Take Two Software Updates and See Me in the Morning: The Case for Software Security Evaluations of Medical Devices

Steven Hanna*, Rolf Rolles†, Andrés Molina-Markham‡, Pongsin Poosankam*§, Kevin Fu‡, Dawn Song*
* Computer Science Division, University of California Berkeley
‡ Department of Computer Science, University of Massachusetts Amherst
§ Computer Science Department, Carnegie Mellon University
† Unaffiliated

***Abstract*—Medical devices used for critical care are becoming increasingly reliant on software; however, little is understood about the security vulnerabilities facing medical devices and their software. To investigate this open question, we analyze the security of software that controls a modern Automated External Defibrillator (AED) used for treating cardiac arrhythmias. This report represents the first public *embedded software* security analysis of a medical device. We identify several software security vulnerabilities and discuss key insights and open challenges in improving software-controlled medical devices to be resistant to malware. We found the AED would accept counterfeit firmware updates. We did not locate any standard cryptographic controls. We conclude with recommendations and open challenges in securing medical devices.**[1]

I. Introduction

Life-critical medical devices increasingly contain significant embedded software responsible for safe and effective patient care. Devices range from life-sustaining implantable pacemakers to life-supporting devices such as drug infusion pumps, insulin pumps, and cardiac defibrillator monitors. Little is known about the state of affairs of the security of embedded medical device software itself. Recent research analyzes the wireless security of a medical device, but not the embedded software itself [16]. Thus, our present work assesses how software/firmware updates [5] and software security vulnerabilities are likely to impact the safety and effectiveness of computer-controlled medical devices. Furthermore, medical devices have the unique property that they must do everything they can to *fail open* in order to ensure that a life-critical device continues to operate even in the wake of an adverse event. Studying the susceptibility of medical devices to malware now is important because (1) software in medical devices is becoming increasingly complex; (2) more and more medical devices are becoming networked with wireless Internet connectivity; (3) more medical devices are evolving from electro-mechanical to software-controlled devices; and (4) analyzing security after a potential risk becomes a tangible threat would be too late for effective deployment of defensive technology.

Over the last three decades in the U.S. marketplace, software has played an increasing role both in the function of medical devices and as a cause for medical device recalls [12]. In the early 1980s, approximately 6% of recalls were due to computer software issues [20]. The U.S. Food and Drug Administration reports that software played a causal role in approximately 18% of the recalls for "510k-approved" devices between 2005 and 2009 [10, p.80]. Software is currently the third leading cause of recalls—just behind design flaws (23%) and manufacturing flaws (37%), but ahead of labeling flaws (12%). We analyzed records in the FDA database of Medical & Radiation Emitting Device Recalls—finding that between 2002–2010, there were 537 recalls of software-based medical devices affecting as many as 1,527,311 individual devices in the U.S. marketplace.[2]

Furthermore, today, many medical devices benefit from a safety blanket of physical isolation. A medical device isolated from other computing devices is less exposed to malware. However, a growing number of medical devices relying on wireless communication and Internet connectivity are exposed to a much larger surface of potential threats. Therefore, the best way to control the spread of medical malware is to focus on reducing the susceptibility of medical devices by strengthening software defense against malicious intent.

In order to assess the state of current security threats, we perform the first public software security analysis of a medical device, namely, an automatic external defibrillator. AEDs are responsible for evaluating a patient's heart rhythm, and if required, provide a shock to attempt to restore the patient's heart rhythm to normal. We analyzed the Cardiac Science G3 Plus AED model 9390A, an automatic external defibrillator manufactured since 2005, and the Windows-based utilities for updating the AED and configuring the device.

We demonstrate a set of vulnerabilities that makes the device potentially susceptible to malware. The discovered vulnerabilities include a buffer overflow, which leads to arbitrary code execution, cryptographic flaws in password protection, and a software update mechanism that accepts counterfeit firmware. These issues highlight the impending threat of malware, and in that light, we discuss a hypothetical construction of a worm.

We use these vulnerabilities to motivate the need for improved, effective defenses against malware, as well as to highlight open challenges in applying computer security principles to medical devices. Our assessment demonstrates real vulnerabilities in medical devices and their software and gives a first glimpse into the viability of malware that can be expected in software-based medical devices.

