHI '10)*, 2010, pp. 917–926. [Online]. Available: http://doi.acm.org/10.1145/1753326.1753462

[50] P. Kumar and H.-J. Lee, "Security issues in healthcare applications using wireless medical sensor networks: A survey," *Sensors*, vol. 12, no. 1, pp. 55–91, 2011. [Online]. Available: http://www.mdpi.com/1424-8220/12/1/55

[51] S. Avancha, A. Baxi, and D. Kotz, "Privacy in mobile technology for personal healthcare," *ACM Comput. Surv.*, vol. 45, no. 1, pp. 3:1–3:54, Dec. 2012.

[52] K. Ellis and N. Serinken, "Characteristics of radio transmitter fingerprints," *Radio Science*, vol. 36, no. 4, pp. 585–597, 2001.

[53] J. Hall, M. Barbeau, and E. Kranakis, "Detecting rogue devices in bluetooth networks using radio frequency fingerprinting," in *IASTED International Conference on Communications and Computer Networks*, 2006.

[54] K. B. Rasmussen and S. Čapkun, "Implications of radio fingerprinting on the security of sensor networks," in *Proc. 3rd International Conference on Security and Privacy in Communications Networks and the Workshops (SecureComm 2007)*, 2007, pp. 331–340.

[55] M. Rostami, A. Juels, and F. Koushanfar, "Heart-to-heart (H2H): Authentication for implanted medical devices," in *Proc. 20th ACM Conference on Computer and Communications Security (CCS 2013)*, Nov. 2013. [Online]. Available: http://www.aceslab.org/sites/default/files/H2H.pdf

[56] N. Henry Jr., N. Paul, and N. McFarlane, "Using bowel sounds to create a forensically-aware insulin pump system," in *Proc. 4th USENIX Workshop on Health Information Technology (HealthTech)*, 2013.

[57] C. Hu, X. Cheng, F. Zhang, D. Wu, X. Liao, and D. Chen, "OPFKA: Se-

cure and efficient ordered-physiological-feature-based key agreement for wireless body area networks," *Proc. 32nd IEEE International Conference on Computer Communications (INFOCOM 2013)*, 2013.

[58] L. Shi, J. Yuan, S. Yu, and M. Li, "ASK-BAN: Authenticated secret key extraction utilizing channel characteristics for body area networks," in *Proc. 6th ACM conference on Security and privacy in wireless and mobile networks (WiSec '13)*, 2013, pp. 155–166. [Online]. Available: http://doi.acm.org/10.1145/2462096.2462123

[59] H. Alemzadeh, R. Iyer, Z. Kalbarczyk, and J. Raman, "Analysis of safety-critical computer failures in medical devices," vol. 11, no. 4, pp. 14–26, 2013.

[60] N. O. Tippenhauer, L. Malisa, A. Ranganathan, and S. Čapkun, "On limitations of friendly jamming for confidentiality," in *Proc. 34th Annual IEEE Symposium on Security and Privacy (SP 2013)*, 2013, pp. 160–173. [Online]. Available: http://dx.doi.org/10.1109/SP.2013.21

[61] M. Zhang, A. Raghunathan, and N. Jha, "MedMon: Securing medical devices through wireless monitoring and anomaly detection," vol. 7, no. 6, pp. 871–881, Dec. 2013.

[62] S. Chang, Y. Hu, H. Anderson, T. Fu, and E. Y. L. Huang, "Body area network security: Robust key establishment using human body channel," in *Proc. 3rd USENIX Workshop on Health Security and Privacy (HealthSec)*,

vol. 37, no. 6, Aug. 2012. [Online]. Available: https://www.usenix.org/conference/healthsec12/workshop-program/presentation/Chang

[63] D. B. Kramer, M. Baker, B. Ransford, A. Molina-Markham, Q. Stewart, K. Fu, and M. R. Reynolds, "Security and privacy qualities of medical devices: An analysis of FDA postmarket surveillance," *PLoS ONE*, vol. 7, p. e40200, Jul. 2012. [Online]. Available: http://dx.doi.org/10.1371%2Fjournal.pone.0040200

[64] L. Shi, M. Li, S. Yu, and J. Yuan, "BANA: Body area network authentication exploiting channel characteristics," in *Proc. 5th ACM conference on Security and Privacy in Wireless and Mobile Networks (WiSec '12)*, 2012, pp. 27–38. [Online]. Available: http://doi.acm.org/10.1145/2185448.2185454

[65] I. Martinovic, D. Davies, M. Frank, D. Perito, T. Ros, and D. Song, "On the feasibility of side-channel attacks with brain-computer interfaces," in *Proc. 21st USENIX Security Symposium (USENIX Security '12)*, 2012, p. 34. [Online]. Available: http://dl.acm.org/citation.cfm?id=2362793.2362827

[66] K. K. Venkatasubramanian, A. Banerjee, and S. K. S. Gupta, "PSKA: Usable and secure key agreement scheme for body area networks," vol. 14, no. 1, pp. 60–68, 2010.

[67] A. D. Jurik and A. C. Weaver, "Securing mobile devices with biotelemetry," in *Proc. 20th International Conference on Computer Communications and Networks (ICCCN 2011)*, 2011, pp. 1–6.

[68] Z. Jiang, M. Pajic, A. Connolly, S. Dixit, and R. Mangharam, "Real-time heart model for implantable cardiac device validation and verification," in *Proc. 2010 22nd Euromicro Conference on Real-Time Systems (ECRTS '10)*, 2010, pp. 239–248. [Online]. Available: http://dx.doi.org/10.1109/ECRTS.2010.36

[69] S. Gollakota, H. Hassanieh, B. Ransford, D. Katabi, and K. Fu, "They can hear your heartbeats: Non-invasive security for implantable medical devices," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 2–13, Aug. 2011. [Online]. Available: http://doi.acm.org/10.1145/2043164.2018438

[70] K. K. Venkatasubramanian and S. K. S. Gupta, "Physiological value-based efficient usable security solutions for body sensor networks," *ACM Trans. Sen. Netw. (TOSN)*, vol. 6, no. 4, pp. 31:1–31:36, Jul. 2010. [Online]. Available: http://doi.acm.org/10.1145/1777406.1777410

[71] T. Halevi and N. Saxena, "On pairing constrained wireless devices based on secrecy of auxiliary channels: The case of acoustic eavesdropping," in *Proc. 17th ACM conference on Computer and Communications Security (CCS 2010)*, 2010, pp. 97–108. [Online]. Available: http://doi.acm.org/10.1145/1866307.1866319

[72] F. Xu, Z. Qin, C. Tan, B. Wang, and Q. Li, "IMDGuard: Securing implantable medical devices with the external wearable guardian," in *Proc. 30th IEEE International Conference on Computer Communications (INFOCOM 2011)*, 2011, pp. 1862–1870.

[73] X. Hei, X. Du, J. Wu, and F. Hu, "Defending resource depletion attacks on implantable medical devices," in *Proc. IEEE Global Telecommunications Conference (GLOBECOM 2010)*, 2010, pp. 1–5.

[74] K. K. Venkatasubramanian, A. Banerjee, and S. K. S. Gupta, "Plethysmogram-based secure inter-sensor communication in body area networks," in *Proc. Military Communications Conference (MILCOM 2008)*, Nov. 2008, pp. 1–7.

[75] K. B. Rasmussen, C. Castelluccia, T. S. Heydt-Benjamin, and S. Čapkun, "Proximity-based access control for implantable medical devices," in *Proc. 16th ACM conference on Computer and Communications Security (CCS 2009)*, 2009, pp. 410–419. [Online]. Available:

[76] T. Denning, Y. Matsuoka, and T. Kohno, "Neurosecurity: Security and privacy for neural devices," *Journal of Neurosurgery: Pediatrics*, vol. 27, no. 1, p. E7, Jul. 2009. [Online]. Available: http://thejns.org/doi/abs/10.3171/2009.4.FOCUS0985?ai=rw&mi=0&af=R

[77] L. Ballard, S. Kamara, and M. K. Reiter, "The practical subtleties of biometric key generation," in *Proc. 17th conference on Security Symposium (SS '08)*, 2008, pp. 61–74. [Online]. Available: http://dl.acm.org/citation.cfm?id=1496711.1496716

[78] K. K. Venkatasubramanian, A. Banerjee, and S. K. S. Gupta, "EKG-based

key agreement in body sensor networks," in *Proc. 2nd Workshop on Mission Critical Networks (INFOCOM Workshops 2008)*, 2008, pp. 1–6.

[79] T. Denning, K. Fu, and T. Kohno, "Absence makes the heart grow fonder: New directions for implantable medical device security," in *Proc. 3rd conference on Hot Topics in Security (HotSec '08)*, 2008, pp. 5:1–5:7. [Online]. Available: http://dl.acm.org/citation.cfm?id=1496671.1496676

[80] K. K. Venkatasubramanian and S. K. S. Gupta, "Security for pervasive health monitoring sensor applications," in *Proc. 4th International Conference on Intelligent Sensing and Information Processing (ICISIP)*, 2006.

[81] S. Cherukuri, K. K. Venkatasubramanian, and S. K. S. Gupta, "BioSec: a biometric based approach for securing communication in wireless networks of biosensors implanted in the human body," in *Proc. International Conference on Parallel Processing Workshops*, 2003, pp. 432–439.

[82] C. C. Y. Poon, Y.-T. Zhang, and S.-D. Bao, "A novel biometrics method to secure wireless body area sensor networks for telemedicine and m-health," vol. 44, no. 4, pp. 73–81, 2006.

[83] M. Li, S. Yu, J. D. Guttman, W. Lou, and K. Ren, "Secure ad hoc trust initialization and key management in wireless body area networks," *ACM Trans. Sen. Netw. (TOSN)*, vol. 9, no. 2, pp. 18:1–18:35, Apr. 2013. [Online]. Available: http://doi.acm.org/http://dx.doi.org/10.1145/2422966.2422975

[84] C. Hu, N. Zhang, H. Li, X. Cheng, and X. Liao, "Body area network security: A fuzzy attribute-based signcryption scheme," vol. 31, no. 9, pp. 37–46, Sep. 2013.

[85] F. Miao, L. Jiang, Y. Li, and Y.-T. Zhang, "Biometrics based novel key distribution solution for body sensor networks," in *Proc. Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC 2009)*, 2009, pp. 2458–2461.

[86] G. Zhang, C. C. Y. Poon, and Y. Zhang, "A fast key generation method based on dynamic biometrics to secure wireless body sensor networks for p-health," in *Proc. Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC 2010)*, 2010, pp. 2034–2036.

[87] S. Hanna, R. Rolles, A. Molina-Markham, P. Poosankam, K. Fu, and D. Song, "Take two software updates and see me in the morning: The case for software security evaluations of medical devices," in *Proc. 2nd USENIX conference on Health Security and Privacy (HealthSec '11)*, 2011, p. 6. [Online]. Available: http://dl.acm.org/citation.cfm?id=2028026.2028032

[88] P. Roberts. (2011, Oct.) Blind attack on wireless insulin pumps could deliver lethal dose. [Online]. Available: http://threatpost.com/blind-attack-wireless-insulin-pumps-could-deliver-lethal-dose.

[89] J. Radcliffe. (2011, Aug.) Hacking medical devices for fun and insulin: Breaking

the human SCADA system. http://media.blackhat.com/bh-us-11/Radcliffe/BH_US_11_Radcliffe_Hacking_Medical_Devices_WP.pdf.

[90] S. Brands and D. Chaum, "Distance-bounding protocols," in *Advances in Cryptology - EUROCRYPT'93*, ser. Lecture Notes in Computer Science, vol. 765, 1994, pp. 344–359. [Online]. Available: http://dx.doi.org/10.1007/3-540-48285-7_30

[91] Z. Kfir and A. Wool, "Picking virtual pockets using relay attacks on contactless smartcard," in *Proc. 1st International Conference on Security and Privacy for Emerging Areas in Communications Networks (SECURECOMM '05)*, 2005, pp. 47–58. [Online]. Available: http://dx.doi.org/10.1109/SECURECOMM.2005.32

[92] A. Czeskis, K. Koscher, J. R. Smith, and T. Kohno, "RFIDs and secret handshakes: Defending against ghost-and-leech attacks and unauthorized reads with context-aware communications," in *Proc. 15th ACM conference on Computer and Communications Security (CCS 2008)*, 2008, pp. 479–490. [Online]. Available: http://doi.acm.org/10.1145/1455770.1455831

[93] C. Cremers, K. B. Rasmussen, B. Schmidt, and S. Čapkun, "Distance hijacking attacks on distance bounding protocols," in *Proc. 33rd Annual IEEE Symposium on Security and Privacy (SP 2012)*, 2012, pp. 113–127. [Online]. Available: http://dx.doi.org/10.1109/SP.2012.17

[94] M. Goodrich, M. Sirivianos, J. Solis, G. Tsudik, and E. Uzun, "Loud and clear: Human-verifiable authentication based on audio," in *Proc. 26th IEEE International Conference on Distributed Computing Systems (ICDCS 2006)*, 2006, p. 10.