[1] This paper will be presented at the 2nd USENIX Workshop on Health Security and Privacy, San Francisco, CA, August 2011. Corresponding faculty authors: Prof. Kevin Fu (kevinfu@cs.umass.edu) and Prof. Dawn Song (dawnsong@cs.berkeley.edu).

[2] Some devices are affected by multiple recalls. An individual device is counted once for each recall.

II. Automated External Defibrillator (AED) Overview

In this section, we introduce automated external defibrillators and discuss why they represent a quintessential software-based medical device to investigate security. The term *defibrillator* refers to a broad class of devices that use large electrical shocks to treat cardiac arrhythmias that might otherwise lead to a fatal outcome. Defibrillators are divided into two types: *implantable* or *external*. While both types of defibrillators treat cardiac arrhythmias, the devices are radically different in design and purpose. One could draw the following analogy: implanted defibrillators are to mobile phones as external defibrillators are to public phone call boxes. The former are prescribed and tuned to a particular person whereas the latter are available for general use when you can find one. There are two further classifications of external defibrillators: *manual* or *automated*. Trained health care professionals may use manual external defibrillators to treat a wide range of arrhythmias. Our work analyzes the second class: automated external defibrillators (AEDs) that a person with limited medical training may use to treat a more limited (but common) number of cardiac arrhythmias such as ventricular fibrillation.

AEDs are significantly different from implantable cardiac defibrillators, both physically in terms of their hardware, software and connectivity, and in terms of risk. An AED is typically attached to the chest of the individual using two pads, and it delivers an electric shock to reestablish a normal heart rhythm. The conditions that can be treated through defibrillation, for example cardiac arrhythmias of ventricular fibrillation and ventricular tachycardia, generally require prompt attention in order to prevent severe brain damage or death. Studies have shown that the chances of survival improve if an individual receives defibrillation within 3-5 minutes of collapse [1]. For this reason, AEDs are widely available in airports, community centers, schools, government buildings, and other public locations. As of 2009, an estimated 1.5 million AEDs are in circulation worldwide [13], and industry expects an annual growth in sales of 9-12% [11], [8]. In 2005, over 192,400 AED units were sold in the U.S. marketplace; this represents approximately a ten-fold increase in AED sales from 1996–2005 [18].

The FDA has received more than 28,000 adverse event reports for external defibrillators between January 2005 and May 2010, including malfunctions, patient injuries, and deaths. As a result, the FDA has issued 68 recalls, 17 alone in 2009. In 2010, the FDA reported that 280,000 external defibrillators across 14 different models were susceptible to malfunction [9]. 16.2% of AED advisories were confirmed as resulting from software-related issues, with 6 advisories affecting 12,311 individual devices [18]. Only one category had a higher incidence (electrical issues at 21.6% of the causes of a recall) [18]. Given the importance of these devices in saving lives, the FDA began an initiative to promote the development of safer external defibrillators in 2010 [2].

In this paper, we study one particular model of AED, the Cardiac Science G3 Plus with model number 9390A-501. We decided to examine an Automated External Defibrillator (AED) because of its reliance on software updates to address an FDA recall. Depending on how the device is configured, the shock administered can be 150-300 Joules, which can be administered multiple times on one battery before the device requires a recharge. The G3 Plus is pictured in Figure 1.

Fig. 1. The Cardiac Science G3 Plus exploited to install our custom firmware. The AED displays DEVICE COMPROMISED.

III. Case Study

In this section, we analyze the Cardiac Science G3 Plus Automated External Defibrillator. In order to analyze the device, we used IDA Pro 5.6—an inexpensive, commercially available software package—to statically reverse engineer the device's update and diagnostic software, communication protocol, and firmware [3]. Our analysis was conducted using COTS hardware and software, and took upwards of 100 hours. During our analysis, we discovered that **4** vulnerabilities were readily prevalent from reversing the device and its software. We verify our analysis and show the gravity of software exploits on medical devices by executing concrete attacks and demonstrating their potential impact. Both the FDA and the manufacturer of the studied device were notified of our findings. One thing to note is that we explicitly chose not to provide a detailed set of steps for reproducing the attack; instead, we provide the minimal evidence for a security expert to verify our results.

A. Analyzing the G3 Plus

Performing a security analysis of the G3 Plus necessitated reverse engineering device operation and controlling software because it is a closed-platform device, with proprietary hardware, firmware and software.