[95] A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner, "A survey of mobile malware in the wild," in *Proc. 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM '11)*, 2011, pp. 3–14. [Online]. Available: http://doi.acm.org/10.1145/2046614.2046618

[96] T. H. Faris, *Safe and Sound Software: Creating an Efficient and Effective Quality System for Software Medical Device Organizations*. ASQ Quality Press, 2006.

[97] Z. Jiang, M. Pajic, and R. Mangharam, "Model-based closed-loop testing of implantable pacemakers," in *Proc. IEEE/ACM 2nd International Conference on Cyber-Physical Systems*, 2011, pp. 131–140.

[98] K. Sandler, L. Ohrstrom, L. Moy, and R. McVay, "Killed by code: Software transparency in implantable medical devices," http://www.softwarefreedom.org, Jul. 2010.

[99] K. Fu, "Trustworthy medical device software," in *Health Effectiveness of the FDA 510(k) Clearance Process: Measuring Postmarket Performance and Other Select Topics: Workshop Report*, Jul. 2011.

[100] (2014, Mar.) Peach fuzzer. http://peachfuzzer.com/.

[101] (2014, Mar.) Archimedes: Ann Arbor Center for Medical Device Security. http://secure-medicine.org.

[102] American National Standards Institute/Association for the Advancement of Medical Instrumentation (ANSI/AAMI), "Active implantable medical devices — Electromagnetic compatibility — EMC test protocols for implantable cardiac pacemakers and implantable cardioverter defibrillators," 2007.

[103] A. L. Goldberger, L. A. N. Amaral, L. Glass, J. M. Hausdorff, P. C. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C. Peng, and H. E. Stanley, "PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals," *Circulation*, vol. 101, no. 23, pp. e215–e220, Jun. 2000.

[104] I. Radojičić, D. Mandić, and D. Vulić, "On the presence of deterministic chaos in HRV signals," in *Computers in Cardiology 2001*, 2001, pp. 465–468.

[105] (2010) NIST special publication 800-22rev1a: A statistical test suite for the validation of random number generators and pseudo random number generators for cryptographic applications. http://csrc.nist.gov/groups/ST/toolkit/rng/documents/SP800-22rev1a.pdf. NIST.

[106] V. Fischer, "A closer look at security in random number generators design,"

in *Constructive Side-Channel Analysis and Secure Design*, ser. Lecture Notes in Computer Science, 2012, vol. 7275, pp. 167–182. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-29912-4_13

[107] W. Schindler and W. Killmann, "Evaluation criteria for true (physical) random number generators used in cryptographic applications," in *Cryptographic Hardware and Embedded Systems (CHES 2002)*, ser. Lecture Notes in Computer Science, 2003, vol. 2523, pp. 431–449. [Online]. Available: http://dx.doi.org/10.1007/3-540-36400-5_31

[108] M. Poh, D. J. McDuff, and R. W. Picard, "Advancements in noncontact, multiparameter physiological measurements using a webcam," vol. 58, no. 1, pp. 7–11, Jan. 2011.

[109] S. Kwon, H. Kim, and K. S. Park, "Validation of heart rate extraction using video imaging on a built-in camera system of a smartphone," in *Proc. Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC 2012)*, Aug. 2012, pp. 2174–2177.

[110] H. Wu, M. Rubinstein, E. Shih, J. Guttag, F. Durand, and W. Freeman, "Eulerian video magnification for revealing subtle changes in the world," *ACM Trans. Graph. (SIGGRAPH)*, pp. 65:1–65:8, Jul. 2012.

[111] G. Barbosa, "Apple execs Eddy Cue & Craig Federighi talk Apple Music, App Store & more in new interview," Available at http://9to5mac.com/2016/02/12/

apple-execs-eddy-cue-craig-federighi-talk-apple-music-app-store-more-in-new-interview/, February 2016.

[112] A. Covert, "Apple's iMessage is the DEA's worst nightmare," Available at http://money.cnn.com/2013/04/07/technology/security/imessage-iphone-dea/, April 2013.

[113] M. Apuzzo, D. E. Sanger, and M. S. Schmidt, "Apple and other tech companies tangle with U.S. over data access," Available at http://www.nytimes.com/2015/09/08/us/politics/apple-and-other-tech-companies-tangle-with-us-over-access-to-data.html, September 2015.

[114] D. Paletta, "FBI Chief Punches Back on Encryption," *Wall Street Journal*, July 2015. [Online]. Available: http://www.wsj.com/articles/fbi-chief-punches-back-on-encryption-1436217665

[115] A. Griffin, "WhatsApp and iMessage could be banned under new surveillance plans," *The Independent*, January 2015. [Online]. Available: http://www.independent.co.uk/life-style/gadgets-and-tech/news/whatsapp-and-snapchat-could-be-banned-under-new-surveillance-plans-9973035.html

[116] H. Abelson, R. Anderson, S. M. Bellovin, J. Benaloh, M. Blaze, W. W. Diffie, J. Gilmore, M. Green, S. Landau, P. G. Neumann, R. L. Rivest, J. I. Schiller,

B. Schneier, M. A. Specter, and D. J. Weitzner, "Keys under doormats," *Commun. ACM*, vol. 58, no. 10, pp. 24–26, Sep. 2015. [Online]. Available: http://doi.acm.org/10.1145/2814825

[117] N. Messieh, "Apple's iMessage and Facetime blocked in the UAE," *TheNextWeb*, November 2011. [Online]. Available: http://thenextweb.com/me/2011/11/13/apples-imessage-and-facetime-blocked-in-the-uae/

[118] Apple Inc., "Privacy," Available at http://www.apple.com/privacy/approach-to-privacy/, 2015.

[119] Apple Computer, "iOS Security: iOS 9.0 or later," Available at https://www.apple.com/business/docs/iOS_Security_Guide.pdf, September 2015.

[120] F. Raynal, "iMessage privacy," Available at http://blog.quarkslab.com/imessage-privacy.html, October 2013.

[121] "iMessage," In OpenIM Wiki, Available at https://imfreedom.org/wiki/IMessage.

[122] G. Shih and P. Carsten, "Apple begins storing users' personal data on servers in China," *Reuters*, August 2014.

[123] N. Unger, S. Dechand, J. Bonneau, S. Fahl, H. Perl, I. Goldberg, and M. Smith, "SoK: Secure messaging," in *IEEE S&P (Oakland) '15*, 2015.

[124] D. Chiba, T. Matsuda, J. C. N. Schuldt, and K. Matsuura, "Efficient generic constructions of signcryption with insider security in the multi-user setting," in *ACNS '11*, 2011, pp. 220–237.

[125] M. Bellare and C. Namprempre, "Authenticated encryption; relations among notions and analysis of the generic composition paradigm," *J. Cryptol.*, vol. 21, no. 4, pp. 469–491, Sep. 2008.

[126] S. Vaudenay, "Security flaws induced by CBC padding - applications to SSL, IPSEC, WTLS," in *EUROCRYPT '02*, vol. 2332 of LNCS. London, UK: Springer-Verlag, 2002, pp. 534–546.

[127] P. Deutsch, "RFC 1952: GZIP file format specification version 4.3," May 1996.

[128] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Transactions on Information Theory*, vol. 23, no. 3, pp. 337–343, 1977.

[129] "Pushproxy: A man-in-the-middle proxy for ios and os x device push connections," Available at https://github.com/meeee/pushproxy.

[130] M. Bellare and P. Rogaway, "Optimal asymmetric encryption: How to encrypt with RSA," in *EUROCRYPT '94*, A. D. Santis, Ed., vol. 950 of LNCS. Springer, 1994, pp. 92–111.

[131] N. Borisov, I. Goldberg, and E. Brewer, "Off-the-record communication, or, why not to use PGP," ser. WPES '04. ACM Press, 2004, pp. 77–84.

[132] "Textsecure," https://github.com/WhisperSystems/TextSecure/wiki/ProtocolV2, accessed: 2014-11-13.

[133] M. S. Melara, A. Blankstein, J. Bonneau, E. W. Felten, and M. J. Freedman, "CONIKS: Bringing key transparency to end users," in *USENIX '15*. Washington, D.C.: USENIX Association, Aug. 2015, pp. 383–398.

[134] T. Frosch, C. Mainka, C. Bader, F. Bergsma, J. Schwenk, and T. Holz, "How secure is TextSecure?" Cryptography ePrint Archive, October 2014. [Online]. Available: https://eprint.iacr.org/2014/904

[135] I. Goldberg, B. Ustaoğlu, M. D. Van Gundy, and H. Chen, "Multi-party off-the-record messaging," in *CCS '09*, ser. CCS '09. New York, NY, USA: ACM, 2009, pp. 358–368.

[136] N. Unger and I. Goldberg, "Deniable key exchanges for secure messaging," in *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '15. New York, NY, USA: ACM, 2015, pp. 1211–1223. [Online]. Available: http://doi.acm.org/10.1145/2810103.2813616

[137] Y. Chen, T. Li, X. Wang, K. Chen, and X. Han, "Perplexed messengers from the cloud: Automated security analysis of push-messaging integrations," in *CCS '15*, ser. CCS '15. New York, NY, USA: ACM, 2015, pp. 1260–1272.

[138] R. Bardou, R. Focardi, Y. Kawamoto, L. Simionato, G. Steel, and J.-K. Tsay,

"Efficient padding oracle attacks on cryptographic hardware," in *CRYPTO '12*. Springer, 2012, vol. 7417 of LNCS, pp. 608–625.

[139] B. Canvel, A. Hiltgen, S. Vaudenay, and M. Vuagnoux, "Password interception in a SSL/TLS channel," in *CRYPTO '03*. Springer Berlin Heidelberg, 2003, vol. 2729 of LNCS, pp. 583–599.

[140] N. J. AlFardan and K. G. Paterson, "Lucky thirteen: Breaking the TLS and DTLS record protocols," in *IEEE S&P (Oakland) '13*, 2013, pp. 526–540.

[141] T. Jager and J. Somorovsky, "How to break XML encryption," in *ACM CCS '2011*. ACM Press, October 2011.

[142] D. Kupser, C. Mainka, J. Schwenk, and J. Somorovsky, "How to break XML encryption – automatically," in *Proceedings of the 9th USENIX Conference on Offensive Technologies*, ser. WOOT'15. Berkeley, CA, USA: USENIX Association, 2015.

[143] M. Stay, "ZIP attacks with reduced known plaintext," in *Fast Software Encryption, 8th International Workshop, FSE 2001 Yokohama, Japan, April 2-4, 2001, Revised Papers*, 2001, pp. 125–134.

[144] E. Biham and P. C. Kocher, "A known plaintext attack on the PKZIP stream cipher," in *Fast Software Encryption: Second International Workshop. Leuven, Belgium, 14-16 December 1994, Proceedings*, 1994, pp. 144–153.

[145] J. Kelsey, "Compression and information leakage of plaintext," in *FSE '02*, vol. 2365 of LNCS. Springer, 2002, pp. 263–276.

[146] T. Kohno, "Attacking and repairing the winZip encryption scheme," in *ACM CCS '2004*. ACM Press, 2004, pp. 72–81.

[147] J. Rizzo and T. Duong, "The CRIME Attack," Available at https://docs.google.com/presentation/d/11eBmGiHbYcHR9gL5nDyZChu_-lCa2GizeuOfaLU2HOU/edit#slide=id.g1d134dff_1_222, September 2012.

[148] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, and R. Thayer, "RFC 4880: OpenPGP Message Format," Available at https://tools.ietf.org/html/rfc4880, November 2007.

[149] "MCABBER," Available at https://mcabber.com/.

[150] "Trustwave to escape 'death penalty' for SSL skeleton key," 2012. [Online]. Available: http://www.theregister.co.uk/2012/02/14/trustwave_analysis/

[151] "Apple pulls ad-blocking apps that can 'compromise' security," *Engadget*, October 2015. [Online]. Available: http://www.engadget.com/2015/10/09/apple-pulls-ad-blocking-ads-that-can-compromise-security/

[152] R. Shapiro, S. Bratus, and S. W. Smith, ""Weird machines" in ELF: A spotlight on the underappreciated metadata," in *Proceedings of WOOT 2013*.