We discovered that the AED's CPU implements a 16-bit x86 ISA by examining the internals of the AED with CPU processor manuals. From this, we applied static reverse engineering techniques to the *firmware*.

The device comes with three software packages: *MDLink*, *RescueLink* and *AEDUpdate*, responsible for programming device parameters (like shock value), collecting post-cardiac arrest reports, and updating the AED, respectively. We focus our analysis on the device firmware (252 KB), *AEDUpdate* (8 MB), and *MDLink* (680 KB) because of their potential

impact on correct device operation; they are responsible for communicating with the host computer, updating firmware, and programming device parameters.

B. Vulnerabilities in the Cardiac Science G3 AED

Below we detail the key findings of our investigation into the security of the AED and its software. First, we discuss the bugs and vulnerabilities we found in the *MDLink* and *AEDUpdate* software. Following that, we discuss how we can arbitrarily change the G3 Plus's firmware.

***Vulnerability 1**: AEDUpdate integer/buffer overflow.* Initially, *AEDUpdate* sends a packet over the COM port to the AED and then receives a response. In order to verify the vulnerability, we spoofed packets coming from the COM port such that we were able to control the data that *AEDUpdate* received from the device; we thereby triggered the vulnerability and achieved arbitrary code execution.

The *AEDUpdate* program counts the number of '0'-characters in the response and then performs a `memcpy` based on the number of zeros found. Specifically, the program expects to find no more than 8 zeros, so it uses the expression `8-num_zeros` to calculate the size of the buffer to copy. However, if there are more than 8 zeros, the *size* parameter underflows and results in attempting to copy around 4.3 billion bytes of data into a 16-byte buffer, causing the program to throw an exception. In order to exploit this vulnerability, we need to overwrite the most recently registered exception handler with our own and cause this exception to occur before the corrupted return address is checked, which allows us to redirect program flow into arbitrary code.

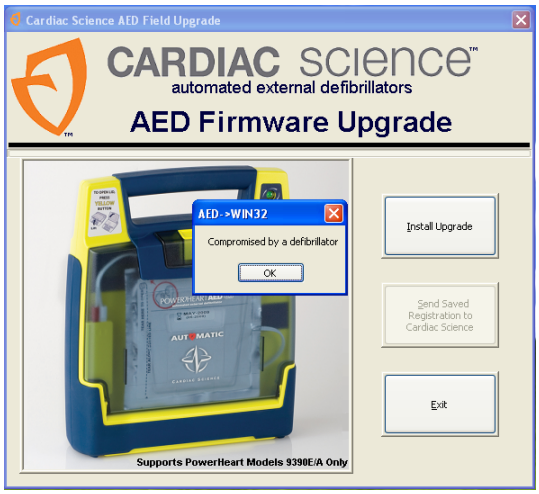

Fig. 2. *AEDUpdate* buffer overflow. Executed code includes a message box showing the potential flow of the vulnerability from the AED (if the firmware were replaced) to the software.

In Windows XP SP2, nearly every module linked into a binary uses Safe Structured Exception Handling (SSEH). This means that the exception handler must be registered with the operating system, and if during execution it is discovered that it is not registered, the process is terminated by the OS [4]. `oledlg.dll` was the only DLL imported into the binary that did not employ SSEH. Using this, we *overwrote* the exception handler on the stack with a subverted one, forcing the program to execute our payload when an exception was thrown[3]. Figure 2 shows *AEDUpdate* being exploited.

Automatic Vulnerability Discovery. Although the AEDUpdate integer/buffer overflow vulnerability was initially found through manual analysis, we later verified that, given appropriate program entry points obtained through reverse engineering, it could also be automatically found within minutes using our vulnerability discovery tool called BitFuzz [7], [19], [6]. BitFuzz applies dynamic symbolic execution [7], [14] to generate test inputs that explore different paths in the program, including paths that contain a crash. When a crash is found, the tool provides a crash report that consists of a crashing execution trace and a crashing input. We run BitFuzz in Linux on a 3.0Ghz Dual Core Intel CPU with 4GB of RAM. After running for 16 minutes and 18 seconds, it successfully generates a crash report that confirms the existence of the integer/buffer overflow vulnerability in *AEDUpdate*.