USENIX, Aug. 2013. [Online]. Available: https://www.usenix.org/conference/woot13/workshop-program/presentation/Shapiro

[153] H. Shacham, "The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86)," in *Proceedings of CCS 2007*, S. De Capitani di Vimercati and P. Syverson, Eds. ACM Press, Oct. 2007, pp. 552–61.

[154] E. Buchanan, R. Roemer, H. Shacham, and S. Savage, "When good instructions go bad: Generalizing return-oriented programming to RISC," in *Proceedings of CCS 2008*, P. Syverson and S. Jha, Eds. ACM Press, Oct. 2008, pp. 27–38.

[155] A. Francillon and C. Castelluccia, "Code injection attacks on Harvard-architecture devices," in *Proceedings of CCS 2008*, P. Syverson and S. Jha, Eds. ACM Press, Oct. 2008, pp. 15–26.

[156] F. Lidner, "Developments in Cisco IOS forensics. CONFidence 2.0," http://www.recurity-labs.com/content/pub/FX_Router_Exploitation.pdf, Nov. 2009.

[157] S. Checkoway, A. J. Feldman, B. Kantor, J. A. Halderman, E. W. Felten, and H. Shacham, "Can DREs provide long-lasting security? The case of return-oriented programming and the AVC Advantage," in *Proceedings of EVT/-WOTE 2009*, D. Jefferson, J. L. Hall, and T. Moran, Eds. USENIX/ACCURATE/IAVoSS, Aug. 2009.

[158] S. Checkoway, L. Davi, A. Dmitrienko, A.-R. Sadeghi, H. Shacham, and

M. Winandy, "Return-oriented programming without returns," in *Proceedings of CCS 2010*, A. Keromytis and V. Shmatikov, Eds. ACM Press, Oct. 2010, pp. 559–72. [Online]. Available: https://www.cs.jhu.edu/~s/papers/noret_ccs2010.html

[159] T. Kornau, "Return oriented programming for the ARM architecture," http://zynamics.com/downloads/kornau-tim--diplomarbeit--rop.pdf, 2009, master thesis, Ruhr-University Bochum, Germany.

[160] R. Roemer, "Finding the bad in good code: Automated return-oriented programming exploit discovery," Master's thesis, UC San Diego, Mar. 2009, online: https://cseweb.ucsd.edu/~rroemer/doc/thesis.pdf.

[161] R. Hund, T. Holz, and F. Freiling, "Return-oriented rootkits: Bypassing kernel code integrity protection mechanisms," in *Proceedings of USENIX Security 2009*, F. Monrose, Ed. USENIX, Aug. 2009, pp. 383–98.

[162] A. Bittau, A. Belay, A. Mashtizadeh, D. Mazieres, and D. Boneh, "Hacking blind," in *Proceedings of IEEE Security and Privacy ("Oakland") 2014*, May 2014, pp. 227–42.

[163] J. Bangert, S. Bratus, R. Shapiro, and S. W. Smith, "Page-fault weird machine: Lessons in instruction-less computation," in *Proceedings of WOOT 2013*. USENIX, Aug. 2013. [Online]. Available: https://www.usenix.org/conference/woot13/workshop-program/presentation/Bangert

[164] J. Vanegue, "The weird machines in proof-carrying code," in *Proceedings of SPW 2014*. IEEE Computer Society, May 2014, pp. 209–13. [Online]. Available: http://www.ieee-security.org/TC/SPW2014/papers/5103a209.PDF

[165] S. Bratus, T. Darley, M. Locasto, M. L. Patterson, R. Shapiro, and A. Shubina, "Beyond planted bugs in "trusting trust": The input-processing frontier," *Security Privacy, IEEE*, vol. 12, no. 1, pp. 83–87, Jan. 2014.

[166] S. Dolan, "`mov` is Turing-complete," Jul. 2013. [Online]. Available: http://www.cl.cam.ac.uk/~sd601/papers/mov.pdf

[167] D. Q. Goldin, S. A. Smolka, P. C. Attie, and E. L. Sonderegger, "Turing machines, transition systems, and interaction," *Information and Computation*, vol. 194, no. 2, pp. 101–28, Nov. 2004. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0890540104001257

[168] M. Atkinson, F. ois Bancilhon, D. DeWitt, K. Dittrich, D. Maier, and S. Zdonik, "The object-oriented database system manifesto," in *Proceedings of DOOD 1989*, W. Kim, J.-M. Nicolas, and S. Nishio, Eds. Elsevier, Dec. 1989, pp. 223–40. [Online]. Available: https://www.cs.cmu.edu/~clamen/OODBMS/Manifesto/Manifesto.PS.gz

[169] *Intel Platform Controller Hub EG20T: Datasheet*, Intel, Jul. 2012. [Online]. Available: http://www.intel.com/content/www/us/en/intelligent-systems/queens-bay/platform-controller-hub-eg20t-datasheet.html

[170] *CoreLinkTM DMA Controller DMA-330*, ARM, Jul. 2010. [Online]. Available: http://infocenter.arm.com/help/topic/com.arm.doc.ddi0424c/DDI0424C_dma330_r1p1_trm.pdf

[171] V. Srinivasan, A. K. Santhanam, and M. Srinivasan. (2005, Dec.) Cell broadband engine processor dma engines, part 1: The little engines that move data. [Online]. Available: http://www.ibm.com/developerworks/library/pa-celldmas/

[172] Intel Server Platform Group, "Intel QuickData technology software guide for Linux," May 2008. [Online]. Available: http://www.intel.com/content/dam/doc/white-paper/quickdata-technology-software-guide-for-linux-paper.pdf

[173] *BCM2835 ARM Peripherals*, Broadcom Corporation, Feb. 2012. [Online]. Available: https://www.raspberrypi.org/wp-content/uploads/2012/02/BCM2835-ARM-Peripherals.pdf

[174] C. Böhm, "On a family of Turing machines and the related programming language," *International Computation Centre Bulletin*, vol. 3, pp. 187–94, Jul. 1964.