***Vulnerability 2**: Weak password authentication scheme.* The *MDLink* software has individual user profiles protected by a password. However, the password file is stored on the local hard drive, and anyone with privileges can simply delete it in order to circumvent the protection mechanisms or circumvent it completely by installing the software fresh on a machine that does not have access restrictions. Furthermore, after we reverse engineered the *MDLink* authentication mechanism, we discovered that the password was obfuscated using a simple `XOR` scheme. From there, we extracted the scheme and wrote a small utility to arbitrarily change or recover a user's password.

***Vulnerability 3**: Credentials stored in plaintext.* While examining *AEDUpdate*, we came across functionality that seemed to upload a file to a remote FTP server. Upon further investigation, we found that the *AEDUpdate* stores the address of the FTP server, username, and password in *plain text* within the update software. When warranted, due either to a failed upgrade or to some other diagnostic problem, the diagnostic file is sent to Cardiac Science. This private user information is collected when a firmware upgrade is started by *AEDUpdate* and includes contact information, serial number, and the firmware version—all potentially available to anyone who accesses the login credentials.

***Vulnerability 4**: Improper use of weak CRC as digital signature.* We discovered that the software begins the update process by sending a cyclic redundancy check (CRC) value of the new firmware. Once transferred, the device calculates the CRC of the received firmware and allows the current firmware to be replaced only if the values match. In order to install custom firmware, we populated its CRC verification table with new values calculated over the entirety of our custom image. To obtain the precise integrity check employed by the AED, we extracted the code responsible for CRC calculation and verification from *AEDUpdate*, which we used to generate a new CRC to convince the AED that our image was legitimate.

This vulnerability allows malicious modifications that could lead to device failure, patient harm, and financial burden. These attacks include: (1) disabling or falsifying integrity checking so

[3]For instance, by corrupting a pointer on the stack.

that when a component fails, the device will pass verification; (2) setting arbitrary shock protocols and shock strength; (3) failing to administer shocks despite the device's appearance of normal operation; (4) bricking the device; (5) destroying the AED's integrity and audit trail; (6) resource exhaustion and DoS by exhausting the device's battery; and (7) executing a conditional payload, depending on the date or time.

C. Impact

Besides the direct security implications of the vulnerabilities discovered, perhaps less obviously, the resulting vulnerabilities—namely *Vulnerability 1* and *Vulnerability 4*—can be combined to create a hybrid piece of malware capable of infecting both AEDs and PCs. Consider the scenario where an attacker replaces the firmware of an AED with custom malware designed to exploit the *AEDUpdate* buffer overflow. This would allow the AED to execute arbitrary code. An attacker could create a worm that infects the host computer and subsequently other AEDs connected to the host. This type of attack could have the purpose of damaging other equipment within an organization or obtaining private data stored in systems other than the affected medical devices. Risks like these punctuate the necessity of designing medical devices with security and privacy in mind from the early stages of requirements specification and throughout the entire software development lifecycle.

IV. Improving Software-based Medical Device Security

In order to improve the security of medical devices, we discuss four key principles which, based on our experiences, mitigate the attacks outlined in this paper against the G3 Plus. Most importantly, we discuss future areas of exploration—including defenses—and stress that solutions must work effectively in the context of patient care.

A. Cryptographically secure device updates are needed to ensure firmware integrity

An end-to-end secure device update prevents attackers from replacing medical device firmware. Recall that the G3 Plus' update mechanism relies on a cyclic redundancy check to detect errors in the transmission of data to the device but lacks a mechanism to guarantee firmware authenticity. We propose that in light of this, devices like the AED with on-device firmware should be designed only to accept signed firmware.

However, the solution to this is not obvious. In addition to having strict power constraints, a complete solution to the problem of providing secure software updates for embedded devices has not yet been found. Cryptographic approaches would create administrative challenges of key management. Attempts have been made to extend a Trusted Computing Platform for software-supported and hardware-supported updates [17]. However, solutions for applying updates to software on medical devices remain poorly understood.

B. Device telemetry verified for integrity and authenticity

Software or devices that rely on medical device telemetry should treat input data as adversarial and should design the device's software to handle a wide range of malformed or malicious inputs. As we saw with the G3 Plus, the update software did not correctly handle the case where a packet was malformed. Accurate modeling of device protocols and testing are the obvious choices to ensure correct operation. Furthermore, greater assurance of correct device operation is attainable by implicitly not trusting telemetry and programmatically employing rigorous sanity checks on inter-device communication. Good programming practices coupled with rigorous static and dynamic analysis checking for security vulnerabilities mitigates many threats, and in the case of the G3 Plus, could have prevented the overflow that led to arbitrary code execution.

C. Passwords should be cryptographically secure and easily managed

The two password mechanisms we found in the program relied on obscuring the location or text of the password, which made them easily recoverable. In medical devices, where passwords potentially provide access to private data or life-critical functionality, they must be securely protected. However, traditional protection mechanisms are muddied by the added complexity of managing passwords in an environment where many people may need to access the same device or software functionality. Furthermore, password revocation and access control add to the intricacy of a secure design.

D. Defenses and updates must be weighed with their risk to the patient

Mechanisms that ensure and maintain device integrity must be weighed with their potential risk to the patient. With respect to updates: What is the purpose of the update? How might it change device operation? Is the update crucial to patient safety? In the case of defending against modification, if a device's firmware has been modified, it may be necessary to identify with certainty the severity of the potential integrity flaw. In other words, the process of verifying that a device has not been tampered with should not make it easier for a malicious party to launch a denial of service attack on a device with critical functionality.

Most notably, we point out the general principle that medical devices, in most cases, need to *fail open*. This is particularly true when emergency access to medical devices is needed. For instance, if a patient with an implantable defibrillator collapses, the treating doctor would need to query the device for information in order to treat the patient. Simply denying access to the doctor is unacceptable, as it puts security ahead of accessibility, which is in direct contention with the usability of the device [15].

This presents a major design challenge for device developers because they are faced with the difficult problem of providing security while judiciously restricting accessibility.

E. Recommendations

Despite the risks outlined in this paper, at this current time, we recommend continued use of AEDs because of their potential to perform lifesaving functions, rather than to disable them for a currently unmeasured attack potential. This does not undermine the severity of the attacks discussed, but serves to

underscore the potential severity as opposed to the the current threat level.

Ensure the update machine is secure.

The computer used for updating and configuring AEDs should be a machine devoted to this purpose and not used for general purpose tasks, such as reading email, printing prescriptions, or obtaining general information from the Internet. For example, when updating a device is necessary, each software update could be started from a fresh virtual machine that is destroyed after each update. In the same way a syringe is freshly unwrapped for each patient and discarded afterwards, the mechanism used to update an AED would not be shared between devices.

Follow FDA guilelines and advisories.

FDA protocol for updating and interacting with the device should be followed. Of note, the FDA begun an external defibrillator improvement initiative which seeks to improve the design and testing of AEDs to offer enhanced ways to notify consumers and react appropriately depending on the severity of the threat.

Remain vigilant.

Owners and maintainers of AEDs must remain vigilant in ensuring that AEDs are safe including: monitoring physical access to the device, routinely updating afflicted devices, and monitoring advisories released about the device. This provides a proactive approach to deterring malicious activity, and constant attention to adverse event reports allows the maintainers to react appropriately and promptly to an advisory.

V. Conclusion

Software security remains an afterthought for medical device design. Our case study of a popular Automated External Defibrillator (AED) identified security flaws in both the embedded software and the COTS software update mechanism. Although our research pertains to a single manufacturer, other devices from different manufacturers may also be susceptible to the same risks. For that reason, manufacturers of medical devices containing software should have plans for assessing specific security risks, detecting security compromises, and recovering from computer security incidents—especially if the manufacturer plans to use wireless communication or Internet connectivity that would increase the device exposure to the risks of malicious software.

VI. Acknowledgments

We thank Dr. Daniel Kramer, Dr. Matthew Reynolds, and Quinn Stewart for their feedback. This research is supported by NSF CNS-0831244 and CNS-0842695, a Sloan Research Fellowship, an NSF graduate research fellowship, the Armstrong Fund for Science, and Cooperative Agreement No. 90TR0003/01 from the Department of Health and Human Services. Its contents are solely the responsibility of the authors and do not necessarily represent the official views of the DHHS or NSF.

References

[1] Cardiopulmonary Resuscitation (CPR) Statistics. American Heart Association. http://www.americanheart.org/presenter.jhtml?identifier=4483.

[2] External defibrillator improvement initiative. FDA, Center for Devices and Radiological Health. http://www.fda.gov/downloads/MedicalDevices/ProductsandMedicalProcedures/CardiovascularDevices/ExternalDefibrillators/UCM233824.pdf.