[175] C. Böhm and G. Jacopini, "Flow diagrams, Turing machines and languages with only two formation rules," *Communications of the ACM*, vol. 9, no. 5, pp. 366–71, May 1966.

[176] G. Vasiliadis, M. Polychronakis, and S. Ioannidis, "GPU-assisted malware," in *Proceedings of MALWARE 2010*, J.-Y. Marion, N. Rathaus, and C. Zhou, Eds. IEEE Computer Society, Oct. 2010, pp. 1–6. [Online]. Available: http://dcs.ics.forth.gr/Activities/papers/gpumalware.malware10.pdf

[177] E. Ladakis, L. Koromilas, G. Vasiliadis, M. Polychronakis, and S. Ioannidis, "You can type, but you can't hide: A stealthy GPU-based keylogger," in *Proceedings of EuroSec 2013*, T. Holz and S. Ioannidis, Eds. ACM, Apr. 2013. [Online]. Available: http://www.cs.columbia.edu/~mikepo/papers/gpukeylogger.eurosec13.pdf

[178] L. Duflot and Y.-A. Perez, "Can you still trust your network card?" Presented at CanSecWest 2010, Mar. 2010. [Online]. Available: http://www.ssi.gouv.fr/IMG/pdf/csw-trustnetworkcard.pdf

[179] A. Triulzi, "Project Maux Mk.II," Nov. 2008. [Online]. Available: http://www.alchemistowl.org/arrigo/Papers/Arrigo-Triulzi-PACSEC08-Project-Maux-II.pdf

[180] ——, "The Jedi packet trick takes over the Deathstar," 2010. [Online]. Available: http://www.alchemistowl.org/arrigo/Papers/Arrigo-Triulzi-CANSEC10-Project-Maux-III.pdf

[181] L. Duflot, Y.-A. Perez, and B. Morin, "What if you can't trust your network card?" in *Proceedings of RAID 2011*, R. Sommer, D. Balzarotti,

and G. Maier, Eds. Springer, Sep. 2011, pp. 378–397. [Online]. Available: http://www.ssi.gouv.fr/IMG/pdf/paper.pdf

[182] M. Becher, M. Dornseif, and C. N. Klein, "FireWire: all your memory are belong to us," Presented at CanSecWest 2005, May 2005. [Online]. Available: https://cansecwest.com/core05/2005-firewire-cansecwest.pdf

[183] T. Garrison. (2011, Sep.) Firewire attacks against Mac OS Lion FileVault 2 encryption. [Online]. Available: http://www.frameloss.org/2011/09/18/firewire-attacks-against-mac-os-lion-filevault-2-encryption/

[184] A. Boileau, "Hit by a bus: Physical access attacks with firewire," Sep. 2006. [Online]. Available: http://www.security-assessment.com/files/presentations/ab_firewire_rux2k6-final.pdf

[185] P. Panholzer, "Physical security attacks on Windows Vista," Mar. 2008. [Online]. Available: https://www.sec-consult.com/fxdata/seccons/prod/downloads/vista_physical_attacks.pdf

[186] L. Kalenderidis and S. Collinson, "Thunderbolts and lightning, very very frightening," Presented at SyScan 2014, May 2014. [Online]. Available: https://www.youtube.com/watch?v=0FoVmBOdbhg

[187] P. Stewin and I. Bystrov, "Understanding dma malware," in *Proceedings*

of DIMVA 2012. Springer-Verlag, Jul. 2012, pp. 21–41. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-37300-8_2

[188] A. Tereshkin and R. Wojtczuk, "Introducing ring −3 rootkits," Presented at Black Hat Briefings, Jul. 2009. [Online]. Available: http://www.blackhat.com/presentations/bh-usa-09/TERESHKIN/BHUSA09-Tereshkin-Ring3Rootkit-SLIDES.pdf

[189] D. Farmer, "IPMI: Freight train to hell," Jan. 2013. [Online]. Available: http://fish2.com/ipmi/itrain.pdf

[190] H. Moore. (2013, Jul.) A penetration tester's guide to IPMI and BMCs. [Online]. Available: https://community.rapid7.com/community/metasploit/blog/2013/07/02/a-penetration-testers-guide-to-ipmi

[191] C. Miller, "Battery firmware hacking: Inside the innards of a smart battery," Presented at Black Hat Briefings, Aug. 2011. [Online]. Available: http://media.blackhat.com/bh-us-11/Miller/BH_US_11_Miller_Battery_Firmware_Public_WP.pdf

[192] M. Brocker and S. Checkoway, "iSeeYou: Disabling the MacBook webcam indicator LED," in *Proceedings of USENIX Security 2014)*. USENIX Association, Aug. 2014, pp. 337–52. [Online]. Available: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/brocker

[193] Aleph One, "Smashing the stack for fun and profit," *Phrack Magazine*, vol. 49, no. 14, Aug. 1996. [Online]. Available: http://www.phrack.org/issues.html?issue=49&id=14

[194] Solar Designer, "Getting around non-executable stack (and fix)," Bugtraq, Aug. 1997. [Online]. Available: http://seclists.org/bugtraq/1997/Aug/0063.html

[195] S. Krahmer, "x86-64 buffer overflow exploits and the borrowed code chunks exploitation technique," Sep. 2005, http://www.suse.de/~krahmer/no-nx.pdf.

[196] E. J. Schwartz, T. Avgerinos, and D. Brumley, "Q: Exploit hardening made easy," in *Proceedings of USENIX Security 2011*, D. Wagner, Ed., Aug. 2011. [Online]. Available: http://users.ece.cmu.edu/~ejschwar/papers/usenix11.pdf

[197] J. Oakley and S. Bratus, "Exploiting the hard-working DWARF: Trojan and exploit techniques with no native executable code," in *Proceedings WOOT 2011*. USENIX Association, Aug. 2011. [Online]. Available: http://dl.acm.org/citation.cfm?id=2028052.2028063

[198] P. Stewin, "A primitive for revealing stealthy peripheral-based attacks on the computing platform's main memory," in *Proceedings of RAID 2013*, Oct. 2013, pp. 1–20. [Online]. Available: http://link.springer.com/chapter/10.1007%2F978-3-642-41284-4_1

[199] Trusted Computing Group, "TCG PC client specific implemen-

tation specification for conventional BIOS," Jul. 2005. [Online]. Available: http://www.trustedcomputinggroup.org/files/temp/64505409-1D09-3519-AD5C611FAD3F799B/PCClientImplementationforBIOS.pdf

[200] I. Hickson, "Web workers editor's draft 19 may 2014," http://www.w3.org/TR/workers/, May 2014.

[201] W. H. A. T. W. Group, "Web workers," http://www.whatwg.org/specs/web-apps/current-work/multipage/workers.html, Jul. 2014.

[202] D. Akhawe, A. Barth, P. E. Lam, J. Mitchell, and D. Song, "Towards a formal foundation of web security," in *Proceedings of the 2010 23rd IEEE Computer Security Foundations Symposium*. IEEE Computer Society, 2010, pp. 290–304. [Online]. Available: http://dx.doi.org/10.1109/CSF.2010.27

[203] D. Glasser, "An interesting kind of javascript memory leak," http://info.meteor.com/blog/an-interesting-kind-of-javascript-memory-leak, 2014.

[204] J. Biniok, "Hash me if you can - a bitcoin miner that supports pure javscript, webworker and webgl mining." https://github.com/derjanb/hamiyoca, 2015.

[205] Y. Tian, Y.-C. Liu, A. Bhosale, L.-S. Huang, P. Tague, and C. Jackson, "All your screens are belong to us: Attacks exploiting the HTML5 screen shar-

ing api," in *Proc. 35th Annual IEEE Symposium on Security and Privacy (SP 2014)*, May 2014.

[206] F. Aboukhadijeh, "Using the HTML5 fullscreen api for phishing attacks," http://feross.org/html5-fullscreen-api-attack/, Oct. 2012, accessed May 27, 2014.

[207] L. Kuppan, "Attacking with HTML5," in *Black Hat Abu Dhabi*, Oct. 2010. [Online]. Available: https://www.usenix.org/conference/healthsec12/workshop-program/presentation/Chang

[208] A. Sacco and F. Muttis, "Html5 heap sprays, pwn all the things," 2012, eUSecWest. [Online]. Available: https://eusecwest.com/speakers.html

[209] "Html5 security cheat sheet," https://www.owasp.org/index.php/HTML5_Security_Cheat_Sheet#Web_Workers/, apr 2014.

[210] S. Son and V. Shmatikov, "The postman always rings twice: Attacking and defending postmessage in html5 websites," in *Proc. 20th Annual Network and Distributed System Security Symposium (NDSS)*. The Internet Society, 2013. [Online]. Available: http://dblp.uni-trier.de/db/conf/ndss/ndss2013.html#SonS13

[211] D. Akhawe, P. Saxena, and D. Song, "Privilege separation in html5 applications," in *Proc. 21st USENIX Conference on Security Symposium*,

Aug. 2012, pp. 23–23. [Online]. Available: http://dl.acm.org/citation.cfm?id=2362793.2362816

[212] "Networked medical devices to exceed 14 million unit sales in 2018," https://www.parksassociates.com/blog/article/dec2013-medical-devices, Dec. 2013.

[213] S. S. Clark, B. Ransford, A. Rahmati, S. Guineau, J. Sorber, W. Xu, and K. Fu, "Wattsupdoc: Power side channels to nonintrusively discover untargeted malware on embedded medical devices," in *Presented as part of the 2013 USENIX Workshop on Health Information Technologies*. USENIX, 2013.

[214] J. Yan, A. Blackwell, R. Anderson, and A. Grant, "Password memorability and security: Empirical results," *IEEE Security and Privacy*, Sep. 2004. [Online]. Available: http://dx.doi.org/10.1109/MSP.2004.81

[215] A. T. Barth, M. A. Hanson, H. C. Powell, D. Unluer, S. G. Wilson, and J. Lach, "Body-coupled communication for body sensor networks," in *Proceedings of the ICST 3rd International Conference on Body Area Networks*, 2008. [Online]. Available: http://dl.acm.org/citation.cfm?id=1460257.1460273

[216] S. Mare, A. Markham, C. Cornelius, R. Peterson, and D. Kotz, "Zebra: Zero-effort bilateral recurring authentication," in *Security and Privacy (SP), 2014 IEEE Symposium on*, May 2014.

[217] M. Ryan, "Bluetooth: With low energy comes low security," in *Proceedings of*

the 7th USENIX Conference on Offensive Technologies. USENIX Association, 2013. [Online]. Available: http://dl.acm.org/citation.cfm?id=2534748.2534754

[218] A. Goode, "Bring your own finger–how mobile is bringing biometrics to consumers," *Biometric Technology Today*, vol. 2014, no. 5, pp. 5–9, 2014.

[219] B. Krzanich, "Intel developer forum san francisco opening keynote," Intel Corporation, Tech. Rep., 2015.

[220] C. Gomez, J. Oller, and J. Paradells, "Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology," *Sensors*, vol. 12, no. 9, p. 11734, 2012. [Online]. Available: http://www.mdpi.com/1424-8220/12/9/11734

[221] A. Czeskis, K. Koscher, J. R. Smith, and T. Kohno, "Rfids and secret handshakes: defending against ghost-and-leech attacks and unauthorized reads with context-aware communications," in *Proceedings of the 15th ACM conference on Computer and communications security*. ACM, 2008, pp. 479–490.

[222] Ekahu, "Asset tracking & management," 2015. [Online]. Available: http://www.ekahau.com/real-time-location-system/solutions/healthcare/asset-tracking-management

[223] "iBeacon for Developers," https://developer.apple.com/ibeacon/, accessed: 2015-08-17.

[224] K. R, W. T, T. JL, and K. BT, "Workarounds to barcode medication administration systems: their occurrences, causes, and threats to patient safety," *J Am Med Inform Assoc*, vol. 15, pp. 408–423, 2008.

[225] T. Labs, "Trusted beacon reference," Apr. 2015. [Online]. Available: http://docs.twocanoes.com/trusted_beacon/index.html

[226] S. Contini, "The factorization of rsa-140," *RSA Laboratories' Bulletin*, vol. 10, pp. 1–2, 1999.

[227] T. Kleinjung, K. Aoki, J. Franke, A. K. Lenstra, E. Thomé, J. W. Bos, P. Gaudry, A. Kruppa, P. L. Montgomery, D. A. Osvik *et al.*, "Factorization of a 768-bit rsa modulus," in *Advances in Cryptology–CRYPTO 2010*. Springer, 2010, pp. 333–350.