[3] IDA Pro. Hex-Rays SA, http://www.hex-rays.com/idapro/.

[4] Safeseh. Microsoft, http://msdn.microsoft.com/en-us/library/9a89h429(VS.80).aspx.

[5] A. Bellissimo, J. Burgess, and K. Fu. Secure Software Updates: Disappointments and New Challenges. In *Proceedings of USENIX Hot Topics in Security (HotSec)*, July 2006.

[6] BitBlaze: Binary analysis for computer security. http://bitblaze.cs.berkeley.edu/.

[7] J. Caballero, P. Poosankam, S. McCamant, D. Babic, and D. Song. Input generation via decomposition and re-stitching: Finding bugs in malware. In *Proceedings of the 17th ACM Conference on Computer and Communication Security*, Chicago, IL, October 2010.

[8] P. S. Chan, H. M. Krumholz, J. A. Spertus, P. G. Jones, P. Cram, R. A. Berg, M. A. Peberdy, V. Nadkarni, M. E. Mancini, and B. K. Nallamothu. Automated External Defibrillators and Survival After In-Hospital Cardiac Arrest. *JAMA : The Journal of the American Medical Association*, 304(19):2129–2136, November 2010.

[9] FDA. FDA Warns Users about Faulty Components in 14 External Defibrillator Models, April 2010. http://www.fda.gov/NewsEvents/Newsroom/PressAnnouncements/ucm209874.htm.

[10] CDRH preliminary internal evaluations — volume I: Preliminary Report and Recommendations, August 2010. http://www.fda.gov/downloads/AboutFDA/CentersOffices/CDRH/CDRHReports/UCM220784.pdf.

[11] Frost and Sullivan. US external defibrillator markets, 2007. http://www.researchandmarkets.com/reports/575550/u_s_external_defibrillator_markets_2007 .htm as cited in Chan et al., Automated External Defibrillators and Survival After In-Hospital Cardiac Arrest, JAMA 19(304), p.3125, November 2010.

[12] K. Fu. Trustworthy medical device software. In *Public Health Effectiveness of the FDA 510(k) Clearance Process: Measuring Postmarket Performance and Other Select Topics: Workshop Report*, Washington, DC, 2011. Institute of Medicine (IOM), National Academies Press.

[13] GlobalData. Global automated external defibrillators (AED) market: Demand to drive growth, June 2009. As cited in Derek Smith, Philips Healthcare, MDR Reporting Factors Over the Past 5 Years, FDA Public Workshop on External Defibrillators, December 2010, http://www.fda.gov/downloads/MedicalDevices/NewsEvents/WorkshopsConferences/UCM249337.pptx.

[14] P. Godefroid, N. Klarlund, and K. Sen. Dart: Directed automated random testing. In *In Programming Language Design and Implementation (PLDI*, 2005.

[15] D. Halperin, T. S. Heydt-Benjamin, K. Fu, T. Kohno, and W. H. Maisel. Security and Privacy for Implantable Medical Devices. *IEEE Pervasive Computing*, 7(1):30–39, 2008.

[16] 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 *Proceedings of the 29th Annual IEEE Symposium on Security and Privacy*, May 2008.

[17] U. Kahn, K. Kursawe, S. Lucks, A.-R. Sadeghi, and C. Stble. Secure Data Management in Trusted Computing. In J. R. Rao and B. Sunar, editors, *Cryptographic Hardware and Embedded Systems CHES 2005*, volume 3659 of *Lecture Notes in Computer Science*, pages 324–338. Springer Berlin / Heidelberg, 2005.

[18] J. S. Shah and W. H. Maisel. Recalls and Safety Alerts Affecting Automated External Defibrillators. *JAMA: The Journal of the American Medical Association*, 296(6):655–660, August 2006.

[19] D. Song, D. Brumley, H. Yin, J. Caballero, I. Jager, M. G. Kang, Z. Liang, J. Newsome, P. Poosankam, and P. Saxena. BitBlaze: A new approach to computer security via binary analysis (invited keynote). In *International Conference on Information Systems Security (ICISS)*, pages 1–25, Hyderabad, India, December 2008.

[20] D. R. Wallace and D. R. Kuhn. Failure Modes in Medical Device Software: An Analysis of 15 Years of Recall Data. *International Journal of Reliability Quality and Safety Engineering*, 8:351–372, 2001.