[228] N. Sastry, U. Shankar, and D. Wagner, "Secure verification of location claims," in *Proceedings of the 2Nd ACM Workshop on Wireless Security*, ser. WiSe '03. ACM, 2003, pp. 1–10. [Online]. Available: http://doi.acm.org/10.1145/941311.941313

[229] E. Bertino and M. S. Kirkpatrick, "Location-based access control systems for mobile users: concepts and research directions." in *SPRINGL*. ACM, 2011, pp. 49–52. [Online]. Available: http://dblp.uni-trier.de/db/conf/gis/springl2011.html#BertinoK11

[230] M. Portnoi and C. Shen, "Location-aware sign-on and key exchange using attribute-based encryption and bluetooth beacons," in *IEEE Conference on Communications and Network Security*, 2013, pp. 405–406. [Online]. Available: http://dx.doi.org/10.1109/CNS.2013.6682750

[231] L. M. Ni, Y. Liu, Y. C. Lau, and A. P. Patil, "Landmarc: indoor location sensing using active rfid," *Wireless networks*, vol. 10, no. 6, pp. 701–710, 2004.

[232] I. LiveViewGPS, "Gps tracking - tracking systems - you can trust," 2015. [Online]. Available: http://www.liveviewgps.com

[233] M. Bhuptani and S. Moradpour, *RFID field guide: deploying radio frequency identification systems*. Prentice Hall PTR, 2005.

[234] K. Finkenzeller, *RFID Handbook: Radio-frequency identification fundamentals and applications*. Wiley, 1999.

[235] P. Misra and P. Enge, *Global Positioning System: Signals, Measurements and Performance Second Edition*. Lincoln, MA: Ganga-Jamuna Press, 2006.

[236] F. Schrooyen, I. Baert, S. Truijen, L. Pieters, T. Denis, K. Williame, and M. Weyn, "Real time location system over wifi in a healthcare environment," *Journal on Information Technology in Healthcare*, vol. 4, no. 6, pp. 401–416, 2006.

[237] R. Want, "Near field communication," *IEEE Pervasive Computing*, no. 3, pp. 4–7, 2011.

[238] B. Specification, "Version 1.1," *Includes: IMS Learning Resource Meta-data Information Model IMS Learning Resource Meta-data XML Binding Specification IMS Learning Resource Meta-data Best Practice and Implementation Guide Available at: www. imsproject. org*, 2001.

[239] S. Feldmann, K. Kyamakya, A. Zapater, and Z. Lue, "An indoor bluetooth-based positioning system: Concept, implementation and experimental evaluation." in *International Conference on Wireless Networks*, 2003, pp. 109–113.

[240] R. Bruno and F. Delmastro, "Design and analysis of a bluetooth-based indoor localization system," in *Personal wireless communications*. Springer, 2003, pp. 711–725.

[241] S. Hay and R. Harle, "Bluetooth tracking without discoverability," in *Location and context awareness*. Springer, 2009, pp. 120–137.

[242] S. K. Opoku, "An indoor tracking system based on bluetooth technology," *arXiv preprint arXiv:1209.3053*, 2012.

[243] C. Gomez, J. Oller, and J. Paradells, "Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology," *Sensors*, vol. 12, no. 9, pp. 11 734–11 753, 2012.

[244] Gimbal, "The gimbal store," Aug. 2015. [Online]. Available: https://store.gimbal.com

[245] Estimote, "Estimote: Real-world context for your apps," Aug. 2015. [Online]. Available: http://estimote.com/#jump-to-products

[246] A. Developer, "Getting started with ibeacon," 2014.

[247] J. Yang, Z. Wang, and X. Zhang, "An ibeacon-based indoor positioning systems for hospitals," 2015.

[248] Z. Chen, Q. Zhu, H. Jiang, H. Zou, Y. C. Soh, L. Xie, R. Jia, and C. Spanos, "An ibeacon assisted indoor localization and tracking system."

[249] S. A. Ortiz and L. M. Ortiz, "Systems and methods for tracking assets using associated portable electronic device in the form of beacons," Mar. 2014, uS Patent App. 14/194,953.

[250] M. Kouhne and J. Sieck, "Location-based services with ibeacon technology," in *Artificial Intelligence, Modelling and Simulation (AIMS), 2014 2nd International Conference on.* IEEE, 2014, pp. 315–321.

[251] Texas Instruments, "MSP430FR5969 launchpad development kit," Jul. 2015. [Online]. Available: http://www.ti.com/tool/MSP-EXP430FR5969

[252] Hardware Breakout, "Bluetooth low energy boosterpack for the launchpad,"

Aug. 2015. [Online]. Available: http://store.hardwarebreakout.com/index.php?route=product/product&product_id=65

[253] T. Bradley, "Pros and cons of bringing your own device to work," Dec. 2011. [Online]. Available: http://www.pcworld.com/article/246760/pros_and_cons_of_byod_bring_your_own_device_.html

[254] D. Manolakis, "Efficient solution and performance analysis of 3-d position estimation by trilateration," *Aerospace and Electronic Systems, IEEE Transactions on*, vol. 32, no. 4, pp. 1239–1248, Oct 1996.

[255] F. Thomas and L. Ros, "Revisiting trilateration for robot localization," *Robotics, IEEE Transactions on*, vol. 21, no. 1, pp. 93–101, Feb 2005.

[256] W. Murphy and W. Hereman, "Determination of a position in three dimensions using trilateration and approximate distances," *Department of Mathematical and Computer Sciences, Colorado School of Mines, Golden, Colorado, MCS-95*, vol. 7, p. 19, 1995.

[257] "OpenMRS Wiki Rosources - Demo Data," https://wiki.openmrs.org/display/RES/Demo+Data, accessed: 2015-08-17.

Vita

Michael Rushanan received his B.S. degree in Computer Science from the University of Baltimore County in 2009. He received his M.S.E degree in Computer Science and M.S. degree in Security Informatics from the Johns Hopkins University in 2015. He enrolled in the Computer Science Ph.D. program at Johns Hopkins University in 2011. He was inducted into Upsilon Pi Epsilon International Computer Science Honor Society in 2011 and served as president for two years. His research interests include systems security, health information technology security, privacy, and applied cryptography. His hobbies include embedded system design and implementation (e.g., Arduino and Raspberry Pi), mobile application development (e.g., iOS and Android), and programming.